Exploring SAS® Viya®

Data Mining and Machine Learning

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *Exploring SAS® Viya®: Data Mining and Machine Learning*. Cary, NC: SAS Institute Inc.

Exploring SAS® Viya®: Data Mining and Machine Learning

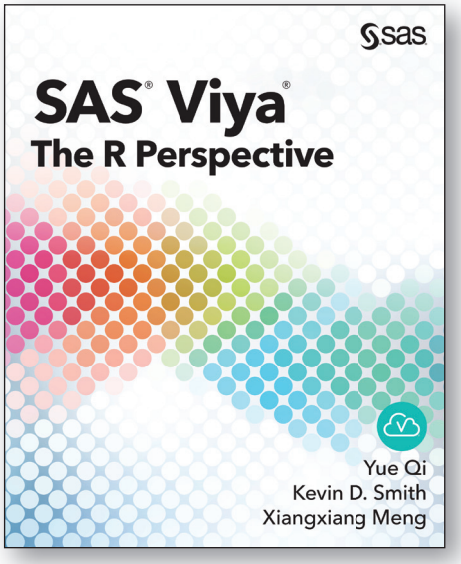
978-1-64295-588-0 (Paperback)
978-1-64295-587-3 (Web PDF)

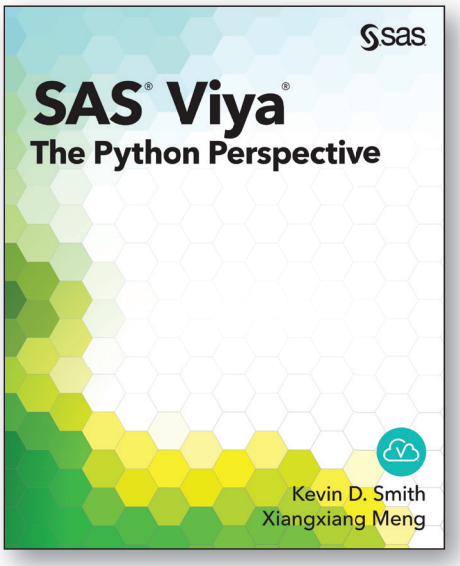
SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

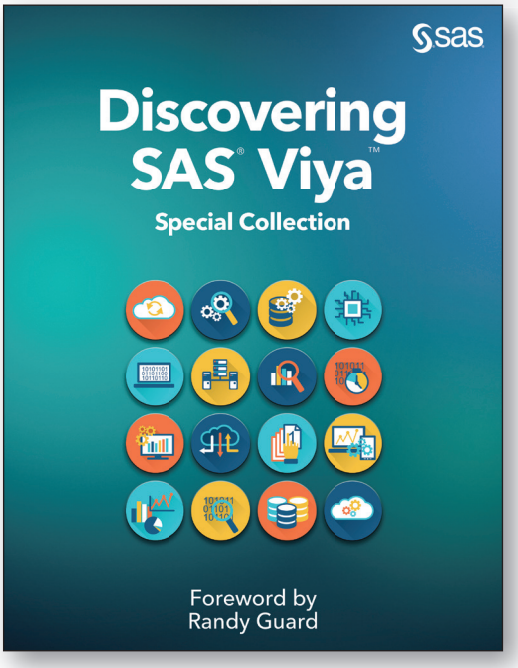
August 2019

Contents

For more information on this topic, check out the books below in the SAS® bookstore:







For 20% off these e-books, visit **sas.com\books** and use code **WITHSAS20.**

sas.com/books
for additional books and resources.

sas
THE POWER TO KNOW.®

About This Book

What Does This Book Cover?

Data mining is the process of finding anomalies, patterns and correlations within large data sets to predict outcomes. Using a broad range of techniques, you can use this information to increase revenues, cut costs, improve customer relationships, reduce risks and more. *Machine learning* is a method of data analysis that uses data mining techniques to automate analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look. SAS® Visual Data Mining and Machine Learning is a powerful analytical solution that enables you to solve your most complex problems in a single, integrated solution powered by SAS® Viya®.

SAS Viya is an open analytics platform that can handle any data type, volume, or speed. A cloud-enabled, in-memory analytics engine, it is elastic, scalable, and fault-tolerant. It contains a standardized code base that supports programming in SAS and other languages, such as Python, R, Java and Lua. In addition, it can deploy seamlessly to any infrastructure or application ecosystem with support for cloud, on-site, or hybrid environments. The high-performance processing power of SAS Viya is provided by SAS Cloud Analytics Services (CAS). CAS is an in-memory engine that can dramatically accelerate data management and analytics with SAS.

In this book, we will explore some of the features of SAS Visual Data Mining and Machine Learning, including:

- Programming in SAS® Studio
- Programming in the Python interface
- Data mining and machine learning tasks
- New, advanced data mining and machine learning procedures available in SAS Viya
- Pipeline building in Model Studio
- Model building and comparison in SAS® Visual Analytics

The extreme flexibility of SAS Visual Data Mining and Machine Learning means that users of all skill levels can visually explore data on their own in these programs while drawing on powerful in-memory technologies for faster analytic computations and discoveries. These programs offer an easy-to-use self-service environment that can scale on an enterprise-wide level. You can manually program with custom code or use the features in SAS Studio, Model Studio, and SAS Visual Analytics to automate your data manipulation and modeling.

The content in this book is based on SAS® Viya® Enablement, a free course available from SAS Education. This book covers how to explore, train, and model data in the SAS Studio, Model Studio, and SAS Visual Analytics environments. This book only begins to show what these programs can do. More information is available in the documentation for specific procedures and features.

If you want to learn more about the features of SAS Viya, how to load data into the CAS server, how to write new code, and how to perform data management and administrative tasks, then you might be interested in reading *Exploring SAS® Viya®: Programming and Data Management*. You might also want to read *Exploring SAS® Viya®: Visual Analytics, Statistics, and Investigations*, which introduces how to use data mining and machine learning tasks in SAS Studio and SAS Visual Analytics, as well as how to access all of your enabled features in SAS Viya and configure your home page.

Is This Book for You?

SAS Data Mining and Machine Learning software is designed for anyone in your organization who wants to use and derive insights from data – data scientists, business analysts, management, and other analytics professionals. From data management to model development and deployment, everyone works in the same integrated environment.

SAS Visual Data Mining and Machine Learning automatically generates insights that enable you to identify the most common variables across all models, the most important variables selected across models, and assessment results for all models. Natural language generation capabilities are used to create a project summary written in simple language, enabling you to easily interpret reports. Analytics team members can add project notes to the insights report to facilitate communication and collaboration among team members.

Even if you don't know SAS code, SAS Visual Data Mining and Machine Learning lets you embed open-source code within an analysis and call open-source algorithms seamlessly within a Model Studio flow. This facilitates collaboration across your organization because users can program in the language of their choice. You can also take advantage of SAS Deep Learning with Python (DLPy), our open-source package on GitHub, to use Python within Jupyter notebooks to access high-level APIs for deep learning functionalities.

What Should You Know about the Examples?

The content in this book is based on SAS® Viya® Enablement, a free course available from SAS Education. You can follow along with the examples in real time by watching the videos if you prefer.

This book includes tutorials for you to follow to gain hands-on experience with SAS Studio, Model Studio, and SAS Visual Analytics. Wherever possible, the source of the sample data or similar data is provided in a link. Some features shown might be available only if your site has licensed that feature in SAS Viya. Therefore, the options in your version of SAS might look different.

We Want to Hear from You

Do you have questions about a SAS Press book that you are reading? Contact us at saspress@sas.com.

SAS Press books are written *by* SAS Users *for* SAS Users. Please visit sas.com/books to sign up to request information about how to become a SAS Press author.

Learn about new books and exclusive discounts. Sign up for our new books mailing list today at https://support.sas.com/en/books/subscribe-books.html.

Chapter 1: Programming in SAS Studio and the Python Interface

Introduction

SAS Viya is a new product offering from SAS that showcases a rich set of data mining and machine learning capabilities that run on a robust, in-memory distributed computing infrastructure. This product provides a single environment for data scientists to perform necessary tasks associated with data preparation, feature engineering, model training, assessment, and deployment.

With SAS Viya, you can access SAS analytics through programmatic actions written in SAS or through interfaces with other programming languages such as Python, Java, and Lua. In this section, we will look at some of the capabilities of SAS Visual Data Mining and Machine Learning that can be programmed manually using SAS Studio, then look at another example using the Python interface.

The main interface for SAS Viya is SAS Studio, a web-based user interface that offers an array of utilities and conveniences for composing your applications in the SAS programming language. First, let's look at some of the capabilities of SAS Viya through an example of programming in SAS Studio.

Programming in SAS Studio

In this section, we will look at a simple, start-to-finish machine learning solution that can be programmed manually in SAS Viya through the SAS Studio interface. This code loads and prepares the data, builds and compares three models, and provides score code for new data. In the next chapter, we will look at how some of these steps can be automated using the Data Mining and Machine Learning tasks in SAS Studio.

The data source used in this example contains a list of donors to an organization. This data source has a target variable, TARGET_B, which is a binary variable that has the value 1 for a person who has donated during a mailing. The example data is available to download from the SAS Visual Data Mining and Machine Learning documentation. See the link in the Resources section.

Load and Explore Data

Program 1.1 shows a program that loads the data into CAS, then runs PROC CARDINALITY to produce an output table that contains information about the levels of nominal variables, basic statistics, and an accounting of missing values.

Program 1.1: Loading and Exploring Data

```
/*****************************/
/* Setup Environment         */
/*****************************/

options cashost="&cashost." casport=&casport.;❶
libname mylib "/opt/sasinside/DemoData";
%let outdir=/opt/sasinside/DemoData;

libname mycaslib cas "/opt/sasinside/DemoData";❷
/*****************************/
/* Copy local data into CAS  */
/*****************************/

data mycaslib.donor_raw_data; ❸
  Set mylib.donor_raw_data;
run;

/*****************************/
/* Data Exploration          */
/*****************************/
proc cardinality data=mycaslib.donor_raw_data outcard=mycaslib.donor_raw card;
run; ❹

proc print data=mycaslib.donor_raw_card(where(_NMISS_>0));
run;

data donor missing;
  set mycaslib.donor_raw_card(where=(_nmiss_>0) keep=_varname_ _nmiss_ _noobs_);
  _percentmiss_ = (_nmiss_/_noobs_)*100;
  Label _percentmiss_ = 'Percent Missing';

proc sgplot data=donor_missing;
  vbar _varname_ / response=_percentmiss_ datalabel categoryorder=respdesc;
run; ❺
```

❶ As previously mentioned, SAS Viya runs on an in-memory distributed computing infrastructure, which is called CAS. So the first things we need to do in Program 1.1 are to specify the host and port information to the CAS server and start a session, which creates a workspace for our code to run in. In the CAS environment, data must be loaded in a CAS data set to allow efficient distributed management of the data.

❷ Here we specify a library location within our current session.

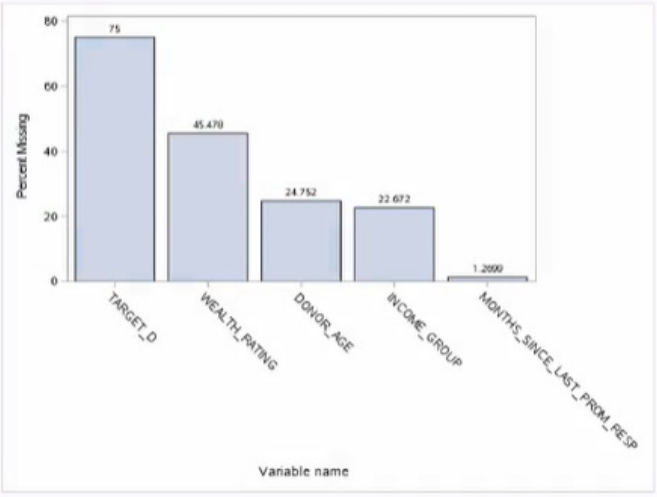
❸ Here we load a data set into CAS from an existing SAS data set on our local disk. There are numerous options for loading data from various file formats and data sources, including different data management systems, such as Hadoop and Amazon S3.

❹ As with any data analysis project, we start by examining the data regarding the nature of the variables through an analysis of all of the values. Executing PROC CARDINALITY produces an output table that contains information about the levels of nominal variables, basic statistics, and an accounting of missing values.

❺ Here we are specifically printing a table of all variables with missing values so that we can deal with them properly. The results of the PROC PRINT and PROC SGPLOT are shown in Output 1.1.

Output 1.1: Results of Program 1.1

Obs	_VARNAME_	_FMTWIDTH_	_TYPE_	_RLEVEL_	_ORDER_	_MORE_	_CARDINALITY_	_NOBS_	_SUMFREQS_	_NMISS_	_MISSFMT_	_VISIBLE_	_MIN_	_MAX_	_MEAN_
1	TARGET_D	12	N	INTERVAL	ASC	Y	20	19372	4843	14529	.	87.53	1	200	15.624344415
2	DONOR_AGE	12	N	INTERVAL	ASC	Y	20	19372	14577	4795	.	27.19	0	87	58.919056559
3	INCOME_GROUP	12	N	CLASS	ASC	N	7	19372	14980	4392	.	100	1	7	3.0076433912
4	WEALTH_RATING	12	N	CLASS	ASC	N	10	19372	10562	9810	.	100	0	9	5.0053967062
5	MONTHS_SINCE_LAST_PROM_RESP	12	N	INTERVAL	ASC	Y	20	19372	19126	246	.	66.06	-12	36	19.038899927



Partition Data

Armed with knowledge about the nature of our data from Output 1.1, we next use PROC PARTITION to partition the data set into training and validation subsets, specifying the target of interest in our modeling problem to ensure appropriate stratification.

Program 1.2: Partitioning into Training and Validation

```
proc partition data=mycaslib.donor_raw_data partition samppct=70;
  by target_b;
  output out=mycaslib.donor_raw_partind copyvars=(_ALL_);
run;
data mylib.donor_raw_partind;
  set mycaslib.donor_raw_partind;
run;
```

Program 1.2 creates a new data set with a partition index, as shown in Output 1.2. Note that some modeling procedures have partitioning as an integrated option, so doing this up front might not always be necessary.

Output 1.2: Results of Program 1.2

The PARTITION Procedure

Stratified Sampling Frequency

Index	Target Variable Indicates for Response to 97NK Mailing	Number of Obs	Number of Samples
0	0	14529	10170
1	1	4843	3390

Output CAS Tables

CAS Library	Name	Number of Rows	Number of Columns
CASUSER(viyauser)	DONOR_RAW_PARTIND	19372	51

Impute Missing Values

Because we have missing values, we next run PROC VARIMPUTE in Program 1.3 to impute the missing values through any of a number of methods. In this case, we are simply using the median. This creates score code that can then be run against the original data set or any future data set to impute the missing values, as shown in Output 1.2.

We can also see by running PROC CONTENTS that a new variable, IM_DONOR_AGE, now represents the donor age in our data set, as shown in Output 1.3.

Program 1.3: Imputation

```
proc varimpute data=mycaslib.donor_raw_partind;
  input donor_age /ctech=median;
  code file="&outdir./impute1.sas";
  output out=mycaslib.donor_raw_partind copyvars=(_ALL_);
run;

proc contents data=mycaslib.donor_raw
```

Output 1.3: Results of Program 1.3

The VARIMPUTE Procedure

Imputation Requests

Imputation Method	Number of Variables
Median	1

Imputation Information

Variable	Imputation Method	Result Variable	N	Number of Missing	Imputed Value
DONOR_AGE	Median	IM_DONOR_AGE	14677	4795	60

The CONTENTS Procedure

Data Set Name	MYCASLIB.DONOR_RAW_PARTIND	Observations	19372
Member Type	DATA	Variables	52
Engine	CAS	Indexes	0
Created	10/12/2016 11:30:02	Observation Length	416
Last Modified	10/12/2016 11:30:02	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	wlatin1 Unicode (UTF-8)		

Engine/Host Dependent Information

Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
45	CARD_PROM_12	Num	8	
9	CLUSTER_CODE	Char	2	
3	CONTROL_NUMBER	Char	8	
5	DONOR_AGE	Num	8	
11	DONOR_GENDER	Char	3	
49	FILE_AVG_GIFT	Num	8	
50	FILE_CARD_GIFT	Num	8	
28	FREQUENCY_STATUS_97NK	Num	8	
10	HOME_OWNER	Char	3	
52	IM_DONOR_AGE	Num	8	
12	INCOME_GROUP	Num	8	
6	IN_HOUSE	Num	8	
44	LAST_GIFT_AMT	Num	8	
40	LIFETIME_AVG_GIFT_AMT	Num	8	
36	LIFETIME_CARD_PROM	Num	8	
38	LIFETIME_GIFT_AMOUNT	Num	8	
39	LIFETIME_GIFT_COUNT	Num	8	
41	LIFETIME_GIFT_RANGE	Num	8	
42	LIFETIME_MAX_GIFT_AMT	Num	8	
43	LIFETIME_MIN_GIFT_AMT	Num	8	
37	LIFETIME_PROM	Num	8	
17	MEDIAN_HOME_VALUE	Num	8	
18	MEDIAN_HOUSEHOLD_INCOME	Num	8	

Because we will need to provide lists of variables as inputs in some of the procedures, it is convenient to define macro variables to hold this information as shown in Program 1.4 for class in interval variables.

Program 1.4: Macro Variable Definition

```
%let class_var=in_house urbanicity ses cluster_code home_owner donor_gender
    income_group published_phone overlay_source wealth_rating pep_star
    recency_status_96NK frequency_stsatus_97NK
%let interval_var=months_since_origin im_donor_age mor_hit_rate median_home_value
    median_household_income pct_owner_occupied per_capita_income pct_attribute1
    pct_attribute2 pct_attribute3 pct_attribute4 recent_star_status
    recent_response_prop recent_avg_gift_amt recent_card_respone_prop
    recent_avg_card_gift_amt recent_response_count recent_card_response_count
    months_since_last_prom_resp lifetime_card_prom lifetime_prom lifetime_gift_amount
    lifetime_gift_count lifetime_avg_gift_amt lifetime_gift_range
    lifetime_max_gift_amt lifetime_min_gift_amt last_gift_amt card_prom_12
    number_prom_12 months_since_last_gift months_since_first_gift file_avg_gift
    file_card_gift;
```

Variable Selection

Before proceeding to build models, we can run PROC VARRREDUCE to perform supervised variable selection that identifies the most important features relative to our specified target.

Program 1.5: Variable Selection

```
proc varreduce data=mycaslib.donor_raw_partind technique=dsc
/*Discriminant analysis for class target*/
  class target_b &class_var.;
  reduce supervised target_b=&class_var. &interval_var. /maxeffects = 20;
  ods output selectionsummary=summary;
run;

data out_iter (keep=Iteration VarExp Base Increment Variable);
  set summary;
  Increment=dif(VarExp);
  If Increment ='.' Then
    Increment=0;
  Base=VarExp - Increment;
run;

proc transpose data=out_iter_trans;
  label _NAME_='Group';
  by _NAME_;
run;

title "Variance Explained by Interation";

proc sgplot data=out_iter_trans;
  yaxis label="Variance Explained";
  vbar Iteration / response=COL1 group=_NAME_;
run;
```

Output 1.5: Partial Results of Program 1.5

Selected Effects

Number	Selected Variable	Variable Type
1	RECENT_CARD_RESPONSE_PROP	INTERVAL
2	MONTHS_SINCE_LAST_GIFT	INTERVAL
3	MEDIAN_HOME_VALUE	INTERVAL
4	RECENT_AVG_GIFT_AMT	INTERVAL
5	INCOME_GROUP	CLASS
6	MONTHS_SINCE_FIRST_GIFT	INTERVAL
7	SES	CLASS
8	CLUSTER_CODE	CLASS
9	NUMBER_PROM_12	INTERVAL
10	FREQUENCY_STATUS_97NK	CLASS
11	LIFETIME_MAX_GIFT_AMT	INTERVAL
12	PCT_ATTRIBUTE4	INTERVAL
13	DONOR_GENDER	CLASS
14	MEDIAN_HOUSEHOLD_INCOME	INTERVAL
15	PEP_STAR	CLASS
16	RECENT_CARD_RESPONSE_COUNT	INTERVAL
17	RECENCY_STATUS_96NK	CLASS
18	LIFETIME_AVG_GIFT_AMT	INTERVAL
19	WEALTH_RATING	CLASS
20	MONTHS_SINCE_ORIGIN	INTERVAL

Output 1.5 indicates the top 20 variables that are identified as significant and should be used as inputs for modeling.

Model Building

Now we are ready to build some models. We use PROC LOGSELECT to create a logistic regression model in Program 1.6.

Program 1.6: Logistic Regression Modeling

```
proc logselect data=mycaslib.donor_raw_partind;
  class target_b &class_var.;
  model target_b (event='1')=&class_var. &interval_var.;
  selection method=forward  /*(choose=validate stop=validate)*/
  partition rolevar=_partind_(train='1' validate='0')
  code file="&outdir./logselect1.sas";
*   output data=mycaslib.logselect_scored copyvars=(_ALL_);
  ods output FitStatistics=fitstats(rename=(Ntrees=Trees));
run;

/* Score the data using the generated model */
data mycaslib.donor_scored_forest;
  set mycaslib.donor_raw_partind;
  %include "&outdir./forest1.sas";
  p_target_b0=1-p_target_b;
run;
```

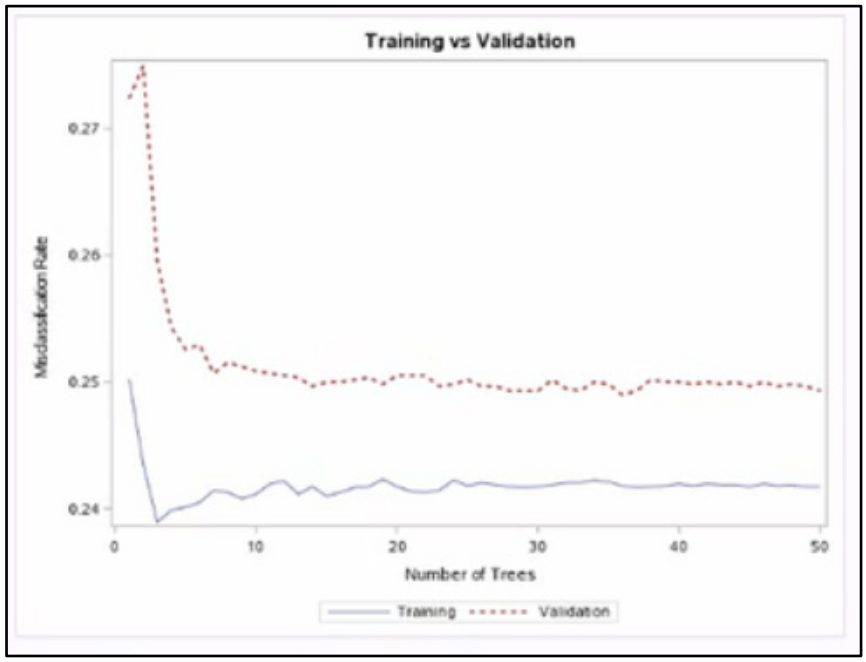
Next, we run PROC FOREST to generate a random forest model.

Program 1.7: Random Forest Modeling

```
proc forest data=mycaslib.donor_raw_partind ntrees=50 intervalbins=20 minleafsize=5;
  input &interval_var. / level = interval;
  input &class_var. / level = nominal;
  target target_b / level = nomial;
  partition rolevar=_partind_ (train='1' validate='0');
  code file="&outdir./forest1.sas";
*   output data=mycaslib.forest_scored copyvars=(_ALL_);
  ods output FitStatistics=fitstats(rename=(Ntrees=Trees));
run;

/*Score the data using the generated model */
```

```
data mycaslib.donor_scored_forest;
  set mycaslib.donor_raw_partind;
  %include "&outdir./forest1.sas";
run;

/* create data set from forest state output */
data fitstats;
  set fitstats;
  label Trees = 'Number of Trees';
  label MiscTrain = 'Training';
  label MiscValid = 'Validation';
run;

/*plot misclassification as function of number of trees */
proc sgplot data=fitstats;
  title "Training vs Validation";
  series x=Trees y=MiscTrain;
  series x=Trees y=MiscValid/lineattrs=(pattern=shortdash thickness=2);
  yaxis label='Misclassification Rate';
run;
title;
```

Output 1.7: Partial Results of Program 1.1G



We also run PROC GRADBOOST to create a gradient boosting model.

Program 1.8: Gradient Boosting Modeling

```
proc gradboost data=mycaslib.donor_raw_partind ntrees=50 intervalbins=20 maxdepth=5;
  input &interval_var. / level = interval;
  input &class_var. / level = nominal;
  target target / level=nominal;
  partition rolevar=_partind_(train='1' validate='0');
  code file="&outdir./gradboost.sas";
run;


/*Score the data using the generated model */

data mycaslib.donor_scored_gradboost;
  set mycaslib.donor_raw_partind;
  %include "&outdir./gradboost1.sas";
run;
```

Notice that in each case, we generate score code that can be used to score future data, which of course is the whole goal of our application.

Model Assessment

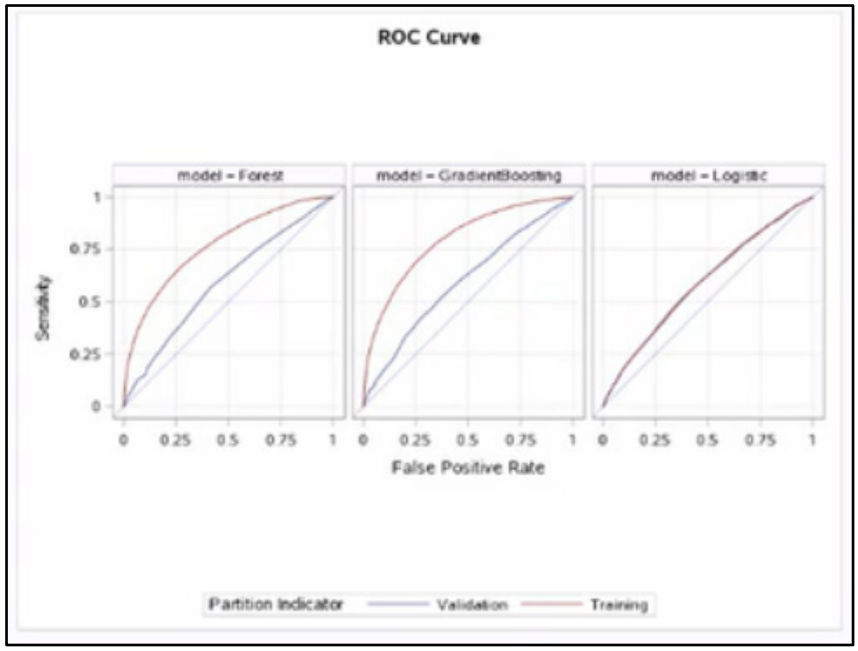
Now that we have trained some models, we can assess them using PROC ASSESS to provide fit statistics, specifying that it should write output to specific data sets that can then be used for graphing.

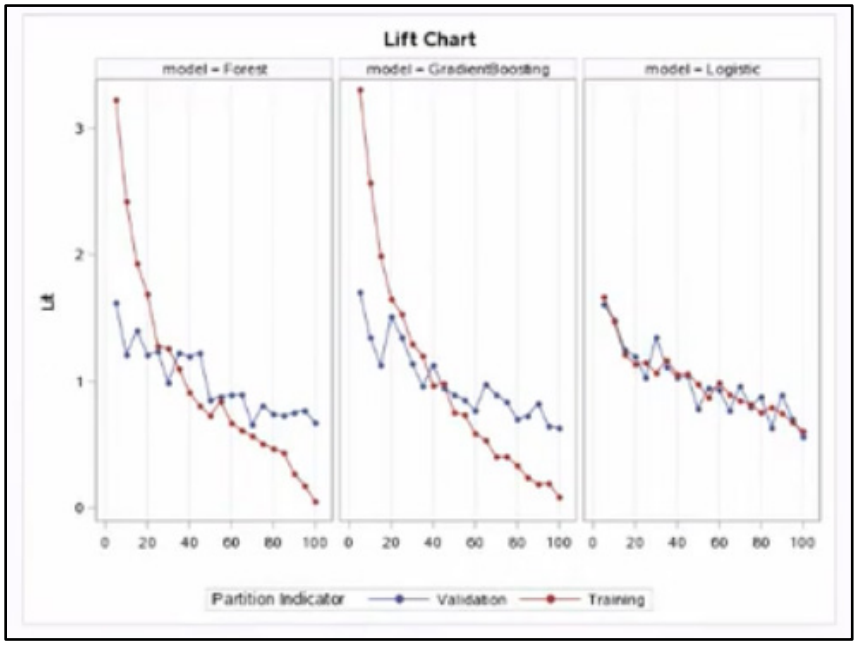
Program 1.9: Model Assessment

```
%macro assess_model(prefix=, var_evt=, var_nevt=);

proc assess data=mycaslib.donor_scored_&prefix.;
  input &var_evt.;
  target target_b / level=nominal event='1';
  fitstat pvar=&var_nevt. / pevent='0';
  by _partind_;

*   ods select fitstat rocinfo liftinfo;
*   ods html exclude fitstat rocinfo liftinfo;

  ods output
    fitstat=mylib.&prefix._fitstat
    rocinfo=mylib.&prefix._rocinfo
    liftinfo=mylib.&prefix._liftinfo;
run;

%mend assess_model;

%assess_model(prefix=logistic, var_evt=p_target_b, var_nevt=p_target_b0);
%assess_model(prefix=forest, var_evt=p_target_b1, var_nevt=p_target_b0);
%assess_model(prefix=gradboost, var_evt=p_target_b1, var_nevt=p_target_b0);

/*****************************/
/* ROC and Lift Charts       */
/*****************************/

ods graphics on;

proc format;
  value partind1b1
    0 = 'Validation'
    1 = 'Training'
  ;
run;

data mylib.all_rocinfo;
  set mylib.logistic_rocinfo (keep=sensitivity fpr _partind_ in=l)
      mylib.forest_rocinfo(keep=sensitivity fpr _partind_ in=f)
      mylib.gradboost_rocinfo(keep=sensitivity fpr _partind_ in=g);

  length model $ 16;
  select;
    when (l) model='Logistic';
    when (f) model='Forest';
    when (g) model='GradientBoosting':
  end;
run;

data mylib.all_liftinfo;
  set mylib.logistic_liftinfo(keep=depth lift cumlift _partind_ in=l)
  set mylib.forest_liftinfo(keep=depth lift cumlift _partind_ in=f)
  set mylib.gradboost_liftinfo(keep=depth lift cumlift _partind_ in=g);
```

```
length model $ 16;
  select;
    when (l) model='Logistic';
    when (f) model='Forest';
    when (g) model='GradientBoosting':
  end;
run;

proc sgpanel data=mylib.all_rocinfo aspect=1;
  panelby model / layout=columnlattice spacing=5;
  title "ROC Curve";
  rowaxis values=(0 to 1 by 0.25) grid offsetmin=.05 offsetmax=.05;
  colaxis values=(0 to 1 by 0.25) grid offsetmin=.05 offsetmax=.05;
  lineparm x=0 y=0 slope=1 / transparency=.7;
  series x=fpr y=sensitivity / group=_partind_;
  format _partind_ partind1b1.;
run;

proc sgpanel data=mylib.all_liftinfo;
  panelby model / layout=columnlattice spacing=5;
  title "Lift Chart";
  colaxis lael= ' ' grid;
  series x=depth y=lift / group=_partind_ markers marerattrs=(symbol=circlefilled);
  format _partind_ partind1b1.;
run;

title;

ods graphics off;
```

Output 1.9: Results of Program 1.9





In Output 1.9, you can see that we have used PROC SGPANEL to plot the ROC curves and lift charts for the models that we have created.

Score New Data

Finally, as previously mentioned, the goal of all this is to deploy the models in such a way that they can be used to score new observations. And of course, this must account for every step in the process, including data preparation, such as the imputation that we performed. The score code generated from each of these steps can be included and executed in a DATA step, as shown in Program 1.10 for the gradient boosting model.

Program 1.10: Score New Data

```
/* On CAS */
data mycaslib.donor_score_data;
  set mylib.donor_score_data;
run;

data mycaslib.donor_scored;
  set mycaslib.donor_score_data;

  %include "outdir./impute1.sas";
  %include "outdir./gradboost1.sas";
run;
```

Programming with the Python Interface

With SAS Viya, you can access SAS analytics through programmatic actions written in SAS or through interfaces with other programming languages such as Python, Java, and Lua. In this section, we will look at some of the capabilities of SAS Visual Data Mining and Machine Learning that can be accessed using Python.

For this example, we will use a data set of anonymized bank data that contains observations from a large Financial Services firm. It is available to download from GitHub from https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/bank. We will use this data to try to classify whether an account holder would make at least one purchase. The observations contain account information for consumers of home equity lines of credit, automobile loans, and other types of short- to medium-term credit instruments. The target variable B_TGT quantifies account responses over the current campaign season. A value of 1 indicates at least one purchase. A value of 0 indicates no purchases.

You can access the same SAS Analytics on SAS Viya using a variety of different Python development environments. In this case, we will build the model and access machine learning actions using a Jupyter notebook. Our program will load and prepare the data. Then we will build a few different machine-learning models and evaluate them.

Import Packages

Visual Data Mining and Machine Learning runs on an in-memory distributed computing infrastructure. The first thing that we need to do is specify the information for connecting to the in-memory server and start a session. This process will create a workspace for our code to run in. To connect to the server, we'll import the SWAT package, as shown in Program 1.11.

Program 1.11: Import Packages

```
from swat import ❶
from matplotlib import pyplot as plt ❷
import pandas as pd ❸
from swat.render import render_html ❹
%matplotlib inline
```

❶ SWAT stands for SAS Scripting Wrapper for Analytics Transfer. It's the general name given to the modules that can be used to access and interact with the SAS Viya platform using Python syntax.

❷ We also import the matplotlib package for plotting purposes.

❸ The pandas package is imported to help explore our data.

❹ The RENDER_HTML function is imported for viewing the output from our models.

Connect to the CAS Server and Start CAS Session

We are ready to connect to the CAS server using the specified server details and our user and password credentials, which are contained within the authinfo file, as shown in Program 1.12. Then we start our CAS session.

Program 1.12: CAS Server Connection Details and Start Session

```
cashost="cloud.example.com"
casport=5570
casauth='-/.authinfo'

sess = CAS(cashost, casport, authinfo="-/.authinfo", caslib="casuser")
```

Import Action Sets

Now that we have connected, we can import the action sets that we need to build our model. In Program 1.13, you can see we are importing a variety of action sets to help us with data pre-processing and for building different types of machine learning models.

Program 1.13: Import Action Sets

```
sess.loadactionset(actionset="dataStep")
sess.loadactionset(actionset="dataPreprocess")
sess.loadactionset(actionset="cardinality")
sess.loadactionset(actionset="sampling")
sess.loadactionset(actionset="decisionTree")
sess.loadactionset(actionset="neuralNet")
sess.loadactionset(actionset="svm")
sess.loadactionset(actionset="astore")
sess.loadactionset(actionset="percentile")
```

Load Data into CAS

We need to load our data in to SAS Cloud Analytics Services (CAS), using the upload_file action in Program 1.14. Notice the note. It indicates that the table BANK_RAW has been added to the caslib.

Program 1.14: Load Data into CAS

```
indata_dir="/opt/sasinside/DemoData"
indata="bank_raw"
if not sess.table.tableExists(table=indata).exists:
    tbl = sess.upload_file(indata_dir+"/"+indata+.".sas7bdat", casout="name":indata))

NOTE: Cloud Analytic Services made the uploaded file available as table BANK_RAW in
      caslib CASUSER(viyauser).
NOTE: The table BANK_RAW has been created in caslib CASUSER(viyauser) from binary
data uploaded to Cloud Analytic Services.
```

We can also check the tables in our library by calling the TABLEINFO function, as shown in Program 1.15.

Program 1.15: TABLEINFO function

```
sess.tableinfo()
```

Output 1.15: TABLEINFO function output

Out[27]: § TableInfo

	Name	Rows	Columns	Encoding	CreateTimeFormatted	ModTimeFormatted	JavaCharSet	CreateTime	ModTime	Global	Re
0	BANK_RAW	1060038	25	utf-8	27Sep2016:13:51:30	27Sep2016:13:51:30	UTF8	1.790603e+09	1.790603e+09	0	0

elapsed 0.00157s · user 0.001s · mem 0.0693MB

We can see in Output 1.15 that the BANK_RAW table was added to our CAS library and that it has 25 columns and approximately 1 million rows.

Explore Data

We can now begin to explore our data. Let's look at a sample of five observations in our data set using the HEAD function in Program 1.16.

Program 1.16: View First Five Observations from Data Set

```
tbl.head()
```

Output 1.16: Output from Program 1.16

Out[51]: Selected Rows from Table BANK_RAW

	b_tgt	cat_input1	cat_input2	cnt_tgt	demog_age	demog_ho	demog_pr	int_tgt	r_demog_homeval	r_demog_inc	...	rfm6	rfm7	rfm8	rf
0	1.0	X	A	NaN	NaN	0.0	24.0	7000.0	57600.0	52106.0	...	22.0	4.0	6.0	5
1	1.0	X	A	2.0	NaN	0.0	24.0	7000.0	57587.0	52106.0	...	22.0	4.0	6.0	5
2	1.0	X	A	2.0	NaN	0.0	0.0	15000.0	44167.0	42422.0	...	16.0	3.0	8.0	1
3	0.0	X	A	0.0	68.0	0.0	32.0	NaN	90587.0	59785.0	...	21.0	2.0	7.0	1
4	0.0	X	A	0.0	NaN	0.0	0.0	NaN	100313.0	NaN	...	38.0	5.0	19.0	2

5 rows × 25 columns

If you want to see more than 5 observations, you can enter a parameter of your choice in the HEAD statement. For example, if you enter 10 for the HEAD function—tbl.head(10)—you will see 10 observations.

We can also check information about the columns in the table using the COLUMNINFO function, see the dimensions of the table using the SHAPE function, and get summary statistics for the table using the DESCRIBE function, as shown in Program 1.17.

Program 1.17: View Table Column Information

```
tbl.columninfo()

tbl.shape

tbl.describe()
```

In the output of the DESCRIBE function, we can see the mean, standard deviation, min, max, and quartile values for the different features.

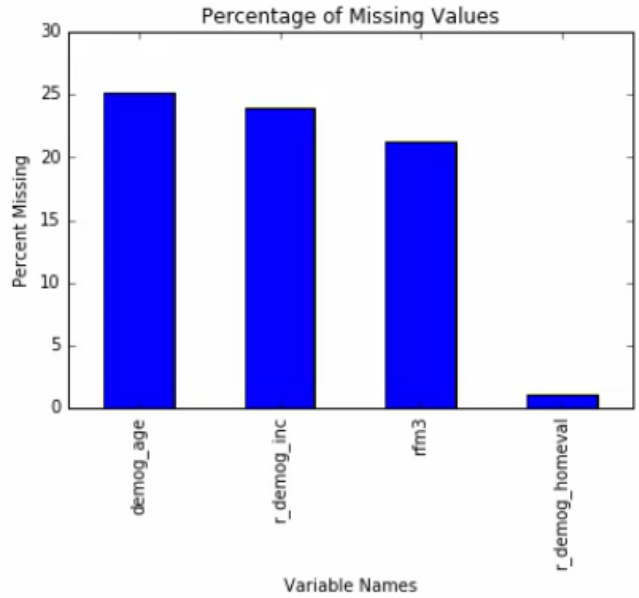
Next, we use the CARDINALITY action set in Program 1.18 to calculate the percentage of missing values by variable and display that information using a bar chart created with matplotlib.

Program 1.18: Explore Data and Plot Missing Values

```
sess.cardinality.summarize(
   table={"name": indata},
   cardinality={"name":"data_card", "replace": True}
)

tbl_data_card=sess.CASTable('data_card')
tbl_data_card.where='_NMISS_>0'

tbl_data_card.vars=['_VARNAME_', '_NMISS_', '_NOBS_']
df_data_card=tbl_data_card.to_frame()
df_data_card['PERCENT_MiSSING']=(df_data_card['_NMISS_'/df_data_card['_NOOBS_'])*100
df_data_card = df_data_card.sort_values(by=['PERCENT_MISSING'], ascending=False)
df_data_card = df_data_card[~df_data_card._VARNAME_.isin(['int_tgt', 'cnt_tgt'])

tbl_forplot=pd.Series(list(df_data_card['PERCENT_MISSING']),
         index=list(df_data_card['_VARNAME_']))
ax=tble_forplot.plot(
    kind='bar',
    title='Percentage of Missing Values'
)
ax.set_ylabel('Percent Missing')
ax.set_xlabel('Variable Names');
```

Output 1.18: Output of Program 1.18

```
NOTE: Writing cardinality.
NOTE: status = 0.
NOTE: The Cloud Analytic Services server processed the request in 0.002409946 seconds.
```



The chart in Output 1.18 shows us that the demog_age, r_demog_inc, and rfm3 features all have between 20% and 25% of their observations missing, and the r_demog_homeval feature has approximately 1% of its values missing.

Impute Missing Values

You can use the DATA PREPROCESS action set to impute the missing values through several methods such as the mean, median, or mode. In this case, we will impute the demog_age variable using the mean and the remaining variables using the median. Note that we output the new table with the imputed variables as bank_prepped_pr, as shown in Program 1.19.

Program 1.19: Impute Missing Values

```
# Pipelined imputation using transform action
sess.dataPreprocess.transform(
   table=("name":indata),
   casOut={"name":"bank_prepped_pr", "replace":True},
   copyAllVars=True
   outVarsNameGlobalPrefix="IM",
   requestPackages=[
      {"impute":{"method":"MEAN"}, "inputs":{"demog_age"}},
      {"impute":{"method":"MEDIAN}, "inputs":{"r_demog_inc", "r_demog_homeval",
       "rfm3"}}
    ]
)
```

Output 1.19: Output from Program 1.19

Out[33]:

§ TransInfo

Transformation Requests for BANK_RAW

	ActualName	NTransVars	ImputeMethod
0	_TR1	1	Mean
1	_TR2	3	Median

§ VarTransInfo

Variable Transformation Information for BANK_RAW

	Variable	Transformation	ResultVar	N	NMiss	ImputedValueContinuous
0	demog_age	IM	IM_demog_age	793177	266861	58.716371
1	r_demog_homeval	IM	IM_r_demog_homeval	1047905	12133	74473.000000
2	r_demog_inc	IM	IM_r_demog_inc	806785	253253	48779.000000
3	rfm3	IM	IM_rfm3	834252	225786	14.000000

§ OutputCasTables

	casLib	Name	Rows	Columns	casTable
0	CASUSER(viyauser)	bank_prepped_pr	1060038	29	CASTable('bank_prepped_pr', caslib='CASUSER(vi...

elapsed 1.06s · user 2.92s · sys 0.64s · mem 22.6MB

You can also add new features to a data set. In Program 1.20, we use a dataStep action to execute SAS code that adds an indicator variable to our data set based on whether the RFM3 observation is missing.

Program 1.20: Create New Indicator Variable for Missing RFM3 Values

```
sess.dataStep.runcode(code ="""
                data bank_prepped;
                    set bank_prepped_pr;
                    if missing(rfm3) then RFM3_IND = 1;
                    else RFM3_IND = 0;
                run;
                 """)
```

Now that we have created some imputed variables and added a new feature to our data set, let's take a quick look at the table using a HEAD statement again in Program 1.21. We can see that our variables have been added.

Program 1.21: View Table and Columns

```
tbl_tmp = sess.CASTable("bank_prepped")
tbl_tmp.head()

tbl_tmp.columns
```

Output 1.21: Output from Program 1.21

Out[35]:

og_inc	...	rfm11	rfm12	demog_genf	demog_genm	account	IM_demog_age	IM_r_demog_homeval	IM_r_demog_inc	IM_rfm3	RFM3_IND
)	...	2.0	28.0	1.0	0.0	100512177	49.000000	80904.0	40611.0	14.00	1.0
)	...	7.0	54.0	1.0	0.0	100512178	58.716371	86993.0	40608.0	7.00	0.0
)	...	6.0	64.0	1.0	0.0	100512179	58.716371	82948.0	40610.0	39.00	0.0
)	...	4.0	69.0	0.0	1.0	100512180	44.000000	84592.0	40607.0	15.00	0.0
)	...	6.0	69.0	0.0	1.0	100512181	43.000000	87768.0	40608.0	25.33	0.0

```
Out[36]: Index(['b_tgt', 'cat_input1', 'cat_input2', 'cnt_tgt', 'demog_age', 'demog_ho',
                'demog_pr', 'int_tgt', 'r_demog_homeval', 'r_demog_inc', 'rfm1', 'rfm2',
                'rfm3', 'rfm4', 'rfm5', 'rfm6', 'rfm7', 'rfm8', 'rfm9', 'rfm10',
                'rfm11', 'rfm12', 'demog_genf', 'demog_genm', 'account', 'IM_demog_age',
                'IM_r_demog_homeval', 'IM_r_demog_inc', 'IM_rfm3', 'RFM3_IND'],
               dtype='object')
```

Partition Data

Now that we have explored our data, we can set up some variables for the features in the data set to reference while building our models. We can also partition the data into training and validation subsets using the sampling action set as shown in Program 1.22.

Program 1.22: Partition Data

```
target            = "b_tgt"
class_inputs      = ["cat_input1", "cat_input2", "rfm3_ind", "dmog_ho"]
class_vars        = [target] + class_inputs
interval_inputs   = ["dmog_age", "rfm1", "rfm2", "rfm4", "rfm5", "rfm6", "rfm7",
                    "rfm8", "rfm9", "rfm10", "rfm11", "rfm12", "IM_demog_age",
                    "IM_r_demog_homeval", "IM_r_demog_inc", "IM_rfm3"]
all_inputs        = interval_inputs + class_inputs

sess.sampling.stratified(
   table={"name":"bank_prepped", "groupBy":"b_tgt"❶},
   output={"casOut": {"name":"bank_part", "replace":True}, "copyVars":"ALL"},❷
   samppct=70,❶
   partind=True
)
```

❶ Let's specify the target of interest in our modeling problem to ensure appropriate stratification. In this example, we are using stratified sampling based on the target variable b_tgt and setting the sampling percentage to 70%.

❷ This creates a new data set named bank_part with a partition index that we will use for building and evaluating our models.

Model Building

We are ready to begin building our models. We'll start with a random forest model, then show the code to build a gradient boosting model, neural net model, and support vector machine model.

Random Forest

The random forest model is in the decisionTree action set. To see what action we need to use, we can use the HELP action and specify the decisionTree action set, as shown in Program 1.23. This gives us a list and description of the actions within that action set, as shown in Output 1.23.

Program 1.23: HELP Action

```
render_html(sess.help(actionset="decisionTree"))
```

Output 1.23: Results of Program 1.23

Name	Description
dtreeTrain	Train a decision tree
dtreeScore	Score a table using a decision tree model
dtreeSplit	Split decision tree nodes
dtreePrune	Prune a decision tree
dtreeMerge	Merge decision tree nodes
dtreeCode	Generate DATA step scoring code from a decision tree model
forestTrain	Train a forest
forestScore	Score a table using a forest model
forestCode	Generate DATA step scoring code from a forest model
gbtreeTrain	Train a gradient boosting tree
gbtreeScore	Score a table using a gradient boosting tree model
gbtreecode	Generate DATA step scoring code from a gradient boosting tree model

In Output 1.23 we can see that forestTrain is used to build a FOREST model. We will use this action to build a random forest model as shown in Program 1.24.

Program 1.24: Random Forest Model

```
rf=sess.decisionTree.forestTrain(
    table={
      "name":"bank_part", ❶
      "where":"strip(put(_partind_, best.))='1'"
},
inputs=all_inputs, ❷
nominals=class_vars, ❸
target="b_tgt", ❸
nTree=30,
nBins=20,
leafSize=5,
maxLevel=21, ❹
crit="GAINRATIO",
varImp=True,
seed=100,
OOB=True,
Vote="PROB",
casOut={"name":"forest_model", "replace":True}
)

# Output model statistics
render_html(rf)

# Score
sess.decisionTree.forestScore( ❺
    table={"name":"bank_part"},
    modelTable={"name":"forest_model"},
    casOut={"name":"_scored_rf", "replace":True},
```

```
    copyVars={"b_tgt", "_partind_"},
    vote="PROB"
)

# Create p_b_tgt0 and p_b_tgt1 as _rf_predp_ is the probability of event in
_rf_predname_
sess.dataStep.runCode(
    code="""data _scored_rf; set _scored_rf; if _rf_predname_=1 then do;
    p_b_tgt1=_rf_predp_; p_b_tgt0=1-p_b_tgt1; end; if _rf_predname_=0 then do;
    p_b_tgt0=_rf_predp_; p_b_tgt1=1-p_b_tgt0; end; run;"""
)
```

❶ For the input table, we will use the bank_part table and specify only those observations within our partitioned data set that are selected for training.

❷ Now we'll choose the all_inputs variable that we created earlier as the input for our model.

❸ Select the class variables for the nominal parameter and set the target variable as b_tgt.

❹ We can set additional parameters for our model such as the number of trees, the number of possible bins for splitting features, the minimum number of observations for relief, and the maximum levels for the tree. We can easily change these parameters and rebuild the model if we choose. For example, you could change the number of trees to 50.

❺ Let's also score the partition's data using the forestScore action to create the _scored_rf table.

In the output of this program, we can see some fit statistics for the model as well as a table showing the variable importance. If we would like to reference any of those tables, we can see the table names by looking at the keys of the model, as shown in Program 1.25.

Program 1.25: Model Keys

```
rf.keys()
```

Output 1.25: Results of Program 1.25

```
Out[41]: odict_keys(['ModelInfo', 'ErrorMetricInfo', 'DTreeVarImpInfo', 'OutputCasTables'])
```

In this case, we want to look at the variable importance table, so we will access it using the DTreeVarIMpInfo key as shown in Program 1.26 and Output 1.26.

Program 1.26: View Table

```
rf['DTreeVarImpInfo']
```

Output 1.26: Results of Program 1.26

	Variable	Importance	Std
0	rfm5	17348.415363	2506.502293
1	rfm9	5736.186864	376.234291
2	IM_r_demog_homeval	4129.797530	206.349047
3	rfm7	3901.969888	981.340609
4	cat_input2	640.518017	89.159134
5	rfm6	515.926860	181.698745
6	rfm8	509.001955	143.520447
7	rfm11	407.434129	39.625314
8	IM_r_demog_inc	392.690911	16.223021
9	RFM3_IND	279.737197	235.296548
10	rfm10	195.989876	5.912201
11	cat_input1	192.417600	40.041340
12	IM_demog_age	179.368573	3.502274
13	rfm12	148.223325	10.702744
14	demog_age	118.171470	3.735738
15	rfm2	96.736941	6.871811
16	demog_ho	37.275554	3.343904
17	rfm1	5.607402	2.163071
18	IM_rfm3	4.862302	2.535385

Now let's run through similar exercises to create a gradient boosting model, a neural net model, and a support vector machine model.

Gradient Boosting

Program 1.27: Gradient Boosting Model

```
gb=sess.decisionTree.gbtreeTrain(
    table={
      "name":"bank_part",
      "where":"strip(put(_partind_, best.))='1'"
},
inputs=all_inputs,
nominals=class_vars,
target="b_tgt",
nTree=10,
nBins=20,
maxLevel=6,
varImp=True,
casOut={"name":"gb_model", "replace":True}
)

# Output model statistics
render_html(gb)

# Score
sess.decisionTree.gbtreeScore(
    table={"name":"bank_part"},
    modelTable={"name":"gb_model"},
    casOut={"name":"_scored_gb", "replace":True},
    copyVars={"b_tgt", "_partind_"},
)

# Create p_b_tgt0 and p_b_tgt1 as _gbt_predp_ is the probability of event in
_gbt_predname_
sess.dataStep.runCode(
    code="""data _scored_gb; set _scored_gb; if _gbt_predname_=1 then do;
    p_b_tgt1=_gbt_predp_; p_b_tgt0=1-p_b_tgt1; end; if _gbt_predname_=0 then do;
    p_b_tgt0=_gbt_predp_; p_b_tgt1=1-p_b_tgt0; end; run;"""
```

Neural Network

Program 1.28: Neural Network Model

```
nn=sess.neuralNet.annTrain(
    table={
      "name":"bank_part",
      "where":"strip(put(_partind_, best.))='1'"
},
inputs=all_inputs,
nominals=class_vars,
target="b_tgt",
hiddens={2},
acts={"TANH"},
combs={"LINEAR"},
targetAct="SOFTMAX",
errorFunc="ENTROPY",
std="MIDRANGE",
randDist="UNIFORM",
scaleInit=1,
nloOpts={
    "optmlOpt":{"maxIters:250, "fConv":1e-10},
    "lbfgsOpt":{"numCorrections":6},
    "printOpt":{"printLevel":"printDetail"},
    "validate":{"frequency":1}
},
casOut={"name":"nnet_model", "replace":True}
)

# Output model statistics
render_html(nn)

# Score
sess.neuralNet.annScore(
    table={"name":"bank_part"},
    modelTable={"name":"nnet_model"},
    casOut={"name":"_scored_nn", "replace":True},
    copyVars={"b_tgt", "_partind_"},
)

# Create p_b_tgt0 and p_b_tgt1 as _nn_predp_ is the probability of event in
_nn_predname_
sess.dataStep.runCode(
    code="""data _scored_nn; set _scored_nn; if _nn_predname_=1 then do;
    p_b_tgt1=_nn_predp_; p_b_tgt0=1-p_b_tgt1; end; if _nn_predname_=0 then do;
    p_b_tgt0=_nn_predp_; p_b_tgt1=1-p_b_tgt0; end; run;"""
```

Support Vector Machine

Program 1.29: Support Vector Machine Model

```
sv=sess.svm.svmTrain(
    table={
      "name":"bank_part",
      "where":"strip(put(_partind_, best.))='1'"
},
inputs=all_inputs,
nominals=class_vars,
target="b_tgt",
kernel="POLYNOMIAL",
degree=2,
id={"b_tgt", "_partind_"},
savestate={"name":"svm_astore_model", "replace":True}
)

# Output model statistics
render_html(sv)

# Score using ASTORE
sess.astore.score(
    table={"name":"bank_part"},
```

```
    rstore={"name":"svm_astore_model"},
    out={"name":"_scored_svm", "replace":True}
)
```

As you can see, we can access all the latest machine learning algorithms in SAS Viya using Python. Notice that in the support vector machine model, we are scoring the data using the ASTORE action set. This scoring action helps to efficiently score models that can contain complex scoring code, such as support vector machines, random forest, and factorization machines.

Model Assessment

Now that we have trained our models, we can assess them using the PERCENTILE action set. In Program 1.30, we create a function named ASSESS MODEL so that we can apply the same action to the different models in order to evaluate and compare them. We will take the subset of our data that is assigned to our validation set and then retrieve the predicted probabilities of an event occurring for each of the four models that we've built.

Program 1.30: Assess Models

```
def assess_model(prefix):
    return sess.percentile.assess(
        table={
          "name":"_scored_" + prefix,
          "where": "strip(put(_partind_, best.))='0'"
     },
     inputs=[{"name":"p_b_tgt1"}],
     response="b_tgt",
     event="1",
     pVar={"p_b_tgt0"),
     pEvent={"0"}
     )

rfAssess=assess_model(prefix="rf")
rf_fitstat =rfAssess.FitStat
rf_rocinfo =rfAssess.ROCInfo
rf_liftinfo=rfAssess.LIFTInfo

gbAssess=assess_model(prefix="gb")
gb_fitstat =gbAssess.FitStat
gb_rocinfo =gbAssess.ROCInfo
gb_liftinfo=gbAssess.LIFTInfo

nnAssess=assess_model(prefix="nn")
nn_fitstat =nnAssess.FitStat
nn_rocinfo =nnAssess.ROCInfo
nn_liftinfo=nnAssess.LIFTInfo

svmAssess=assess_model(prefix="svm")
svm_fitstat =svmAssess.FitStat
svm_rocinfo =svmAssess.ROCInfo
svm_liftinfo=svmAssess.LIFTInfo
```

Create ROC and Lift Plots Using Validation Data

Let's compare the models using their Area Under the ROC Curve (AUC) metric, ROC curves, and LIFT charts.

Program 1.31: Prepare Assessment Results for Plotting

```
# Add new variable to indicate type of model
rf_liftinfo["model"]="Forest"
rf_rocinfo["model"]="Forest"
gb_liftinfo["model"]="GradientBoosting"
gb_rocinfo["model"]="GradientBoosting"
nn_liftinfo["model"]="NeuralNetwork"
nn_rocinfo["model"]="NeuralNetwork"
svm_liftinfo["model"]="SVM"
svm_rocinfo["model"]="SVM"
```

```
# Append data
all_liftinfo=rf_liftinfo.append(gb_liftinfo, ignore_index=True) \
    .append(nn_liftinfo, ignore_index=True) \
    .append(svm_liftinfo, ignore_index=True)
all_rocinfo=rf_rocinfo.append(gb_rocinfo, ignore_index=True) \
    .append(nn_rocinfo, ignore_index=True) \
    .append(svm_rocinfo, ignore_index=True)
```

We will print the AUC scores as shown in Program 1.32 and Output 1.32 and then plot the ROC curves and LIFT charts for each of the models.

Program 1.32: Print AUC

```
print("AUC (using validation data)".center(80, '-'))
all_rocinfo[["model", "C"]].drop_duplicates(keep="first").sort_values(by="C",
    ascending=False)
```

Output 1.32: AUC

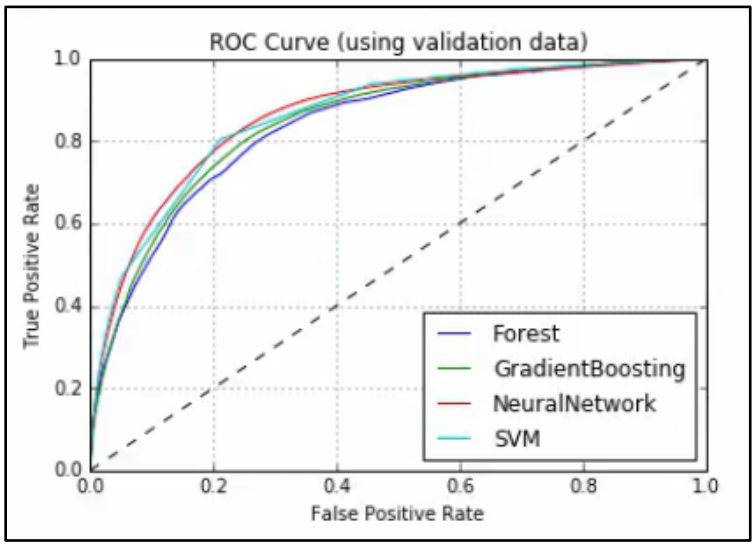
Out[48]:

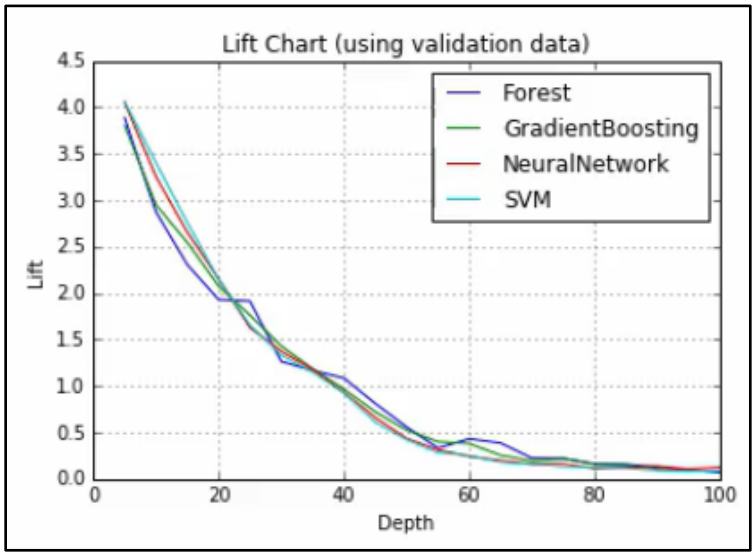
	model	C
200	NeuralNetwork	0.863268
300	SVM	0.860612
100	GradientBoosting	0.845840
0	Forest	0.837054

Program 1.33: Draw ROC and Lift Plots

```
#/* Draw ROC charts */
plt.figure()
for key, grp in all_rocinfo.groupby(["model"]):
     plt.plot(grp["FPR"], grp["Sensitivity"], label=key)
plt.plot([0,1], [0,1], "k—")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.grid(True)
plt.legend(loc="best")
plt.title("ROC Curve (using validation data)")
plt.show()

#/* Draw lift charts */
plt.figure()
for key, grp in all_liftinfo.groupby(["model"]):
     plt.plot(grp["Depth"], grp["Lift"], label=key)
plt.xlabel("Depth")
plt.ylabel("Lift")
plt.grid(True)
plt.legend(loc="best")
plt.title("Lift Curve (using validation data)")
plt.show()

# End CAS session
sess.close()
```

Output 1.33: ROC and Lift Plots





In this case, we can see that the NeuralNetwork model seems to perform slightly better on the validation data for the models that we have constructed.

Conclusion

This chapter presented two simple, start-to-finish machine learning solutions. One was programmed in SAS Viya through the SAS Studio interface and the other accessed the analytics within SAS Viya through the Python interface, using the Jupyter notebook.

Although this can all be written completely manually, SAS Studio provides tasks and code snippets that generate much of this code for you so that you don't have to worry about the appropriate syntax or structure and all of the options are clearly presented to you. The next chapter will explore some of these Data Mining and Machine Learning tasks.

Resources

This chapter is based on the "Data Mining and Machine Learning on SAS Viya" videos in SAS® Viya® Enablement, a free course available from SAS Education.

You might find the following documentation and resources helpful as you learn more about programming in SAS Visual Data Mining and Machine Learning:

- SAS Visual Data Mining and Machine Learning 8.4 in SAS Viya 3.4
- SAS® Viya® 3.2 Programming: Getting Started with SAS Viya for Python
- Link to donor data set available to download from SAS Visual Data Mining and Machine Learning documentation: https://support.sas.com/documentation/prod-p/vdmml/zip/567885_donor_data.zip
- Link to download Bank data set link: https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/bank

Chapter 2: Data Mining and Machine Learning Tasks in SAS Studio

Introduction

As you have seen in the previous chapter, you can use your existing SAS programming knowledge in SAS Viya to efficiently and effectively analyze your data. SAS Viya also offers several new data mining and machine learning procedures that are available for sites with SAS Viya installed. You can see a complete list of the procedures with explanations in the documentation.

In this chapter, we will look at several data mining and machine learning tasks in SAS Studio that use these procedures. A predefined task in SAS Studio is an XML and Apache Velocity code file that generates SAS code and formats the results for you. Tasks include SAS procedures from simple data listings to complex analytical procedures. You might already be familiar with tasks in SAS Studio that use SAS 9. If you license SAS Visual Analytics, SAS Visual Data Mining and Machine Learning, or any other SAS Viya products, you will have access to certain predefined tasks in SAS Studio.

This chapter is by no means a comprehensive guide to all of the data mining and machine learning procedures, tasks, or features in SAS Viya. It is an introduction to using SAS Viya for your statistical programming so that you can begin to take advantage of the scalability, elasticity, and flexibility of this modern analytics platform.

The data source used in the examples contains a list of donors to an organization. This data source has a target variable, TARGET_B, which is a binary variable that has the value 1 for a person who has donated during a mailing. The example data is available to download from the SAS® Visual Data Mining and Machine Learning documentation. See the Resources section for the link.

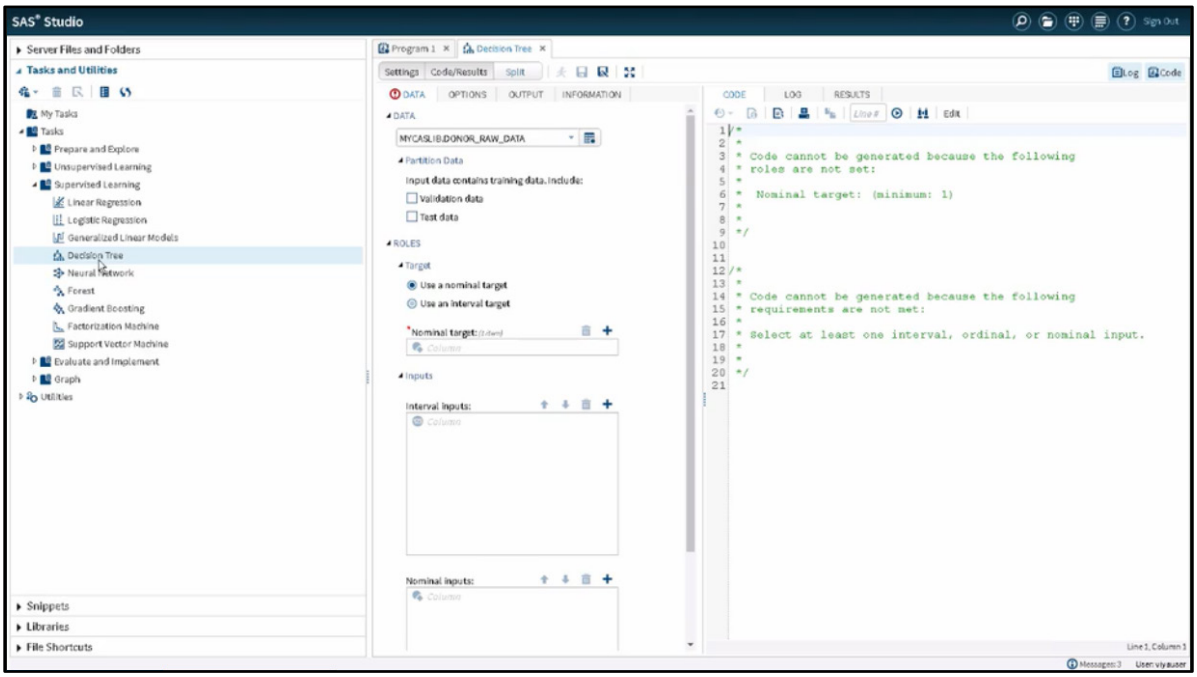
Decision Tree

In this section, you will learn how to use the Auto-tune option when developing a supervised learning model in SAS Viya. This particular example is focused on the decision tree.

The data source for this example contains a list of donors and non-donors to an organization and has a binary target variable named TARGET_B, which has a value “1” for those who donated during a mailing, and a value “0” for those who did not.

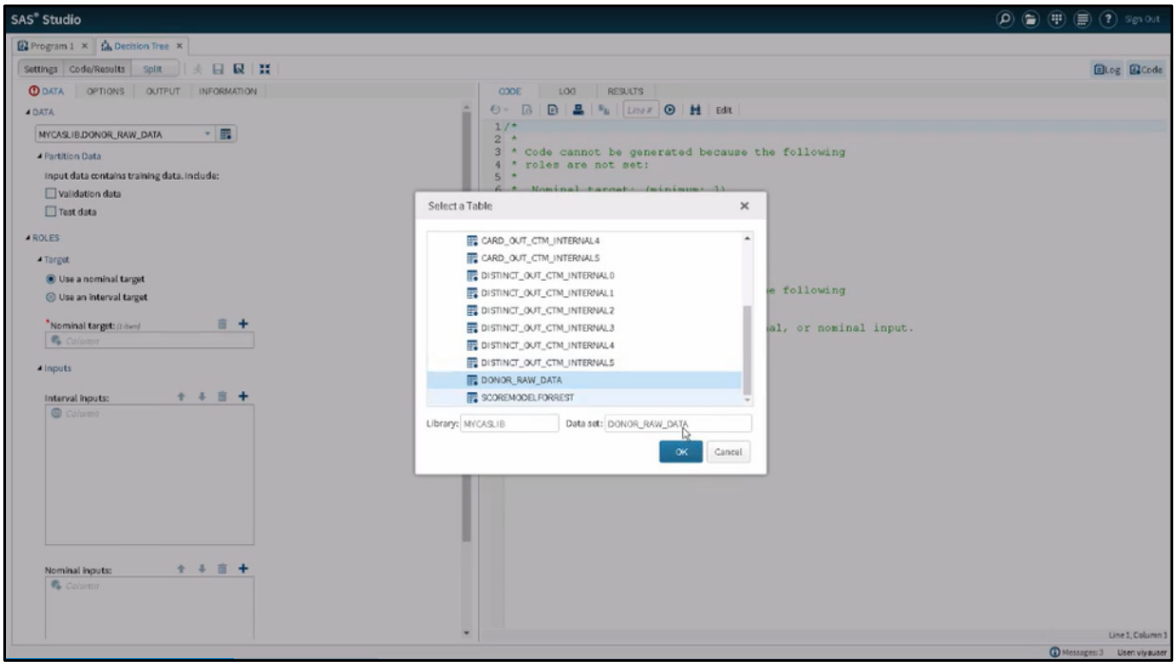
To begin, in SAS Studio click the **Tasks and Utilities** option on the left panel. Then, expand the **Tasks** option and expand the **Supervised Learning** option. From that folder, double-click the **Decision Tree** option as shown in Figure 2.1. This opens the Decision Tree panel at the center of your screen. You can see four tabs in the center panel: DATA, OPTIONS, OUTPUT, and INFORMATION.

Figure 2.1: Decision Tree Task



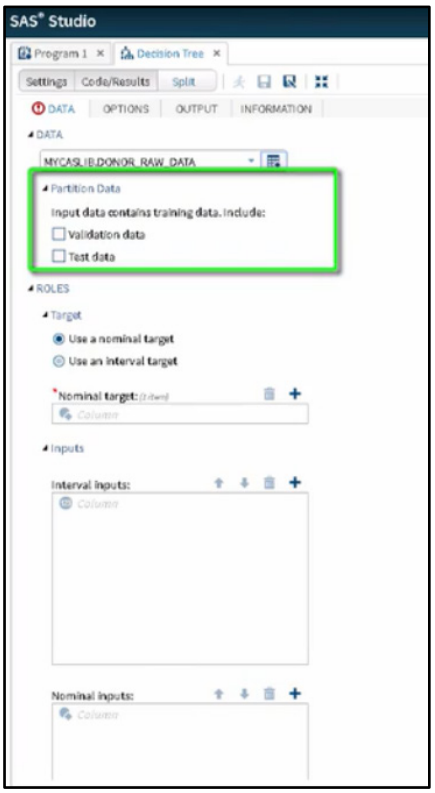
Now let's learn how to configure the Decision Tree by using the Auto-tune option. Click the **DATA tab** in the center panel. In the DATA property, click the **Select a table** button to select the table that we are going to use for this example. Your table should be available in Libraries ▶ My Libraries ▶ MYCASLIB. Select the table named DONOR_RAW_DATA, as shown in Figure 2.2. Click OK.

Figure 2.2: Select a Table



Keep Validation and Test data deselected in the **Partition Data** option, as shown in Figure 2.3 outlined in a green box.

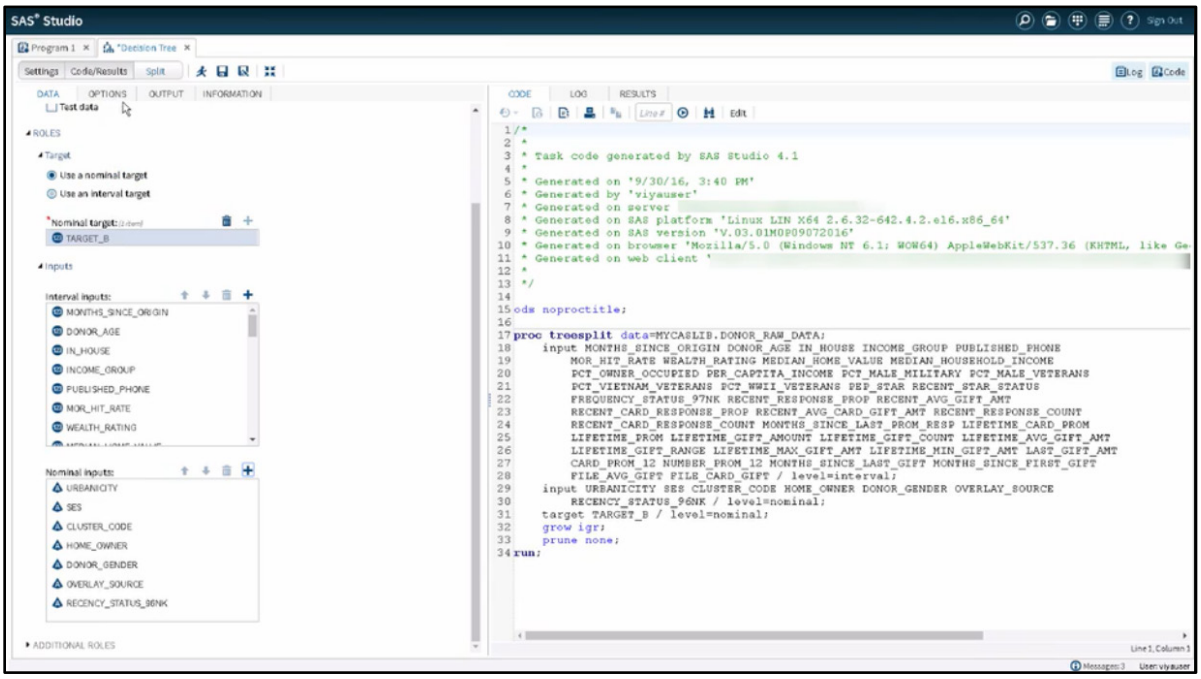
Figure 2.3: Partition Data Option



Let's look at the Roles property next. On the TARGET option, select **Use a nominal target**. On Nominal target, click the **Add a column** button that appears as a + sign in the upper right. In the Columns window that opens, select the variable TARGET_B. Click OK in the Columns window. Let's go now to the Input options. On Interval inputs, click the **Add columns** button (+). A pop-up menu will show all the interval variables. Click MONTHS_SINCE_ORIGIN. Scroll down to the end of the variable window selection. Press the shift key and click the variable FILE_CARD_GIFT. This will select all interval variables except TARGET_D. Click OK in the Columns window.

Notice that the CODE panel on the right begins to populate the TREESPLIT procedure. On Nominal Inputs, click the **Add columns** button (+). A pop-up menu will show all the nominal variables. Click URBANICITY. Scroll down to the end of the variable selection window, press shift, and click the variable RECENCY_STATUS_96NK. This will select all nominal variables except TARGET_D and CONTROL_NUMBER. Click OK in the Columns window. Your screen should now look like Figure 2.4.

Figure 2.4: DATA Tab Options



Click the **OPTIONS tab** in the left panel. Under the METHODS property, select the option **Automatically tune selected options**. The auto-tune procedure initially sets the depth of the tree as 6 with a lower bound at 2 and an upper bound at 20. We will leave these settings as they are. It means that the Auto-tune method will try different depths in order to achieve the best performance according to the target variable.

Also, the Auto-tune procedure sets the number of bins used for interval variables to 20 with a lower bound at 20 and an upper bound at 200. It means that the Auto-tune method will try different bins along the tuning, always aiming for the best performance in relation to the particular target variable selected.

The Auto-tune procedure will use different splitting criteria methods. Information gain ratio is the default method, but CHAID, Chi-square, Entropy, and Gini index also will be tried. You can change this setting by clicking the **Edit** button and change both the order and the method itself that will be used. For now, we will leave the default selected.

At the initial settings, the Auto-tune process does not prune the tree. You might want to change this and use the C4.5 pruning method. However, that option can increase the running time. In this example, we will leave it as-is.

As splitting criteria, the Auto-tune procedure sets the minimum number of observations for a leaf to 5 and the maximum branches of a node to 2. In other words, there will be no leaves with fewer than five observations, and the splitting process will be binary, that is, a two-way split.

The METHODS property section in the left tab should now look like Figure 2.5.

Figure 2.5: OPTIONS METHODS Properties

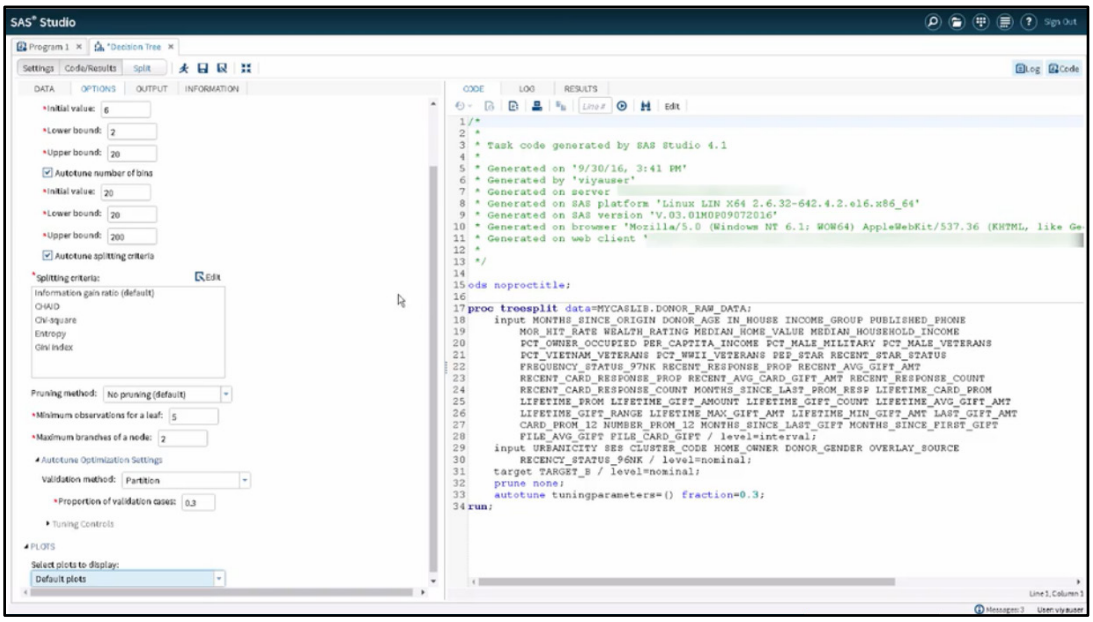
Under the Autotune Optimization Settings section, the Auto-tune procedure sets the validation method as Partition and the proportion of validation cases as 0.3 or 30%. You also have the option to use the K-fold cross validation as a validation method. Do not select K-fold cross validation.

Under Tuning Controls, the Auto-tune procedure sets the maximum time of 60 minutes for the optimization process based on five iterations evaluating 50 functions. The number of evaluations per iteration will be 10 according to these defaults.

Finally, on the PLOTS option, you can select to output particular plots. The default is the whole tree and the zoom tree. Or you can select the ones that you like in addition to the whole tree and the zoom tree. For example, the depth of the sub-tree and the variable importance chart. Or you can even suppress all the plots.

After selecting these options in the OPTIONS tab, your screen should look like Figure 2.6.

Figure 2.6: OPTIONS METHOD Properties, continued

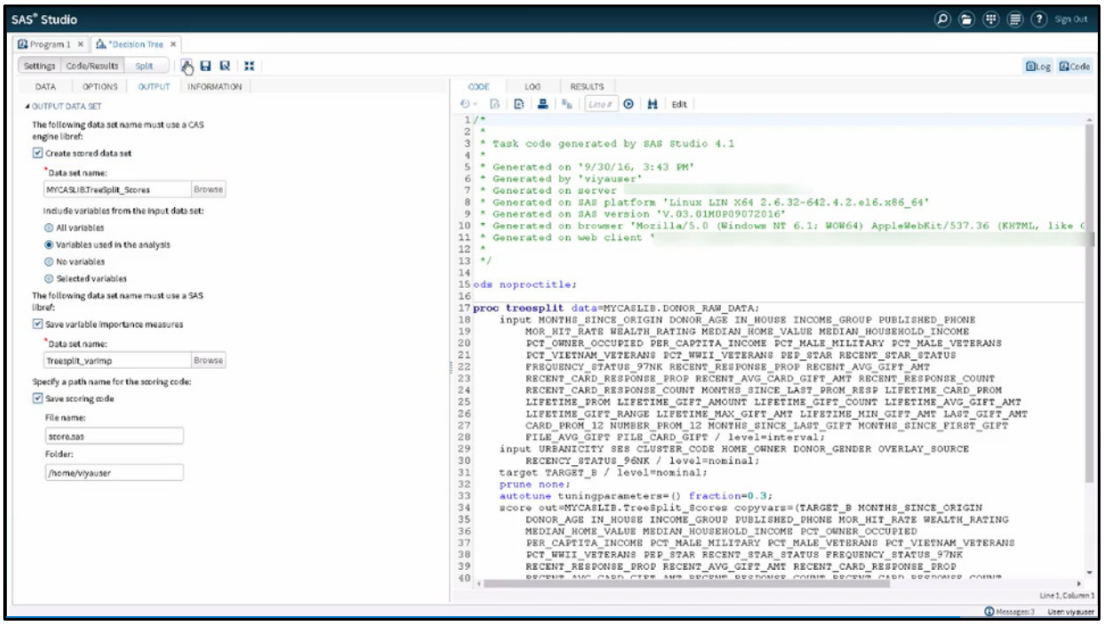


Click the **OUTPUT** tab. On the **OUTPUT DATA SET** property, check the option **Create score data set**. For the **Data set name** option, set your library and data set names. For example, enter MYCASLIB.Treesplit_scores. Notice how the CODE panel on the right populates the TREESPLIT procedure with the SCORE OUT= option.

For **Include variables from the input data set**, select **Variables used in the analysis**. Check the option **Save variable importance measures**. Enter the data set name for your variable importance table. If you don't enter any library name, SAS Viya will set it as the Work library. Let's enter just the data set name as Treesplit_varimp. If the file already exists in the Work library, you are going to be asked to overwrite it.

Check the option **Save scoring code**. Leave the File name and the Folders as they are. Your screen should now look like Figure 2.7.

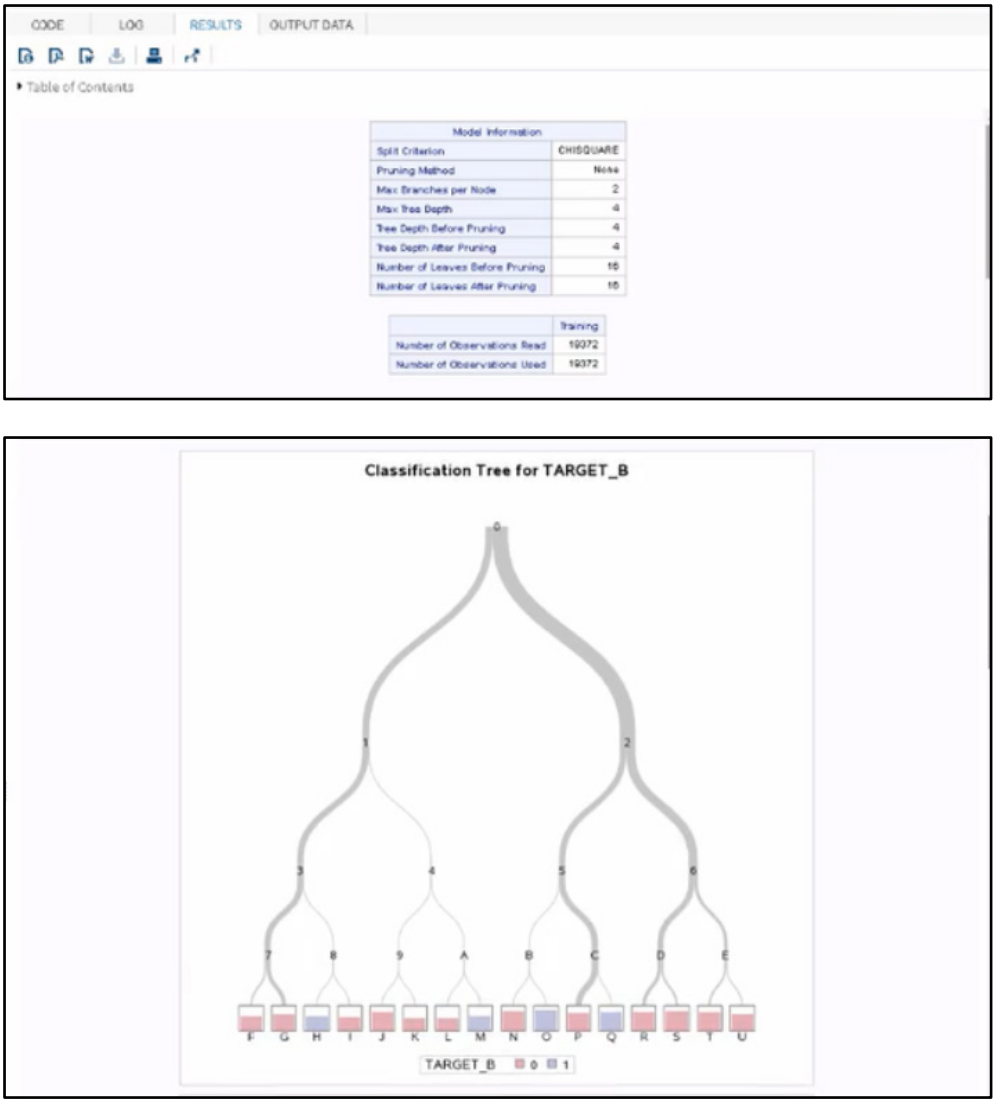
Figure 2.7: OUTPUT Properties



Click the **Run** shortcut button to run the task. While the code is running, notice the CODE tab changes to the LOG tab, meaning that the running process is being populated, including errors, warnings, and notes.

When the run is over, the RESULTS tab shows up with the model information where it can be seen that chi-square was selected by Auto-tuning as the Split Criterion, as shown in Figure 2.8. It also shows the classification tree for the target, the sub-tree, the fit statistics for the selected tree, the variable importance table, the output table containing the scores, and a series of tables summarizing the results of auto-tuning.

Figure 2.8: Partial Results of TREESPLIT Procedure



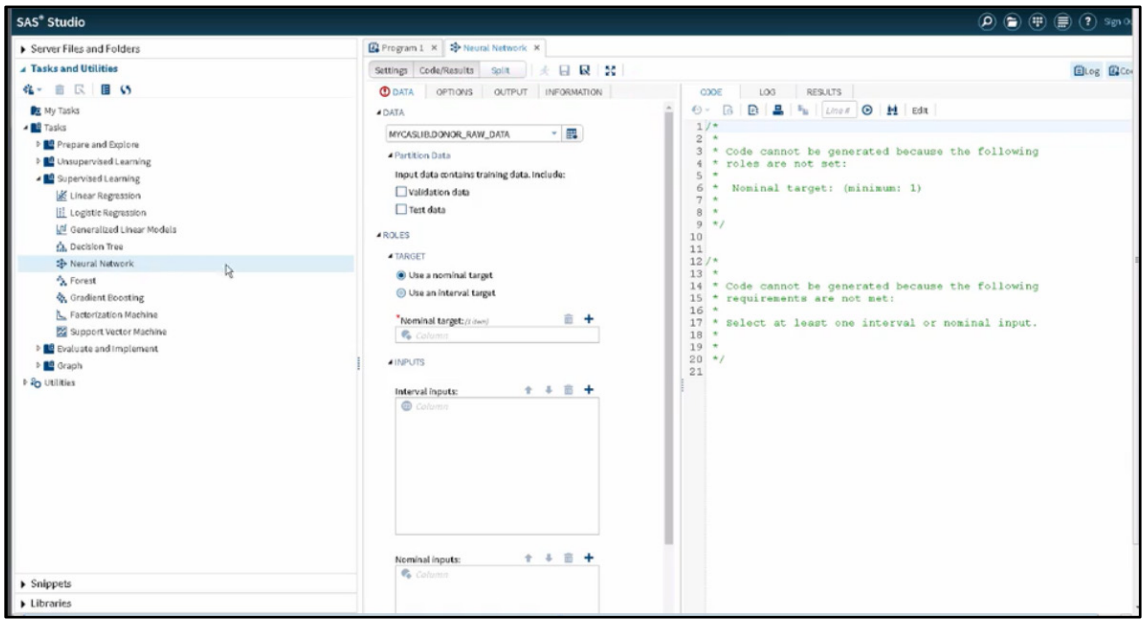
Neural Network

In this section, you will learn how to use the Auto-tune option when developing a supervised learning model in SAS Viya. This particular example is focused on the neural network.

The data source for this example contains a list of donors and non-donors to an organization and has a binary target variable named TARGET_B, which has a value "1" for those who donated during a mailing, and a value "0" for those who did not.

To begin, in SAS Studio, click the **Tasks and Utilities** option on the left panel. Then, expand the **Tasks** option and expand the **Supervised Learning** option. From that folder, double-click the **Neural Network** option as shown in Figure 2.9. This opens the Neural Network panel at the center of your screen. You can see four tabs in the center panel: DATA, OPTIONS, OUTPUT, and INFORMATION.

Figure 2.9: Neural Network Task

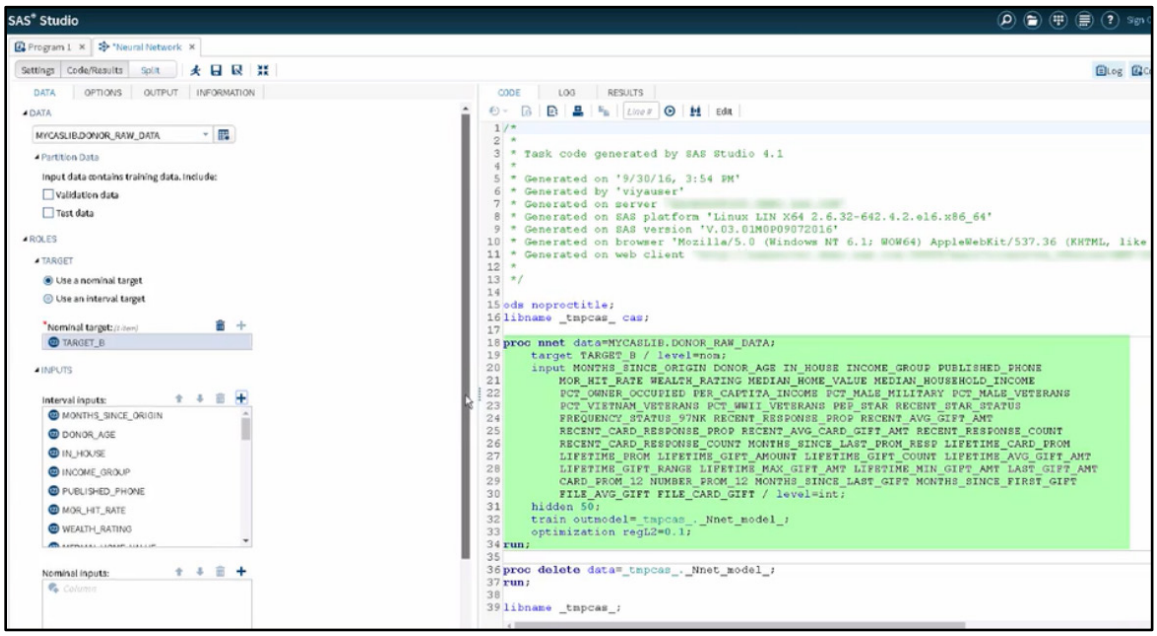


We will learn how to configure the Neural Network by using the Auto-tune option.

On the DATA tab in the DATA property, click the **Select a table** button to select the table that will be used in this example. The table should be available in Libraries ▶ My Libraries ▶ MYCASLIB. Select the table named DONOR_RAW_DATA. Click OK. On the **Partition Data** option, do not select Validation data or Test data.

Let's go to the ROLES property. On the TARGET option, select **Use a nominal target**. On Nominal target, click the **Add a column** button (+) and select the variable TARGET_B in the Column window. Click OK. On Interval inputs, click the **Add columns** button (+). A pop-up menu shows all the interval variables. Click MONTHS_SINCE_ORIGIN, scroll down to the end of the variable window selection, hold down the Shift key, and click the variable FILE_CARD_GIFT. This selects all the interval variables except TARGET_D. Click OK in the Columns window. Notice that the CODE panel to the right begins to populate the NNET procedure, as shown in Figure 2.10.

Figure 2.10: DATA Tab Options



On Nominal inputs, click the **Add columns** button. A pop-up menu shows all the nominal variables. Click URBANICITY. Scroll down to the end of the variable window selection, hold down the Shift key, and click the variable RECENCY_STATUS_96NK. Click OK in the Columns window.

Click the **OPTIONS tab**. Under the METHODS property, select the option Automatically tune selected options. As a default, under the **Data Preparation** option, the Auto-tune procedure does not include the missing values, as they will be discarded during the training process. If you select **the Include missing values** option, be aware that for nominal variables, the missing values are treated as only another level of the input. For interval variables, missing values are treated as the mean of all observations in the data set.

The initial settings for the Auto-tune procedure in terms of the number of hidden layers is one, with a lower bound of zero (no hidden layers) and an upper bound of two (two hidden layers). Also, the initial value for the hidden units, or neurons, is 50, with a lower bound of 50 and an upper bound of 500.

Notice under the **Training Details** section that the auto-tune does not use the prediction error weights as a default during the training process. We will leave it as-is.

Under **Optimization**, the default optimization method is based on limited memory. You also have the option based on stochastic gradient descent. You could also set a maximum time to run the auto-tune process. For now, we will keep it cleared, which means there is no time limit for the optimization process. You can also set the maximum number of iterations. Let's keep it as 250. Finally, the auto-tune process uses regularization as a method to avoid over-fitting the model during the training. You can set L1 and L2 regularization parameters, including initial values, lower bounds, and upper bounds. Let's leave the default settings.

The default validation method for the optimization is based on partition, and the proportion of validation cases is set to 0.3, or 30% of the observations. You also have the K-fold cross validation option, but let's leave it as partition in this example.

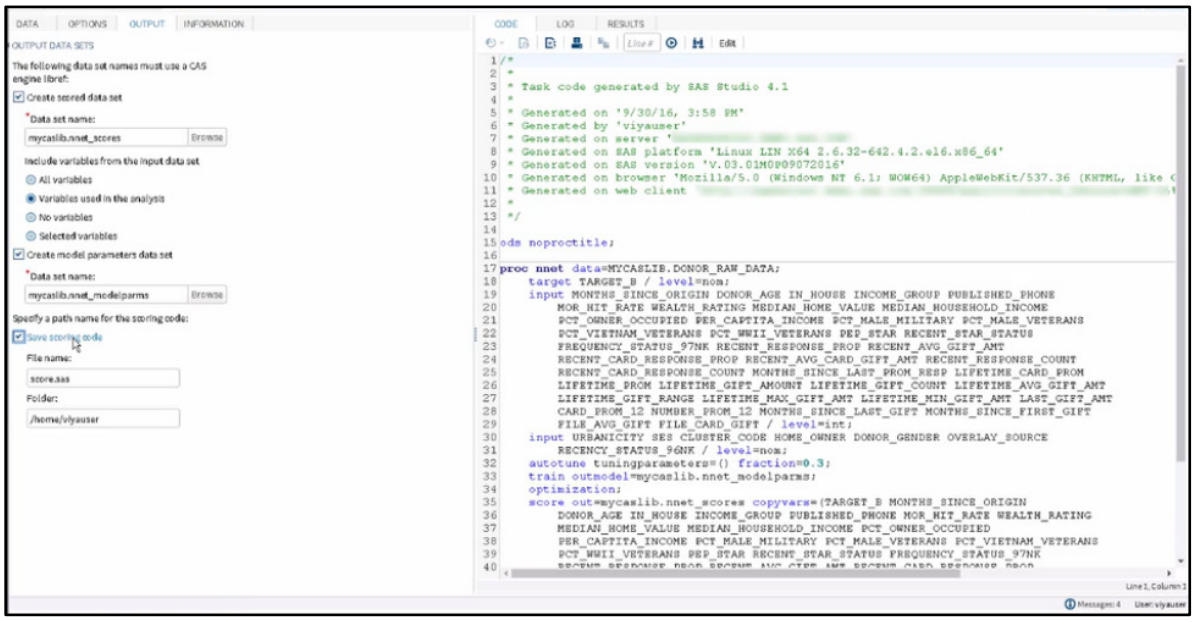
Under the **Tuning Controls** option, a few parameters are set as default. There is a maximum time of 60 minutes for the entire training process, a maximum of 5 iterations, a maximum of 50 evaluations, and a maximum of 10 evaluations per iteration. Let's leave these options as the default.

Click the **OUTPUT** tab. On the **OUTPUT DATASETS** property, select the **Create scored data set** option. When you select that option, notice that the code for PROC NNET disappears from the code panel. Also, the Run shortcut (F3) button is disabled. On the **Data set name** option, set your library and data set names. For example, enter MCASLIB.Nnet_scores. Notice that when you populate that information, the Run shortcut button is enabled and the code returns to the CODE panel. Now, the score OUT= option in the code is populated with MYCASLIB.Nnet_scores.

In the **Include variables from the input data set** option, select **Variables used in the analysis**. Select the **Create model parameters data set** option. When you select that option, notice again that the code for PROC NNET disappears from the CODE panel and the Run shortcut button is disabled. For the **Data set name option,** set your library and data set names. For example, enter MYCASLIB.Nnet_modelparms. Notice that when you populate that information, the Run shortcut button is enabled and the code returns to the CODE panel. Now the train OUTMODEL= option is populated with MCASLIB.Nnet_modelparms in the NNET procedure.

Select the **Save scoring code** option. Do not change the filename and the folder. Your screen should now look like Figure 2.11.

Figure 2.11: OUTPUT properties



Click the **Run** shortcut button. When the code is running, notice that the CODE tab changes to the LOG tab. This means that the running process activity is populated including errors, warnings, and notes. If you get any error during the process, the LOG panel will be visible with the proper information. Otherwise, if you don't get any errors, the run is over.

The RESULTS panel shows you the Iteration history for the neural network training process. Either the model converges, or it does not. A summary of the model information, a tuner information table, information about the tuner results with defaults or best configurations, and the score table information, which includes the misclassification are shown, as seen in Figure 2.12.

Figure 2.12: Partial Results of NNET Procedure

Best Configuration	
Evaluation	16
Hidden Layers	1
Neurons in Hidden Layer 1	37
Neurons in Hidden Layer 2	0
L1 Regularization	0.290168.9
L2 Regularization	0.0254383
Misclassification Error Percentage	26.92

Tuner Summary	
Initial Configuration Objective Value	27.7828
Best Configuration Objective Value	26.9161
Worst Configuration Objective Value	72.6734
Initial Configuration Evaluation Time in Seconds	3.8052
Best Configuration Evaluation Time in Seconds	24.0269
Number of Improved Configurations	4
Number of Evaluated Configurations	46
Total Tuning Time in Seconds	954.33

Tuner Task Timing		
Task	Seconds	Percent
Model Training	953.03	99.86
Model Scoring	1.29	0.15
Total Objective Evaluations	954.32	100.00
Tuner	0.01	0.00
Total Time	954.33	100.00

Score Information	
Number of Observations Read	19372
Number of Observations Used	7208
Misclassification Error (%)	26.331853496

You have now learned how to develop a supervised learning model based on a neural network to predict future donors.

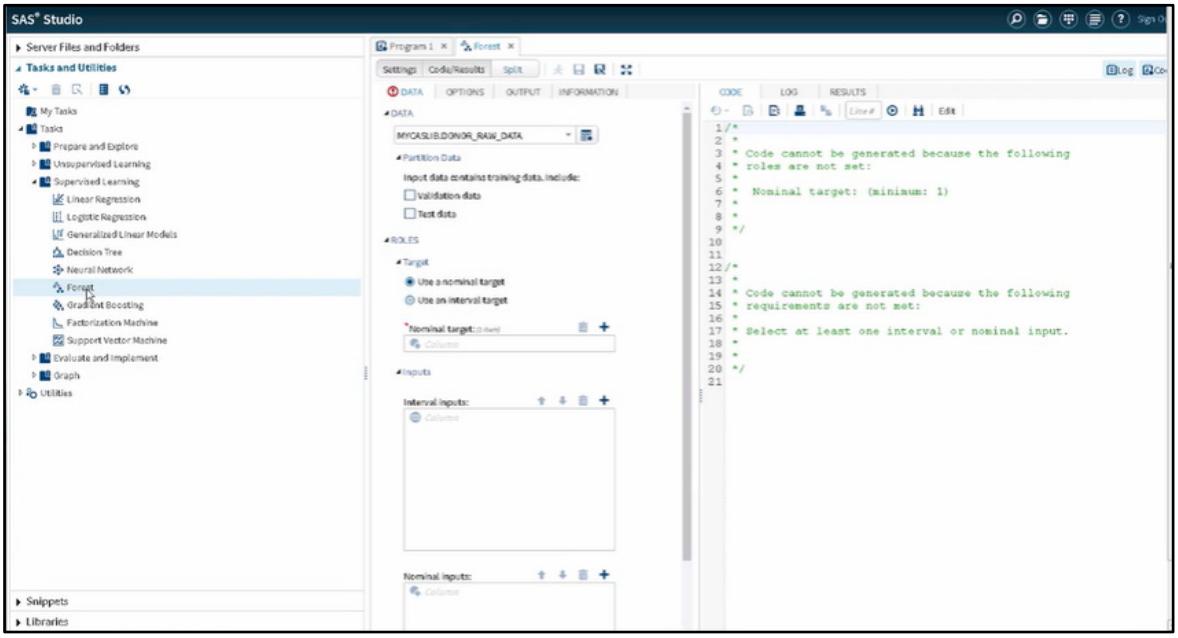
Forest Model

In this section, you will learn how to use ensemble and auto-tune methods to develop a supervised learning model. This particular example is focused on a Forest model.

The data source for this example contains a list of donors and non-donors to an organization and has a binary target variable named TARGET_B, which has a value "1" for those who donated during a mailing, and value "0" for those who did not.

To begin, in SAS Studio, click the **Tasks and Utilities** option on the left panel. Then, expand the **Tasks** option and expand the **Supervised Learning** option. From that folder, double-click the **Forest** option as shown in Figure 2.13. This opens the Forest panel at the center of your screen. You can see four tabs in the center panel: DATA, OPTIONS, OUTPUT, and INFORMATION.

Figure 2.13: Forest Task



Now let's learn how to configure the Forest and programmatically edit Forest options. On the DATA tab, in the DATA property, click the **Select a table** button to select the table that we are going to use in this example. Your table should be available in Libraries ▶ My Libraries ▶ MYCASLIB. Select the table named DONOR_RAW_DATA. Click OK to close the Select a Table window. Keep **Validation data** and **Test data** de-selected on the **Partition Data** option.

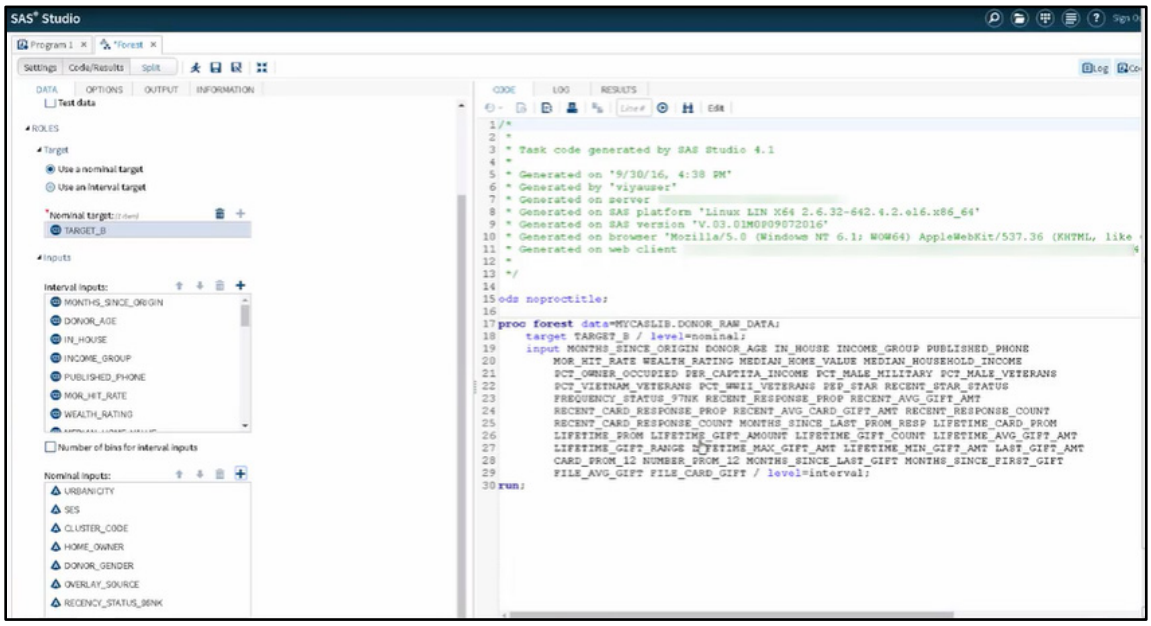
Let's go to the **ROLES** property. On the **TARGET** option, select **Use a nominal target**. On **Nominal target**, click the **Add a column** button (+) and select the variable TARGET_B in the Columns window. Click OK.

Let's go now to the Inputs. On **Interval inputs**, click the **Add columns** button (+). A pop-up menu will show all the interval variables. Click MONTHS_SINCE_ORIGIN. Scroll down to the end of the variable window selection, press Shift, and click the last variable, FILE_CARD_GIFT. This will select all interval variables, except TARGET_D. Click OK in the Columns window.

On **Nominal inputs**, click the **Add columns** button (+). A pop-up menu will show all the nominal variables. Click URBANICITY. Scroll down to the end of the variable window selection, press Shift, and click the last variable, RECENCY_STATUS_96NK. This will select all the nominal variables, except TARGET_D and CONTROL_NUMBER. Click OK in the Columns window.

Notice that the code window begins to populate with the Forest procedure as shown in Figure 2.14.

Figure 2.14: DATA Tab Options



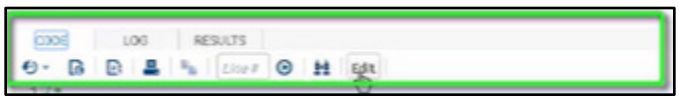
Click the **OPTIONS** tab, then click the **Run** shortcut button. This action will create a Forest with the default properties of 100 trees, where each tree is trained on 60% of the available data. Upon completion, notice that the results appear in the **RESULTS** panel.

In the Results, we see the summary of model information, which contains the Out of Bag (OOB) Misclassification Rate of 0.24958. In addition, we can observe a summary of observation counts. As you scroll down through the Results, you will see the Variable Importance Table. The last table is the Fit Statistics Table, which provides a summary of common fit statistics based on the number of trees in the forest.

Adjust Ensemble Parameters Manually

Let's attempt to improve the model performance by adjusting some ensemble parameters by editing the PROC FOREST code. The first step in editing the code is to click the **CODE** tab so that the code appears. Click the **Edit SAS Code** shortcut button to edit the code. It is the button labeled Edit on the toolbar as shown in Figure 2.15.

Figure 2.15: Edit SAS Code Shortcut Button



After you click the Edit button, notice that a new program tab appears. Program 2.1 shows the current code.

Program 2.1: PROC FOREST Code

```
ods noproctitle;

proc forest data=MYCASLIB.DONOR_RAW_DATA;
   target TARGET_B / level=nominal;
   input MONTHS_SINCE_ORIGIN DONOR_AGE IN_HOUSE INCOME_GROUP PUBLISHED_PHONE
         MOR_HIT_RATE WEALTH_RATING MEDIAN_HOME_VALUE MEDIAN_HOUSEHOLD_INCOME
         PCT_OWNER_OCCUPIED PER_CAPITA_INCOME PCT_ATTRIBUTE1 PCT_ATTRIBUTE2
         PCT_ATTRIBUTE3 PCT_ATTRIBUTE4 PEP_STAR RECENT_STAR_STATUS
         FREQUENCY_STATUS_97NK RECENT_RESPONSE_PROP RECENT_AVG_GIFT_AMT
         RECENT_CARD_RESPONSE_PROP RECENT_AVG_CARD_GIFT_AMT RECENT_RESPONSE_COUNT
         RECENT_CARD_RESPONSE_COUNT MONTHS_SINCE_LAST_PROM_RESP LIFETIME_CARD_PROM
         LIFETIME_PROM LIFETIME_GIFT_AMOUNT LIFTEIME_GIFT_COUNT LIFETIME_AVG_GIFT_AMT
         LIFETIME_GIFT_RANGE LIFETIME_MAX_GIFT_AMT LIFETIME_MIN_GIFT_AMT
         LAST_GIFT_AMT CARD_PROM_12 NUMBER_PROM_12 MONTHS_SINCE_LAST_GIFT
         MONTHS_SINCE_FIRST_GIFT FILE_AVG_GIFT FILE_CARD_GIFT / level=interval;
```

```
    input URBANICITY SES CLUSTER_CODE HOME_OWNER DONOR_GENDER OVERLAY_SOURCE
          RECENCY_STATUS_96NK / level=nominal;
run;
```

Let's now adjust the properties of the Forest procedure by adding two options after the DATA= statement. Program 2.2 shows the adjusted code with additions highlighted in gray.

Program 2.2: Adjusted PROC FOREST Code

```
ods noproctitle;

proc forest data=MYCASLIB.DONOR_RAW_DATA ntrees=200 minleafsize=12;❶
   target TARGET_B / level=nominal;
   input MONTHS_SINCE_ORIGIN DONOR_AGE IN_HOUSE INCOME_GROUP PUBLISHED_PHONE
         MOR_HIT_RATE WEALTH_RATING MEDIAN_HOME_VALUE MEDIAN_HOUSEHOLD_INCOME
         PCT_OWNER_OCCUPIED PER_CAPITA_INCOME PCT_ATTRIBUTE1 PCT_ATTRIBUTE2
         PCT_ATTRIBUTE3 PCT_ATTRIBUTE4 PEP_STAR RECENT_STAR_STATUS
         FREQUENCY_STATUS_97NK RECENT_RESPONSE_PROP RECENT_AVG_GIFT_AMT
         RECENT_CARD_RESPONSE_PROP RECENT_AVG_CARD_GIFT_AMT RECENT_RESPONSE_COUNT
         RECENT_CARD_RESPONSE_COUNT MONTHS_SINCE_LAST_PROM_RESP LIFETIME_CARD_PROM
         LIFETIME_PROM LIFETIME_GIFT_AMOUNT LIFTEIME_GIFT_COUNT LIFETIME_AVG_GIFT_AMT
         LIFETIME_GIFT_RANGE LIFETIME_MAX_GIFT_AMT LIFETIME_MIN_GIFT_AMT
         LAST_GIFT_AMT CARD_PROM_12 NUMBER_PROM_12 MONTHS_SINCE_LAST_GIFT
         MONTHS_SINCE_FIRST_GIFT FILE_AVG_GIFT FILE_CARD_GIFT / level=interval;
    input URBANICITY SES CLUSTER_CODE HOME_OWNER DONOR_GENDER OVERLAY_SOURCE
RECENCY_STATUS_96NK / level=nominal;
```

❶ The ntrees=200 option will increase the number of trees in the Forest to 200. The minleafsize=12 option will require that greater than or equal to 12 observations be present in the leaf node following the split. The value is approximately 0.1% of the training sample used to construct each tree.

After you have adjusted the code, click the **Run** shortcut button. Notice in the Results that the Out of Bag Misclassification rate has improved to 0.24912.

Auto-tune Options

Close the Program window and return to the **OPTIONS** tab in the left pane to make adjustments to the code using the options in the task. Under the **METHODS** property, select the option **Automatically tune selected options**.

The auto-tune process sets the initial number of trees to 20 with a lower bound at 20 and an upper bound at 150. It means that several different numbers of trees between 20 and 150 will be tried in order to find the best configuration. Let's leave it as it is.

The initial bootstrap sample rate is 0.1, meaning that only 10% of the cases will be included in the bootstrap sample for the training process. The lower bound is set to 0.1 and the upper bound is set to 0.9, meaning the bootstrap sampling will consider up to 90% of the cases on the training process. Let's keep the default settings.

During the auto-tune process, the number of inputs to be used during the training process can vary from 1 to 100 – as defined by the lower and upper bounds. Let's keep the default settings for the bounds, and set the initial value to 1.

The default splitting criteria is based on the information gain ratio. You also have criteria options based on CHAID, Chi-square, Entropy, and Gini index. Let's leave it as is.

By default, the maximum depth of a tree is 20, the minimum number of observations for a leaf is 5, and the maximum number of branches is 2. In other words, the tree will have a maximum depth of 20, there will be no leaves with fewer than 5 observations, and the splitting process will be a two-way split – a binary split.

The default variable importance method is Gini, which means that all input variables will be measured important to the model contributing to the prediction or decision based on the Gini criterion. The method to calculate the predicted nominal target levels will be based on the average posterior probability of each level considering all trees. You can also set it as the majority, which is the percentage of trees that predict each level. Let's leave it as probability.

In the **Sampling** section, you can specify a random number seed. For this example, do not select it.

The default validation method used by the auto-tune process is based on the partition data. The default proportion is 0.3 or 30% of the cases. You can also set the validation method to k-fold cross validation. For now, let's leave it as partition.

The tuning options for the auto-tune process are set at 60 minutes as the maximum running time – considering 5 iterations, 50 evaluations, and 10 evaluations per iteration. Let's keep the default options.

In the **PLOTS** section, you can request that plots be created for misclassifications by the number of trees and by variable importance. Do not select either condition for this example.

Click the **OUTPUT** tab. In the **OUTPUT DATASETS** section, select the **Create scored data set** option. When you select that option, notice that the code for PROC FOREST disappears from the code panel. Also, the Run (F3) shortcut button is disabled. On the **Data set name** option, set your library and data set names. For example, enter MYCASLIB.Forest_scores. Notice that when you populate that information, the Run shortcut button is enabled and the code returns to the CODE panel. Now the score OUT= option in the code is populated with MYCASLIB.Forest_scores. In the Include variables from the input data set option, select **Variables used in the analysis**.

Select the **Create model data set** option. When you select that option, notice again that the code for PROC FOREST disappears from the CODE panel and the Run shortcut button is disabled. On the data set name option, set your library and data set names. For example, enter MYCASLIB.Forest_model. Notice that when you populate that information, the Run shortcut button is enabled and the code returns to the CODE panel. Now the OUT= model option is populated with MYCASLIB.Forest_model after the DATA= option in the FOREST procedure.

Select the **Saving scoring model** option. When you select that option, notice that the code for PROC FOREST disappears from the code panel. Also, the Run (F3) shortcut button is disabled. On the **Scoring model data set name** option, set your library and data set names. For example, enter MYCASLIB.ForestScoringModel. Notice that when you populate that information, the Run shortcut button is enabled and the code returns to the CODE panel. In the **Include variables from the input data set** option, select **Variables used in the analysis**. Now the SAVESTATE RSTORE= option in the code is populated with MYCASLIB.ForestScoringModel. Notice that the ID option in the code is populated with the variables used in the analysis.

Select the **Save fit statistics** option. Do not change the data set name. Select the Save variable importance measures option. Do not change the data set name here either.

Click the **Run** shortcut button. While the code is running, notice that the CODE tab changes to the LOG tab. This means that the running process activity is populated including errors, warnings, and notes. If you get any errors during the process, the LOG panel will be visible with the proper information. Otherwise, if you don't get any errors, when the run is over the Results show you the model information.

The Results (Figure 2.16) include information about the number of trees, variables per tree, bootstrap percentage, and some additional information about the final forest – a variable importance table describing the importance of each input for the final forest; a fit statistics table, including average squared error and misclassification error for out-of-bag and training; and a series of tuning information, including a summary of the model, time consumed for the tasks, the best configurations achieved, tuner results, and the output tables.

Figure 2.16: Partial Results of FOREST Procedure

Model Information	
Number of Trees	200
Number of Variables Per Split	7
Seed	1983130769
Bootstrap Percentage	60
Number of Bins	20
Number of Input Variables	47
Maximum Number of Tree Nodes	397
Minimum Number of Tree Nodes	75
Maximum Number of Branches	2
Minimum Number of Branches	2
Maximum Depth	20
Minimum Depth	20
Maximum Number of Leaves	199
Minimum Number of Leaves	38
Maximum Leaf Size	10262
Minimum Leaf Size	12
OOB Misclassification Rate	0.24912244

	Training
Number of Observations Read	19372
Number of Observations Used	19372

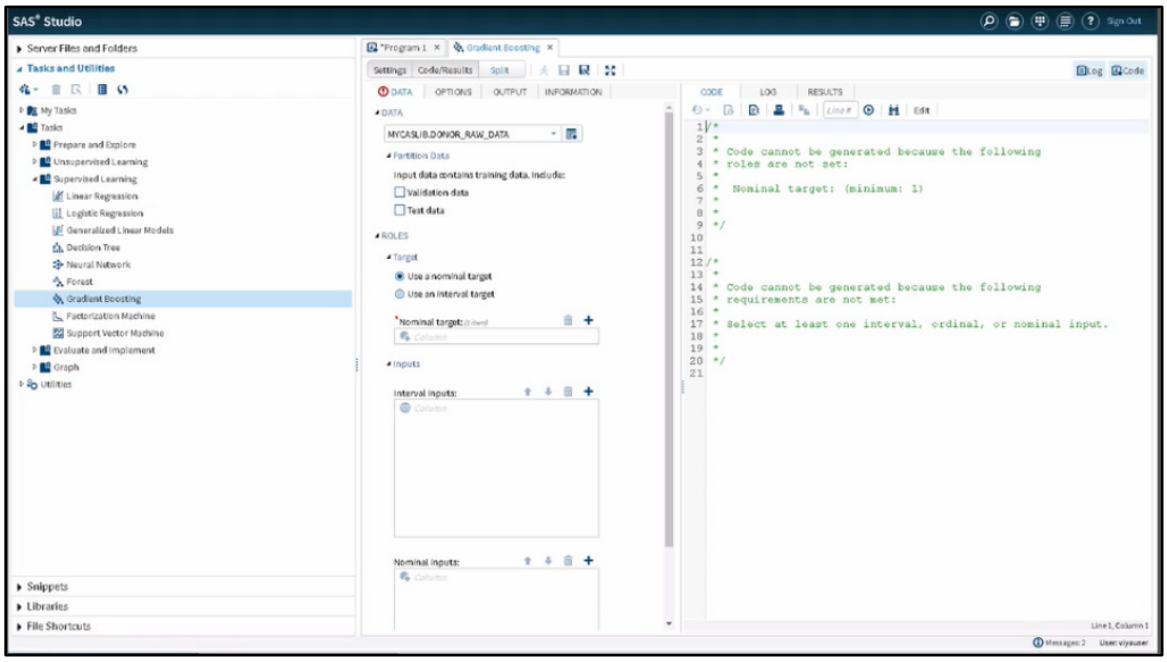
Gradient Boosting

In this section, you will learn how to use ensemble methods when developing a supervised learning model in SAS Viya. This particular example is focused on gradient boosting.

The data source for this example contains a list of donors and non-donors to an organization, and has a binary target variable named TARGET_B, which has a value “1” for those who donated during a mailing, and a value “0” for those who did not.

To begin, in SAS Studio click the **Tasks and Utilities** option on the left panel. Then, expand the **Tasks** option and expand the **Supervised Learning** option. From that folder, double-click the **Gradient Boosting** option as shown in Figure 2.17. This opens the Gradient Boosting panel at the center of your screen. You can see four tabs in the center panel: DATA, OPTIONS, OUTPUT, and INFORMATION.

Figure 2.17: Gradient Boosting Task



In this section, you will learn how to configure the Gradient Boosting task and programmatically edit some options in an attempt to improve model performance. On the DATA tab, in the DATA section, click the **Select a table** shortcut button to select the table that will be used in this example. Your table should be available in Libraries → My Libraries → MYCASLIB. Select the table named DONOR_RAW_DATA. Click OK to close the **Select a Table** window.

In the Partition Data section, select the **Validation data** check box. Enter 0.4 for **Proportion of validation cases**. These actions partition the raw data so that 60% is available for training and 40% is available for validation. Set a random seed by selecting the check box next to **Random number seed**. Enter the value 12345 for the random seed.

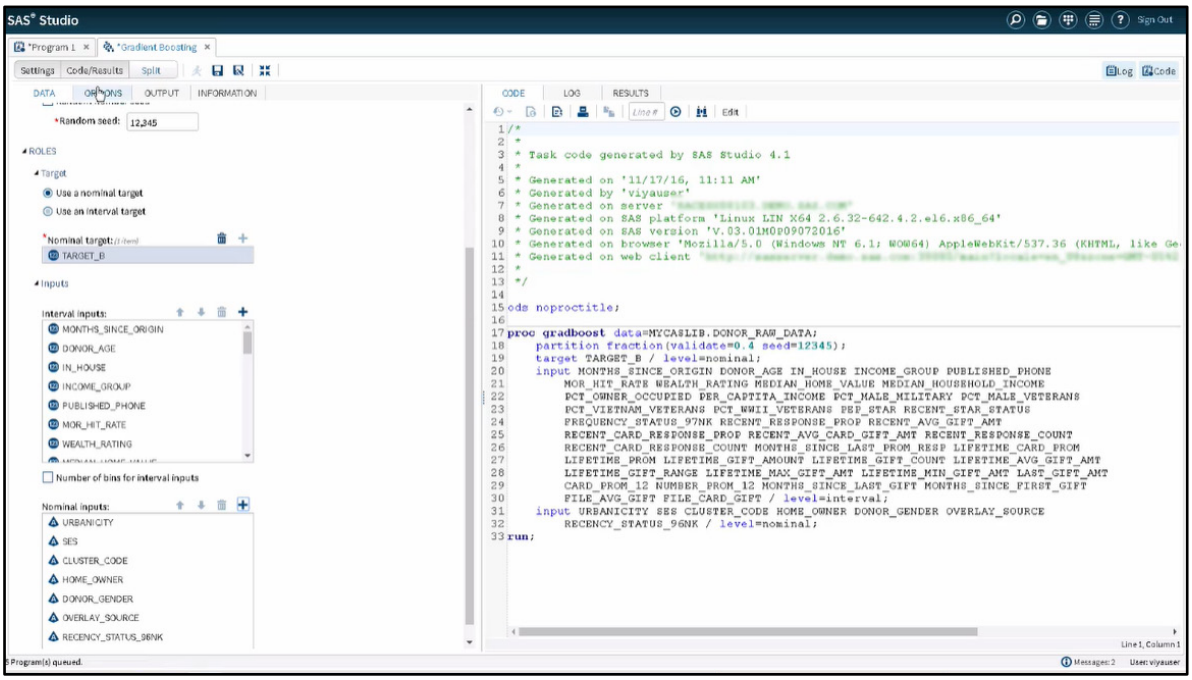
Under the **ROLES** section, in the **Target** properties, select **Use a nominal target**. Next to the **Nominal target** property, click the **Add a column button** (+) and select the variable TARGET_B. Click OK to select the target.

Let's go now to the Inputs. On **Interval inputs**, click the **Add columns** button (+). A pop-up menu will show all the interval variables. Click MONTHS_SINCE_ORIGIN. Scroll down to the end of the variable window selection, press Shift, and click the last variable, FILE_CARD_GIFT. This will select all interval variables, except TARGET_D. Click OK in the Columns window. Notice that the code window begins to populate with the Gradient Boosting procedure.

On **Nominal inputs**, click the **Add columns button** (+). A pop-up menu will show all the nominal variables. Click URBANICITY. Scroll down to the end of the variable window selection, press Shift, and click the last variable, RECENCY_STATUS_96NK. This will select all the nominal variables, except TARGET_D and CONTROL_NUMBER. Click OK in the Columns window.

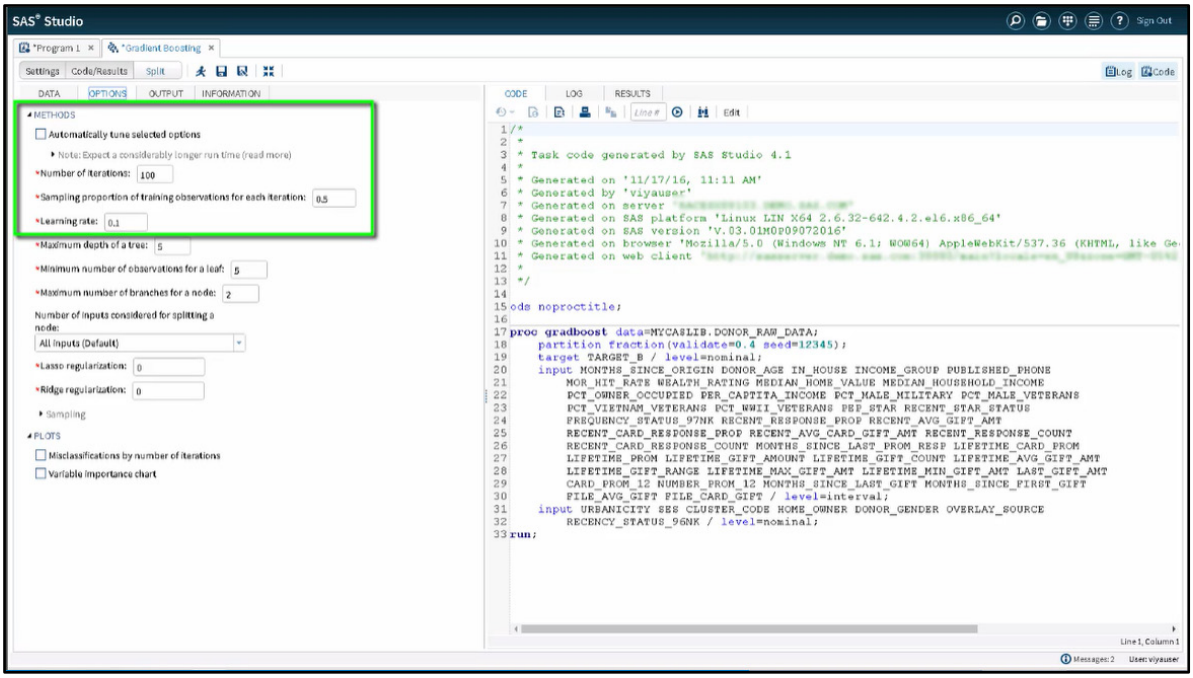
The code window further populates, as shown in Figure 2.18.

Figure 2.18: DATA tab options



Click the **OPTIONS** tab. Under the **Methods** property, as shown in Figure 2.19 notice that by default 100 iterations are performed. This is equivalent to the number of trees that are constructed. The learning rate is 0.1.

Figure 2.19: OPTIONS Tab Options



Click the running man icon or press F3 to run the task. Notice in the right window that the log file is initially shown until the run is complete. If there are no errors, the results are shown. At the top of the results, you will see a table summarizing the model information. Under this table is a summary of observation counts. Next is a table providing variable importance information. And finally, at the end of the results is a table of Fit Statistics. Notice that the validation misclassification rate for the 100th iteration is approximately 0.253.

Let's see if we can reduce the misclassification rate by changing a few of the settings for the Gradient Boosting model by changing the code. In the right window, click the CODE tab. Click the **Edit** button to view the code in an Editor window.

After you click the Edit button, notice that a new program tab appears. Program 2.3 shows the current code.

Program 2.3: PROC GRADBOOST

```
ods noproctitle;

proc gradboost data=MYCASLIB.DONOR_RAW_DATA;
   partition fraction(validate=0.4 seed=12345);
   target TARGET_B / level=nominal;
   input MONTHS_SINCE_ORIGIN DONOR_AGE IN_HOUSE INCOME_GROUP PUBLISHED_PHONE
         MOR_HIT_RATE WEALTH_RATING MEDIAN_HOME_VALUE MEDIAN_HOUSEHOLD_INCOME
         PCT_OWNER_OCCUPIED PER_CAPITA_INCOME PCT_ATTRIBUTE1 PCT_ATTRIBUTE2
         PCT_ATTRIBUTE3 PCT_ATTRIBUTE4 PEP_STAR RECENT_STAR_STATUS
         FREQUENCY_STATUS_97NK RECENT_RESPONSE_PROP RECENT_AVG_GIFT_AMT
         RECENT_CARD_RESPONSE_PROP RECENT_AVG_CARD_GIFT_AMT RECENT_RESPONSE_COUNT
         RECENT_CARD_RESPONSE_COUNT MONTHS_SINCE_LAST_PROM_RESP LIFETIME_CARD_PROM
         LIFETIME_PROM LIFETIME_GIFT_AMOUNT LIFTEIME_GIFT_COUNT LIFETIME_AVG_GIFT_AMT
         LIFETIME_GIFT_RANGE LIFETIME_MAX_GIFT_AMT LIFETIME_MIN_GIFT_AMT
         LAST_GIFT_AMT CARD_PROM_12 NUMBER_PROM_12 MONTHS_SINCE_LAST_GIFT
         MONTHS_SINCE_FIRST_GIFT FILE_AVG_GIFT FILE_CARD_GIFT / level=interval;
    input URBANICITY SES CLUSTER_CODE HOME_OWNER DONOR_GENDER OVERLAY_SOURCE
          RECENCY_STATUS_96NK / level=nominal;
run;
```

In an attempt to improve the performance of the model, we will increase the number of iterations or trees to 200 and decrease the learning rate to 0.01. To do this, at the beginning of the GRADBOOST procedure, next to the DATA= option, type NTREES=200 and LEARNINGRATE=0.01, as shown highlighted in gray in Program 2.4.

Program 2.4: Adjusted PROC GRADBOOST

```
ods noproctitle;

proc gradboost data=MYCASLIB.DONOR_RAW_DATA ntrees=200 learningrate=0.01;
   partition fraction(validate=0.4 seed=12345);
   target TARGET_B / level=nominal;
   input MONTHS_SINCE_ORIGIN DONOR_AGE IN_HOUSE INCOME_GROUP PUBLISHED_PHONE
         MOR_HIT_RATE WEALTH_RATING MEDIAN_HOME_VALUE MEDIAN_HOUSEHOLD_INCOME
         PCT_OWNER_OCCUPIED PER_CAPITA_INCOME PCT_ATTRIBUTE1 PCT_ATTRIBUTE2
         PCT_ATTRIBUTE3 PCT_ATTRIBUTE4 PEP_STAR RECENT_STAR_STATUS
         FREQUENCY_STATUS_97NK RECENT_RESPONSE_PROP RECENT_AVG_GIFT_AMT
         RECENT_CARD_RESPONSE_PROP RECENT_AVG_CARD_GIFT_AMT RECENT_RESPONSE_COUNT
         RECENT_CARD_RESPONSE_COUNT MONTHS_SINCE_LAST_PROM_RESP LIFETIME_CARD_PROM
         LIFETIME_PROM LIFETIME_GIFT_AMOUNT LIFTEIME_GIFT_COUNT LIFETIME_AVG_GIFT_AMT
         LIFETIME_GIFT_RANGE LIFETIME_MAX_GIFT_AMT LIFETIME_MIN_GIFT_AMT
         LAST_GIFT_AMT CARD_PROM_12 NUMBER_PROM_12 MONTHS_SINCE_LAST_GIFT
         MONTHS_SINCE_FIRST_GIFT FILE_AVG_GIFT FILE_CARD_GIFT / level=interval;
    input URBANICITY SES CLUSTER_CODE HOME_OWNER DONOR_GENDER OVERLAY_SOURCE
          RECENCY_STATUS_96NK / level=nominal;
run;
```

Click the **Run** shortcut button or press F3 to run the code. When the code is finished running, the results will be displayed. Notice that by scrolling to the end of the Results window, the performance of the model did improve when the validation misclassification rate dropped to 0.239 on the 200th iteration.

Resources

This chapter is based on the "Data Mining and Machine Learning on SAS Viya" videos in SAS® Viya® Enablement, a free course available from SAS Education.

You might find the following documentation and resources helpful as you learn more about programming in SAS Visual Data Mining and Machine Learning:

- SAS Visual Data Mining and Machine Learning 8.1: Data Mining and Machine Learning Procedures
- Link to Donor data set available to download from SAS Visual Data Mining and Machine Learning documentation: https://support.sas.com/documentation/prod-p/vdmml/zip/567885_donor_data.zip

Chapter 3: Advanced Data Mining and Machine Learning Procedures

Introduction

SAS offers a collection of new, high-performance CAS procedures that are compatible with a multi-threaded approach. In the previous chapter, we looked at tasks in SAS Studio that help automate your programming so that you do not have to manually write your code. However, there are three options for manually writing your programs in SAS Viya:

- **SAS Studio** provides a SAS programming environment for developing and submitting programs to the server.
- **Batch submission** is also still an option.
- **Open-source languages** such as Python, Lua, and Java can submit code to the CAS server.

In this chapter, you will learn the syntax for three of the new, advanced data mining and machine learning procedures: factorization machines, text mining, and community detection with the NETWORK procedure.

Factorization Machines

In this section, you will learn about factorization machines. What are factorization machines? They are a relatively new and powerful tool that enables you to model high-dimensional, sparse data. For example, consider a recommendation problem where users rate items that they have purchased. You have a matrix with rows that correspond to users and columns that correspond to items. There are many users and many items, so the matrix is large, but the ratings are very sparse. Most of the entries in the matrix are missing. Our purpose would be to predict missing entries so that we can find items that each user is likely to rate highly and then recommend those items to each user.

To accomplish our purpose, we will model the ratings using factorization machines. Each rating is assumed to be the sum of the following:

- a global bias
- the average rating over all users and items
- a per-user bias
- the average of the ratings given by the user
- a per-item bias

- the average of the ratings given to that item
- a pairwise interaction term that accounts for the affinity between the user and that particular item

The affinity between users and items is modeled as the inner product between a two vector of features, one for the user and one for the item. The factorization machine's procedure learns the biases and the features for all users and all items. These features are collectively known as factors.

This example uses a data set based on the *MovieLens* data set developed by the GroupLens project at the University of Minnesota. Data sets are available to download at http://grouplens.org/datasets/movielens. In this example, we will look at the MOVLENS10K data set, which has already been loaded in the MYCASLIB library.

Data Exploration

Our first objective is to use PROC PRINT to explore the data. Run just the first PROC PRINT statement shown in Program 3.1 to view a subset of the variables.

Program 3.1: Data Exploration Part 1

```
proc print data=mycaslib.movlens10k(obs=50);
run;
```

The Movie Lens 10k data set contains information about ratings given by users to movies. Users are identified by anonymous numeric IDs. The same happens with movies in the item variable. A rating is a numeric value between 1 (hated the movie) and 5 (loved the movie), as shown in Output 3.1.

Output 3.1: Partial Results of Program 3.1

Obs	user	item	rating	timestamp
1	196	242	3	881250949
2	102	768	2	883748450
3	59	196	5	888205088
4	299	229	3	878192429
5	160	174	5	876860807
6	250	140	3	878092059
7	213	273	5	878870987
8	200	673	5	884128554
9	52	280	3	882922806
10	200	318	5	884128458
11	100	344	4	891374868
12	158	177	4	880134407
13	145	15	2	875270655
14	254	662	4	887347350
15	177	333	4	880130397
16	234	612	3	892079140
17	222	97	4	878181739
18	13	892	3	882774224
19	303	252	3	879544791
20	63	762	3	875747688
21	181	306	1	878962006
22	231	181	4	888605273
23	121	740	3	891390544

Again, if you think of users and movies as a matrix, you will find that the data consists of one observation per rating, or one entry in the matrix. Let's use PROC CARDINALITY to gauge the size of the problem as shown in Program 3.2.

Program 3.2: Data Exploration Part 2

```
proc cardinality data=mycaslib.movlens10k maxlevels=254 outcard=movlens10k_card;
   vars user item rating;
run;

proc print data=movlens10k_card;
run;
```

```
proc print data=mycaslib.movlens10k(where=(user=99));
run;
```

There are 385 users and 1,254 movies in this example. This means a total of 482,790 possible ratings, but we only know 10,000. This is a very sparse data set. For example, looking at user 99, we can see that he or she has rated only 45 out of all 1254 movies.

Modeling

We will now use PROC FACTMAC to learn a factorization machine model.

Program 3.3: Factorization Machine Model

```
proc factmac data=mycaslib.movlens10k outmodel=movlens10k_factors ❶
  maxiter=20 ❷
  nfactors=20 ❸
  learnstep=0.15 ❹
  seed=12345;

  input user item / level=nominal;
  target rating / level=interval;

  output data=mycaslib.movlens10k_scored copyvar=(user item rating);
  code file="&outdir./factmac1.sas";
run;
```

❶ The model itself is saved in a data set specified by the OUTMODEL= option.

❷ You need to select a maximum number of iterations. Set MAXITER= to 20. You can increase the MAXITER= value until you are satisfied with the solution.

❸ You can decide how many factors to learn. NFACTORS=20 is a good value for this example.

❹ You also need to select a learning step parameter. Set LEARNSTEP= to 0.15.

After running the procedure, you are able to look at the Iteration History Table to see how rapidly the approximate loss decreases, as shown in Output 3.3. This is a good way to assess whether the learning step is adequate. Also importantly, you can look at the final exact loss to determine whether the MSE and RMSE are satisfactory. For example, if RMSE is less than 1 on ratings between 1 and 5 stars, that means that, on average, the model is off by one star or less.

Output 3.3: Partial Results of Program 3.3

Iteration History

Iteration	Approximate Loss
1	28.807294829
2	6.1394142181
3	3.3613305683
4	2.3010589705
5	1.7557682462
6	1.429581961
7	1.2132181573
8	1.0596532801
9	0.945251331
10	0.8570894574
11	0.7873194406
12	0.7308159717
13	0.684152751
14	0.6449754808
15	0.6116171422
16	0.5828646232
17	0.557794898
18	0.535715547
19	0.5160970831
20	0.4985201606

Final Exact Loss	
MSE	0.456311
RMSE	0.675508

Scoring

You can also see how closely the model predicted the ratings from the training set by looking at the score output in the data set specified by the OUTPUT statement from Program 3.3. Back to our User 99. You are now able to predict how he or she would rate every movie in our data set. You can do this by running the score code automatically generated by PROC FACTMAC in the file specified in the CODE statement as shown in Program 3.4. This enables you to recommend new movies. For example, you could sort by decreasing order and show only the top 10 highest predicted ratings, excluding the movies already rated by the same user.

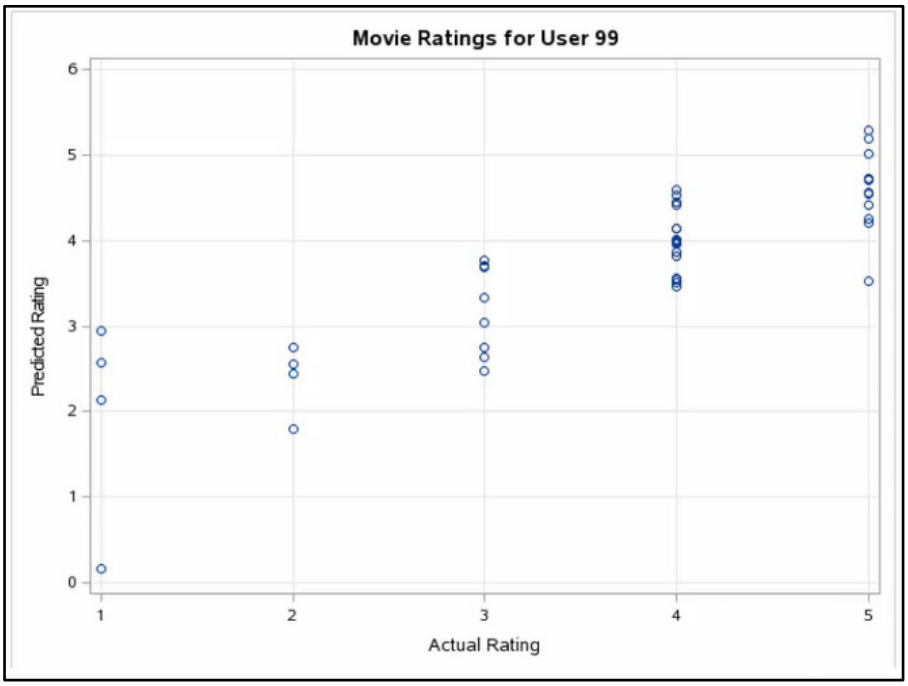
Program 3.4 also shows how to use PROC SGPLOT to visualize the quality of the predictions by plotting actual versus predicted ratings for the same user, as shown in Output 3.4.

Program 3.4: Score and Plot Predicted Versus Actual Ratings

```
/* Score all items for user=99*/
data mycaslib.movlens10k_user99_scored;
  set mycaslib.movlens10k_user99;
  %include "outdir./factmac1.sas";
run;

data recommendations99;
  set mycaslib.movlens10k_user99_scored;
  where rating is not missing;
run;

/* Plot the predicted vs actual ratings*/
proc sgplot data=MYCASLIB.MOVLENS10K_USER99_SCORED;
  title 'Movie Ratings for User 99';
```

```
    scatter x=rating y=p_rating / transparency=0.0 name='Scatter'
    xaxis grid label='Actual Rating';
    yaxis grid label='Predicted Rating' min=0 max=6;
run;
title;
```

Output 3.4: Results of Program 3.4



In conclusion, you can use factorization machines for recommendations or, more generally, for predictive modeling with interval targets when you have very sparse data organized as a matrix.

Text Mining

The TEXTMINE and TMSCORE procedures integrate the functionalities from both natural language processing and statistical analysis to provide essential functionalities for text mining. The procedures support essential natural language processing (NLP) features such as tokenizing, stemming, part-of-speech tagging, entity recognition, customized stop list, and so on. They also support dimensionality reduction and topic discovery through Singular Value Decomposition.

In this example, you will learn about some of the essential functionalities of PROC TEXTMINE and PROC TMSCORE by using a text data set containing 1,830 Amazon reviews of electronic gaming systems. The data set is named Amazon. You can find similar data sets of Amazon reviews at https://snap.stanford.edu/data/web-Amazon.html.

PROC TEXTMINE

The Amazon data set has already been loaded into CAS. The review content is stored in the variable ReviewBody, and we generate a unique review ID for each review. In the proc call shown in Program 3.5 we ask PROC TEXTMINE to do three tasks:

1. parse the documents in table reviews and generate the term by document matrix
2. perform dimensionality reduction via Singular Value Decomposition
3. perform topic discovery based on Singular Value Decomposition results

Program 3.5: PROC TEXTMINE

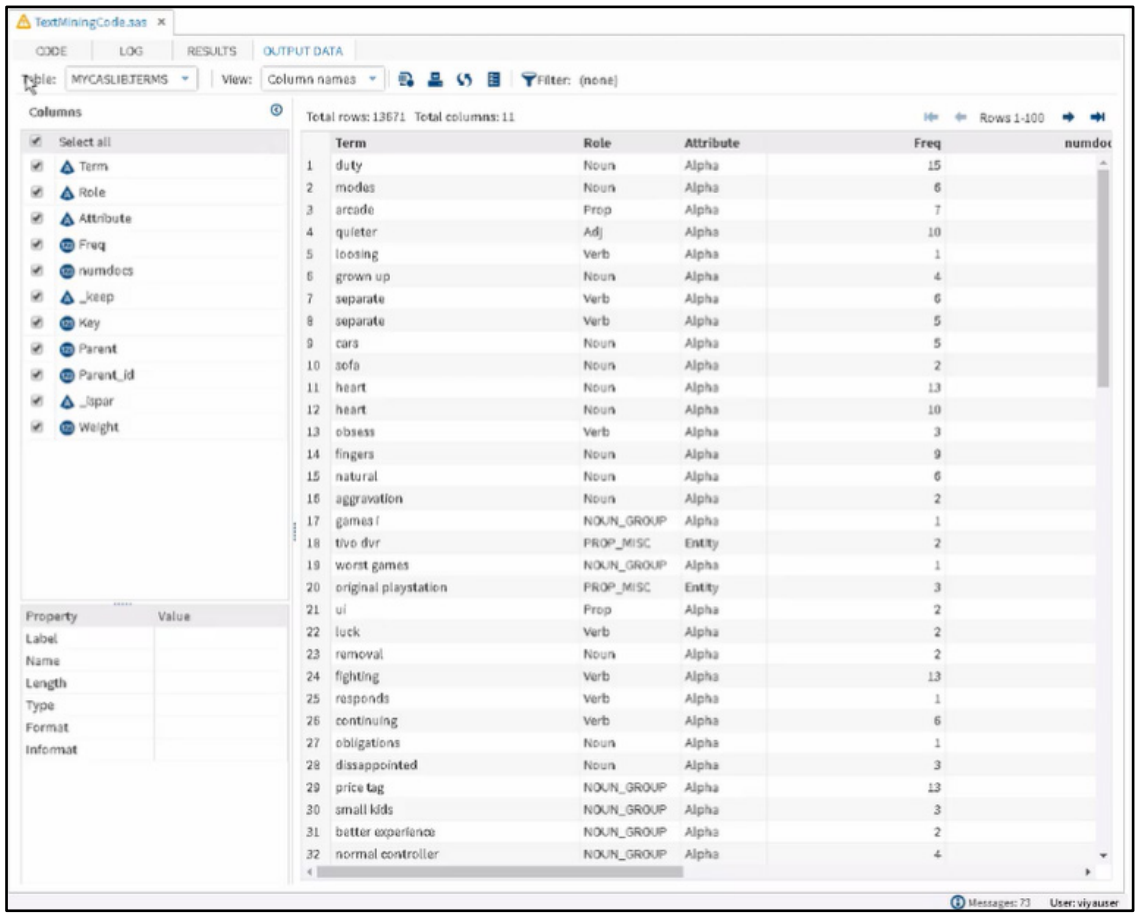
```
data mycaslib.amazon;
    set mylib.amazon;
run;
data mycaslib.engstop;
    set mylib.engstop;
run;

proc textmine data=mycaslib.amazon;
    doc_id id;
    var reviewbody;
  ❶ parse reducef=2 entities=std stoplist=mycaslib.engstop
          outterms=mycaslib.terms outparent=mycaslib.parent
          outconfig=mycaslib.config;
  ❷ svd k=10 svdu=mycaslib.svdu outdocpro=mycaslib.docpro
          outtopics=mycaslib.topics;
run;
```

❶ The first task (parsing) is specified in the PARSE statement. Parameter "reducef" specifies the minimum number of times a term needs to appear in the text to be included in the analysis. Parameter "stop" specifies a list of terms to be excluded from the analysis, such as "the", "this", and "that". Outparent is the output table that stores the term by document matrix, and Outterms is the output table that stores the information of terms that are included in the term by document matrix. Outconfig is the output table that stores configuration information for future scoring.

❷ Tasks 2 and 3 (dimensionality reduction and topic discovery) are specified in the SVD statement. Parameter K specifies the desired number of dimensions and number of topics. Parameter SVDU is the output table that stores the U matrix from SVD calculations, which is needed in future scoring. Parameter OutDocPro is the output table that stores the new matrix with reduced dimensions. Parameter OutTopics specifies the output table that stores the topics discovered.

Click the Run shortcut button or press F3 to run Program 3.5. The terms table shown in Output 3.5 stores the tagging, stemming, and entity recognition results. It also stores the number of times each term appears in the text data.

Output 3.5: Results from Program 3.5



SQL Queries

The SQL queries in Program 3.6 show examples of stemming, topic discovery, and entity recognition.

Program 3.6: SQL Queries

```
proc SQL;
select * from mycaslib.terms where term in ('games', 'game') and role='Noun'; ❶

proc SQL outobs=10;
select * from mycaslib.terms where role in ('Noun')
        order by numdocs descending; ❷

proc SQL outobs=10;
select * from mycaslib.terms where attribute='Entity'
        order by numdocs descending; ❸

proc print data=mycaslib.docpro (obs=5);
run;

proc print data=mycaslib.topics;
run;
```

❶ The first SQL query in Program 3.6 shows an example of stemming. We can see that when the term "games" is used as a noun, it stems the term "game." By using stemming, variations of the same parent term can be grouped together. Stemming helps alleviate the data sparsity problem and also serves as a mechanism for dimensionality reduction. The results in Output 3.6 include a table that shows the stemming results for the term "game". You can get an idea of the general content of a document collection by examining the terms that appear in the most documents.

❷ The next SQL query shows the top 10 Noun terms that appear in the reviews. Based on the query results shown as the second table in Output 3.6, we can tell the reviews are mostly about games and gaming consoles.

❸ Taking advantage of language features such as parts of speech and entities, you can continue to enhance your understanding of the nature of the document collection. The last SQL query shows the top 10 entities that appear in the reviews. Based on the query results shown as the third table in Output 3.6, we can tell if the reviews are mostly about gaming systems and other electronic items. PROP_MISC in the second column represents a miscellaneous proper noun.

Output 3.6: Results from Program 3.6

Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
game	Noun	Alpha	3079	96	Y	5472	.	5472	+	0.1694160914
game	Noun	Alpha	1006	90	Y	5472	5472	5472	.	0.1694160914
games	Noun	Alpha	2073	96	Y	10760	5472	5472	.	0.1693939571

Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
game	Noun	Alpha	3079	96	Y	5472	.	5472	+	0.1694160914
games	Noun	Alpha	2073	96	Y	10760	5472	5472	.	0.1693939571
console	Noun	Alpha	992	95	Y	1863	1863	1863	.	0.1346589498
console	Noun	Alpha	1169	95	Y	1863	.	1863	+	0.1346589498
system	Noun	Alpha	1465	91	Y	6279	6279	6279	.	0.1758053438
system	Noun	Alpha	1691	91	Y	6279	.	6279	+	0.1758053438
game	Noun	Alpha	1006	90	Y	5472	5472	5472	.	0.1694160914
thing	Noun	Alpha	544	89	Y	4140	.	4140	+	0.1369832947
time	Noun	Alpha	672	87	Y	3866	.	3866	+	0.1468943588
year	Noun	Alpha	532	85	Y	5281	.	5281	+	0.1357440856

Term	Role	Attribute	Freq	numdocs	_keep	Key	Parent	Parent_id	_ispar	Weight
ps3	PROP_MISC	Entity	1287	88	Y	6104	.	6104		0.1467909529
xbox	PROP_MISC	Entity	1006	85	Y	3428	.	3428		0.1181900425
wii	PROP_MISC	Entity	1617	71	Y	2756	.	2756		0.266667225
sony	PROP_MISC	Entity	246	65	Y	6504	.	6504		0.1591016995
microsoft	COMPANY	Entity	240	57	Y	5917	.	5917		0.1936881836
netflix	PROP_MISC	Entity	165	53	Y	2150	.	2150		0.2090257093
playstation	PROP_MISC	Entity	188	48	Y	4811	.	4811		0.2421056929
well	LOCATION	Entity	93	47	Y	5079	.	5079		0.2210788812
amazon	LOCATION	Entity	160	46	Y	813	.	813		0.2504696462
nintendo	PROP_MISC	Entity	409	46	Y	5774	.	5774		0.3132811571

Entities are based on complex rules that exploit context to derive additional information about the nature of the term. As you can see from the Amazon entry in the third table in Output 3.6, the Role has a value of "Location". Some of the rules are hardcoded based on common usage, so the location "Amazon" as a region in South America can get confused with a company called Amazon. The user can usually help correct such classification errors by supplying synonym tables or custom entity rules. However, this has not been done in this example.

Representing Data

One of the major challenges with unstructured text data is that most existing statistical methods (such as random forest and logistic regression) cannot be applied directly. PROC TEXTMINE overcomes this problem by using a dimensionality reduction technique, Singular Value Decomposition (SVD) to convert unstructured text data to structured data with a specified number of dimensions.

In our previous PROC TEXTMINE call in Program 3.5, dimensionality was specified to be 10. So in the new structured representation of the documents, each document is represented by 10 values.

Once unstructured text data is converted to structured data representation, many existing statistical methods can be applied to perform other tasks, such as prediction and forecasting. The first PROC PRINT statement in Program 3.7 shows the representation of five documents from the data set created by Program 3.5. This is displayed as the first table in Output 3.7. Based on the SVD results, PROC TEXTMINE is also able to derive topics and assign a list of descriptive terms to each topic. The second table in Output 3.7 shows the 10 topics found by PROC TEXTMINE.

Program 3.7: PROC PRINT of Program 3.5 Output

```
proc print data=mycaslib.docpro(obs=5);
run;

proc print data=mycaslib.topics;
run;
```

Output 3.7: Results of Program 3.7

Obs	id	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	r1	0	0	0	0	0.5427891152	0.0797711786	0.0789485873	0	0.6313566053	0.5423660669
2	r2	0.1208676268	0.1519448046	0.149266436	0.173609307	0.2462166699	0.0907123805	0.2801134592	0.049466743	0.7719512305	0.4054403897
3	r3	0.1412611205	0.1272092851	0.0325636422	0.0529614519	0.1032340391	0.1077224756	0	0.071972989	0.8014022542	0.5387998004
4	r4	0	0.067979027	0	0.0415725437	0.1850327312	0.1840330054	0.1114092531	0.2619559125	0.6266310325	0.6722841801
5	r5	0.2469380536	0	0.0748666253	0.0485017988	0.2513739771	0.1048940675	0.1659100357	0.0382914667	0.6511466858	0.6365466449

Obs	_topicid	_termCutOff	_name
1	1	0.017	+guitar, +hero, tennis game, industry, +game industry
2	2	0.018	+log, x360, de, +profile, facebook
3	3	0.017	proxy, +round, +core gamer, yes, $600
4	4	0.018	x360, +videogame, +strap, wiiconnect24, loading
5	5	0.017	+just sit, online, star, +exercise, +reserve
6	6	0.017	x box, +gym, +note, wii-mote, +implement
7	7	0.017	+lead, wiimote, battlefield, +game demo, nes
8	8	0.018	wii-mote, lighting, +channel, +newcomer, +edit
9	9	0.02	+game, +system, +drive, +play, ps3
10	10	0.02	wii, wii, +game, nintendo, +play

One of the interesting topics in the first row of the second table in Output 3.7 is about Guitar Hero and the gaming industry. If some of the topics seem confusing or ambiguous, you can often peruse individual documents to learn how a topic might have been derived. For example, in the second row of the second table, you can read that “facebook” is a feature for Xbox 360.

PROC TMSCORE

PROC TEXTMINE is used with large training data sets. When you have new documents coming in, you do not need to re-run all the parsing and SVD computations with PROC TEXTMINE. Instead, you can use PROC TMSCORE to score new text data. The scoring procedure parses the new document(s) and projects the text data into the same dimensions using the SVD weights derived from the original training data.

In order to use PROC TMSCORE to generate results consistent with PROC TEXTMINE, you need to provide the following tables generated by PROC TEXTMINE:

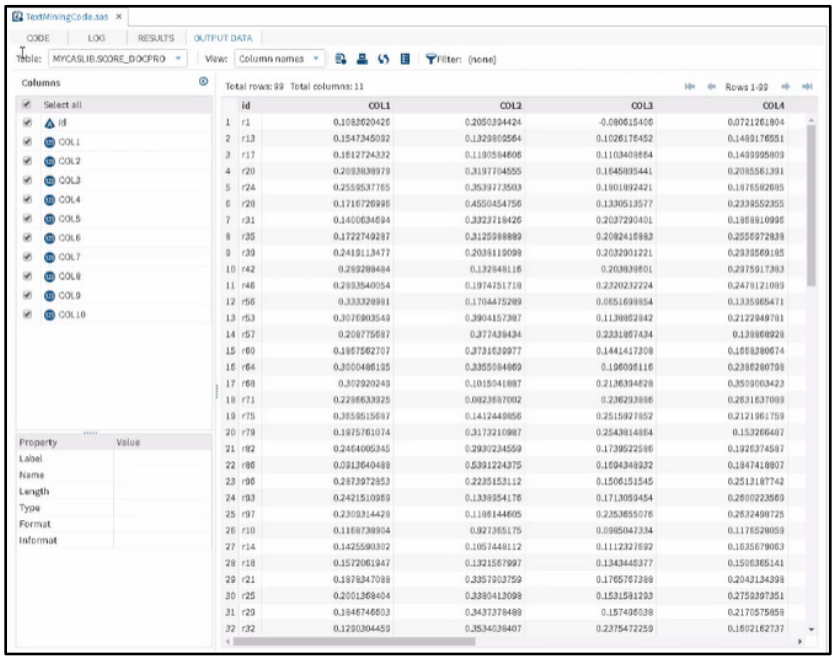
- SVDU table – provides the required information for projection into the same dimensions.
- Config table – provides parameter values for parsing.
- Terms table – provides the terms that should be included in the analysis.

Program 3.8 shows an example of TMSCORE. It uses the same input data layout used for the PROC TEXTMINE code in Program 3.5, so it will generate the same docpro and parent output tables, as shown in Output 3.8.

Program 3.8: PROC TMSCORE

```
Proc tmscore data=mycaslib.amazon svdu=mycaslib.svdu
        config=mycaslib.config terms=mycaslib.terms
        svddocpro=mycaslib.score_docpro outparent=mycaslib.score_parent;
    var reviewbody;
    doc_id id;
run;
```

Output 3.8: Results from Program 3.8



Community Detection

In this section, we will look at how PROC NETWORK, a component of SAS Visual Data Mining and Machine Learning, can help identify communities that exist within network structured data.

Introduction to Networks

Networks consist of nodes connected in pairs by links, and links can indicate directed (A → B) or undirected (A — B) connections between nodes. You can think of directed links as permitting one-way connections and undirected links, two-way. To represent two-way connections with directed links, you use pairs of links, for example, from A to B (A → B) and from B to A (B → A). Networks contain either directed or undirected links, not a mix.

Each link typically has a weight assigned to it. The weight designates the significance or magnitude of the flow across the link. We can describe relationships between individual people, companies, or organizations. The links could represent phone calls, purchases, legal transactions, and so on. We could be talking about links between books, research papers, or social media accounts that refer to each other in some way, perhaps reference citations, or retweets.

PROC NETWORK contains 11 different types of network analysis algorithms that enable you to explore various aspects of network structure, how nodes group together into cliques or communities, the relative importance or centrality of individual nodes, how many nodes can be reached from any given starting node, and so on. Community detection is one of the algorithms available in PROC NETWORK.

Community detection is used with undirected networks. If you want to analyze a directed network, then you will need to convert directed links into undirected links first. There are many ways to process the link weights of "A to B and B to A" pairs of directed links during this process. For example, averaging the weights or adding them together. In community detection, the end result is that each node is assigned to exactly one community.

The communities are detected by examining patterns among the links. Nodes within the same community are more densely connected than nodes in different communities. The weights on the links factor into these calculations.

Modularity is the metric used to assess how effectively a set of communities divides a network. It measures how well community detection reflects the varying density of the weighted links in the network. The modularity is the weighted fraction of all links that are within any defined community minus the weighted fraction of links that would be in the same community if the links were randomly distributed among the nodes.

PROC NETWORK

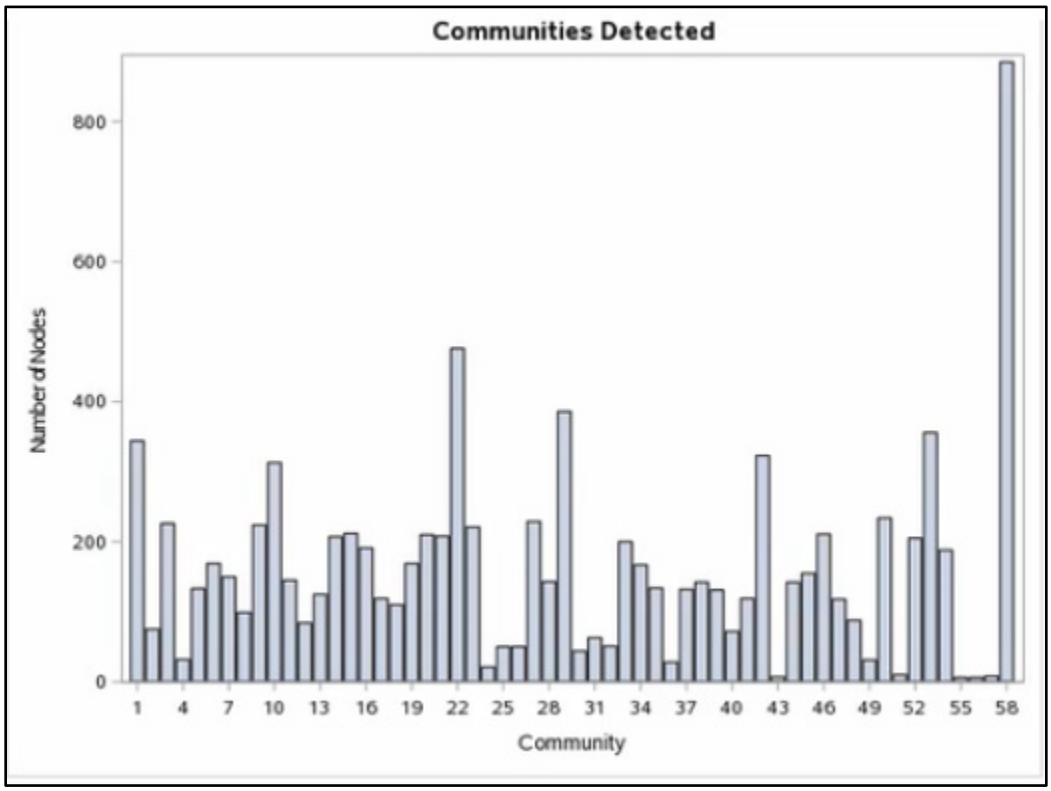
Let's see what communities we can detect in a data set on collaboration in physics research. The ARXIV example data set is available from Stanford University at this link: https://snap.stanford.edu/data/ca-AstroPh.html. In this example, each node in the network is a researcher, and a link represents a collaboration between two researchers. There are almost 9,400 nodes and just over 24,000 links in this undirected network.

Program 3.9: PROC NETWORK

```
cas sascas1;

libname mycas cas sessref='sascas1';
libname local "/u/sasuser/community/";

data mycas.arxiv(partition=(from));
    set local.arxiv(rename=(from_node=from to_node=to));
run;

proc network links=mycas.arxiv outnodes=mycas.arxivOutNodes
        outlinks=mycas.arxivOutLinks;
    community;  ❶
run;

title 'Communities Detected';
proc sgplot data=mycas.arxivOutNodes;
    vbar community_1;
    xaxis label='Community' fitpolicy=thin;
    yaxis label='Number of Nodes';
run;
```

❶ If we run PROC NETWORK with the default Louvain algorithm, according to the Log, it detects 58 communities with modularity just over 0.81, which is very good.

Output 3.9: Results of Program 3.9



The bar chart of the community sizes in Output 3.9 shows that one community has well over 800 nodes. Another has over 400, and the rest are below 400. We would like to even out the community sizes if we can.

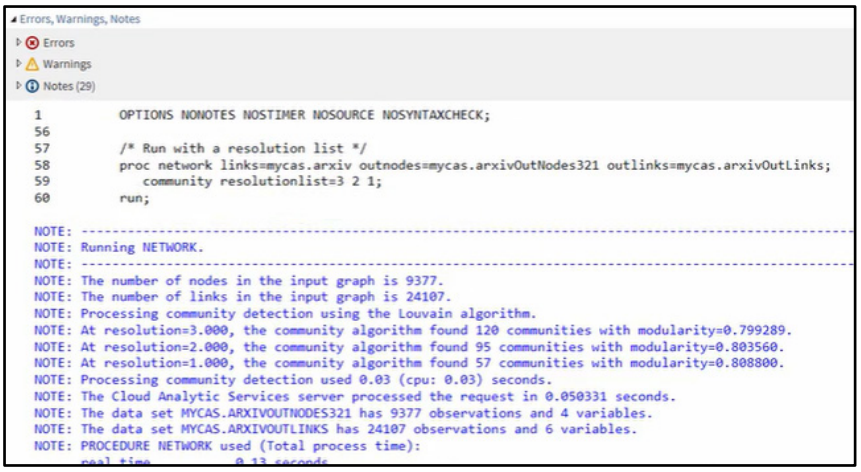
PROC NETWORK provides two ways to reduce unusually large communities. The first option runs the community detection algorithm recursively, processing the largest communities repeatedly. The second option is supplying a list of resolution values. In this example, the second option works best, as shown in Program 3.10, which modifies the code from Program 3.9.

Program 3.10: Modified PROC NETWORK Code

```
proc network links=mycas.arxiv outnodes=mycas.arxivOutNodes321
        outlinks=mycas.arxivOutLinks;
    community resolutionlist= 3 2 1;  ❶
run;

title 'Communities Detected with Resolution=1';
proc sgplot data=mycas.arxivOutNodes321;
    vbar community_3;
    xaxis label='Community' fitpolicy=thin;
    yaxis label='Number of Nodes';
run;

title 'Communities Detected with Resolution=2';
proc sgplot data=mycas.arxivOutNodes321;
    vbar community_2;
    xaxis label='Community' fitpolicy=thin;
    yaxis label='Number of Nodes';
run;

title 'Communities Detected with Resolution=3';
proc sgplot data=mycas.arxivOutNodes321;
    vbar community_1;
    xaxis label='Community' fitpolicy=thin;
    yaxis label='Number of Nodes';
run;
```

❶ The resolution level, which is 1 by default, influences the likelihood that two communities will be merged to form a larger community. Both of PROC NETWORK's algorithms form communities from the bottom up by starting with communities of 1 node each and then iteratively merging nearby communities. A higher resolution value makes large communities less likely and smaller communities more likely. So in addition to the default resolution of 1, we will also run PROC NETWORK with resolution values of 2 and 3. PROC NETWORK runs with the highest resolution first and then speeds up its analysis by using those results as a starting point for the run with the next lowest resolution.

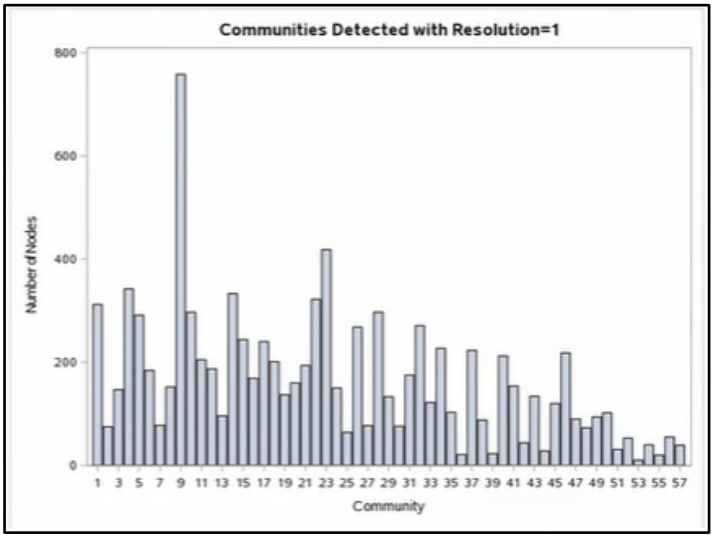
Log 3.10: Log for Program 3.10

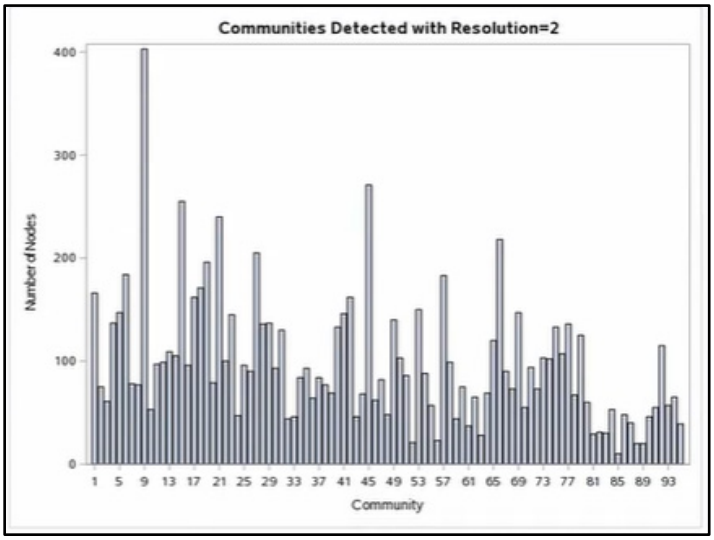


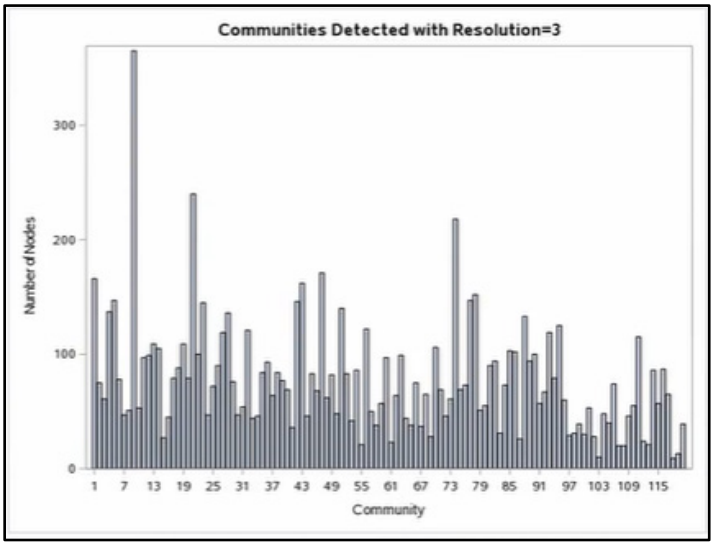
According to the Log, with resolution 1, we get 57 communities. That is slightly different from the previous run because PROC NETWORK ran first with resolution 3, then 2, and then 1, instead of starting with 1. For resolution 2, we get 95 communities and a slightly lower modularity. With resolution 3, there are 120 communities, and there was a further slight drop in modularity.

Let's look at community size in Output 3.10.

Output 3.10: Results of Program 3.10







For resolution 1, much the same as before, 2 communities are dominant, with the largest near 800 in size. For resolution 2, we see more uniformity, and a peak just above 400. For resolution 3, the peak drops below 400, and there is even more uniformity among community sizes. This warrants further investigation. But with only a slight drop in modularity so far, things are moving in a good direction.

PROC NETWORK provides many more forms of network analysis, all of which complement the other capabilities in SAS Visual Data Mining and Machine Learning.

Resources

This chapter is based on the "Data Mining and Machine Learning on SAS Viya" videos in SAS® Viya® Enablement, a free course available from SAS Education.

You might find the following documentation and resources helpful as you learn more about programming in SAS Visual Data Mining and Machine Learning:

- SAS® Viya® 3.2 Programming: Data Mining and Machine Learning Procedures
- Data Mining and Machine Learning Procedures: The FACTMAC Procedure
- Data Mining and Machine Learning Procedures: The TEXTMINE Procedure
- Data Mining and Machine Learning Procedures: The TMSCORE Procedure
- SAS® Viya® 3.2 Programming: Data Mining and Machine Learning: The NETWORK Procedure
- Link to download MovieLens data set: http://grouplens.org/datasets/movielens
- Link to download Amazon review data set: https://snap.stanford.edu/data/web-Amazon.html
- Link to download ARXIV data set: https://snap.stanford.edu/data/ca-AstroPh.html

Chapter 4: SAS Visual Data Mining and Machine Learning in Model Studio

Introduction

Model Studio is a central, web-based platform that includes a suite of integrated data mining tools. The data mining tools that appear in Model Studio are determined by your site's licensing agreement. Model Studio can be used with SAS Visual Data Mining and Machine Learning, SAS Visual Forecasting, or SAS Visual Text Analytics.

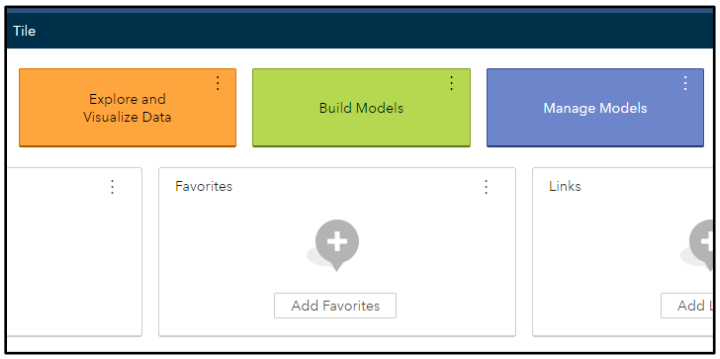
Pipelines

SAS Visual Data Mining and Machine Learning facilitates a process-workflow approach to analytics. This approach includes pipelines. Pipelines are structured flows of analytic actions. Pipelines offer repeatability and can serve as best-practice templates for building large-scale projects. This functionality is provided in the Model Studio application as part of SAS Viya.

Accessing Model Studio

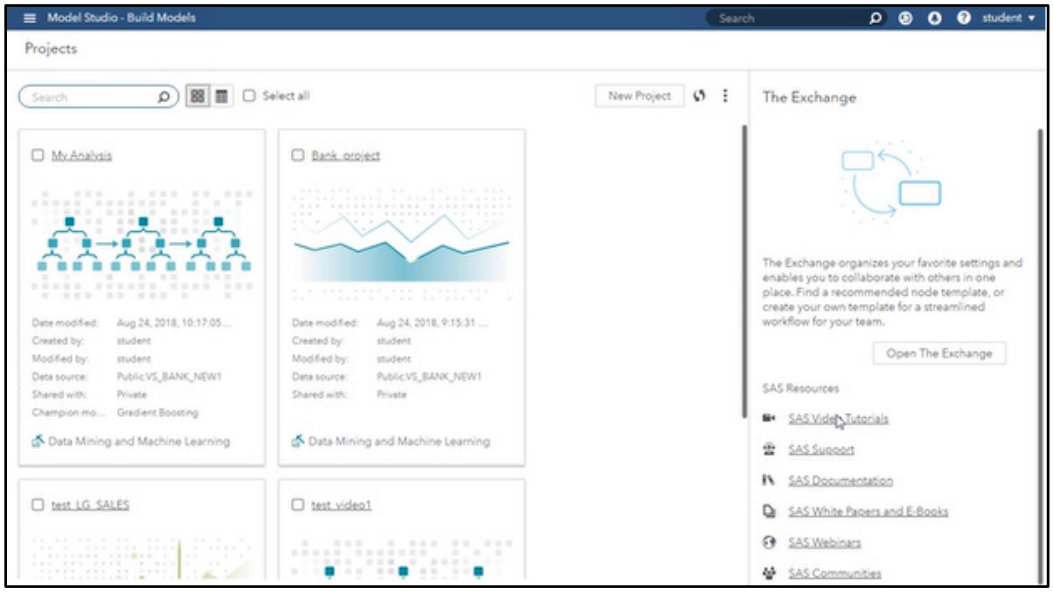
Model Studio is accessed by selecting **Build Models** from the SAS home page, as shown in Figure 4.1.

Figure 4.1: Build Models Tile



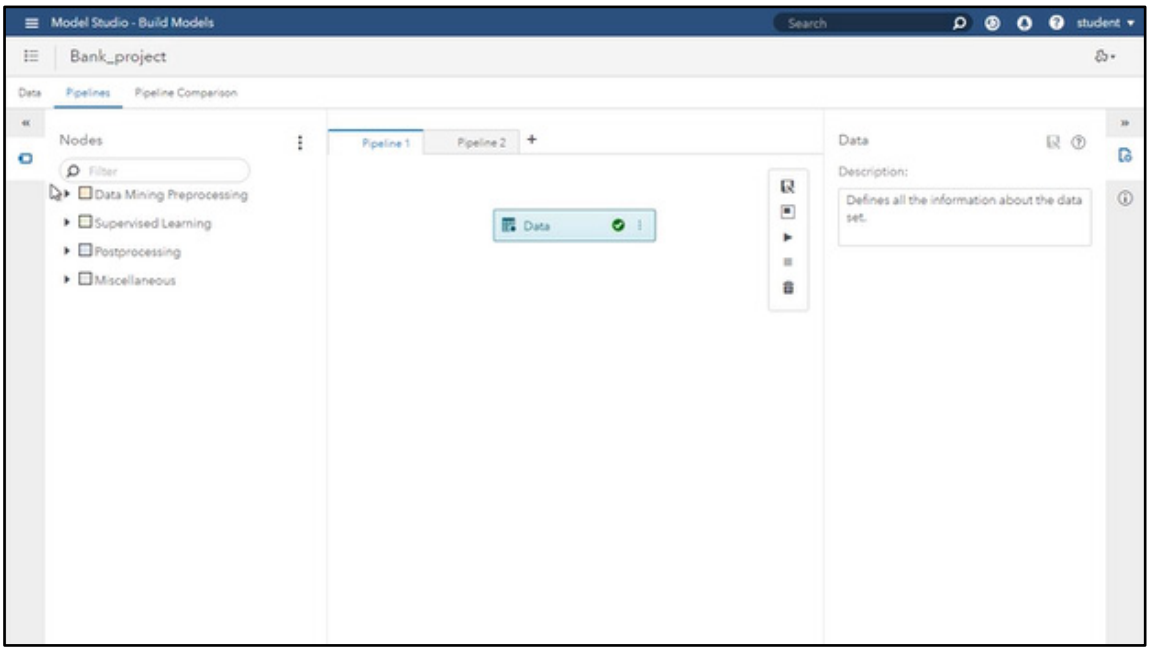
You will then be taken to the Model Studio home page (see Figure 4.2) where you can open an existing project or start a new project.

Figure 4.2: Model Studio Home Page



Once you have started or opened a project and selected your data, new pipelines can be created from scratch or by starting with an existing template. A number of templates are provided with the product or you can create your own. The default Nodes menu shown in the left pane of Figure 4.3 includes preprocessing tools for data exploration, feature extraction, transformation, and selection. Supervised learning tools include a selection of algorithms for building classification and regression models. Post-modeling capabilities, such as ensemble modeling, are available, as well as other nodes for exploring your data, saving results, or inserting custom code into your pipeline.

Figure 4.3: Nodes Menu



New nodes can be added to a pipeline by dragging and dropping them into the Pipeline window to create a custom pipeline.

Champion models from all pipelines that you have run are listed on the Pipeline Comparison tab. You can select and compare any models that have been created. Then, you can proceed to the next steps in the analytics life cycle by registering the champion model to SAS Model Manager or publishing it to a database.

All nodes and pipeline templates can be viewed and managed via the Exchange, which is on the right side of the home screen when you open Model Studio. (See Figure 4.2.)

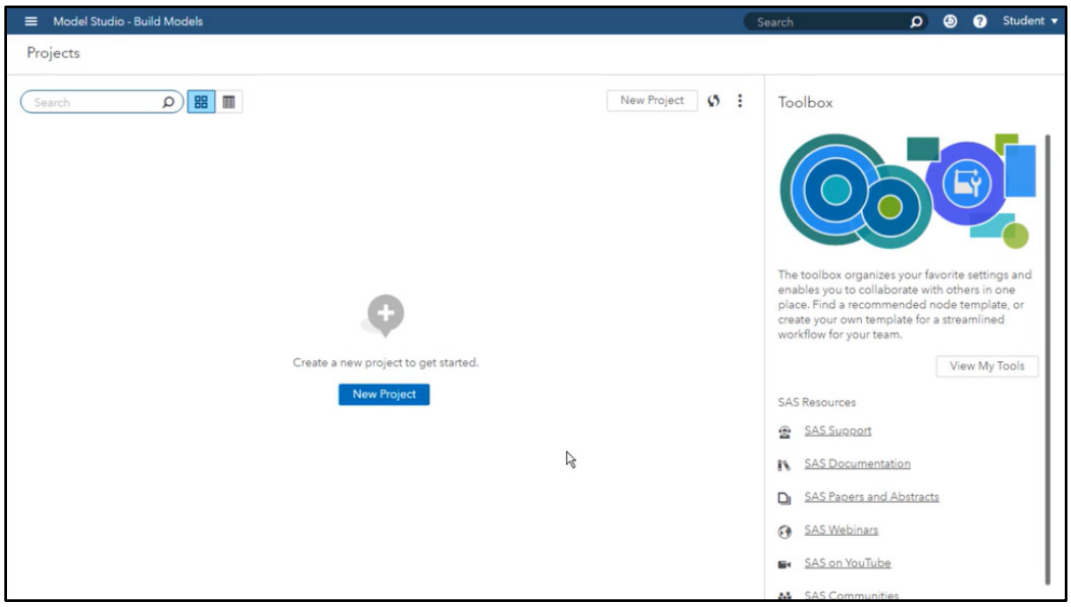
Now that you are familiar with the basics of Model Studio, let's learn how to perform specific actions.

Impute Missing Values

In this section, we will learn how to impute missing values in Model Studio. We will use the banking data set VS_BankData. This data set identifies customers that have purchased a banking product. The target will be whether the customer purchased a particular project. The data consists of 1.1 million rows and 23 columns. It is available to download from GitHub from https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/bank.

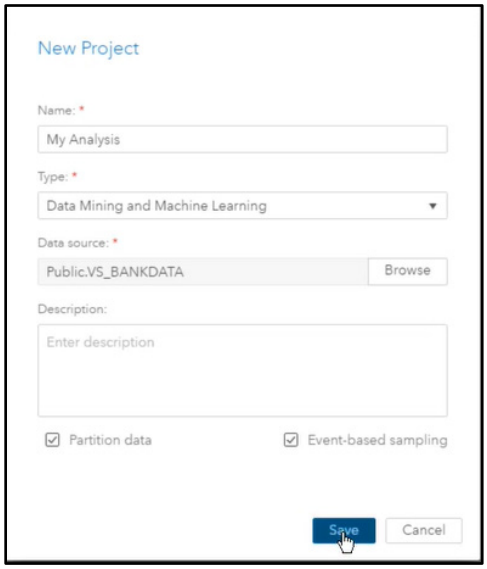
We will start on the Model Studio home page. The first step is to create a new project. Click the **New Project** button in the center of the screen, as shown in Figure 4.4.

Figure 4.4: Model Studio Home Page



The New Project window appears, as shown in Figure 4.5. Let's call this project My Analysis. Assign the Type as Data Mining and Machine Learning. For the Data Source, click **Browse** and navigate to VS_BankData (you should have already loaded your data source into CAS). We are going to build a predictive model; therefore, we will keep the default settings checked for Partition data and Event-based sampling. Click **Save** to create the project.

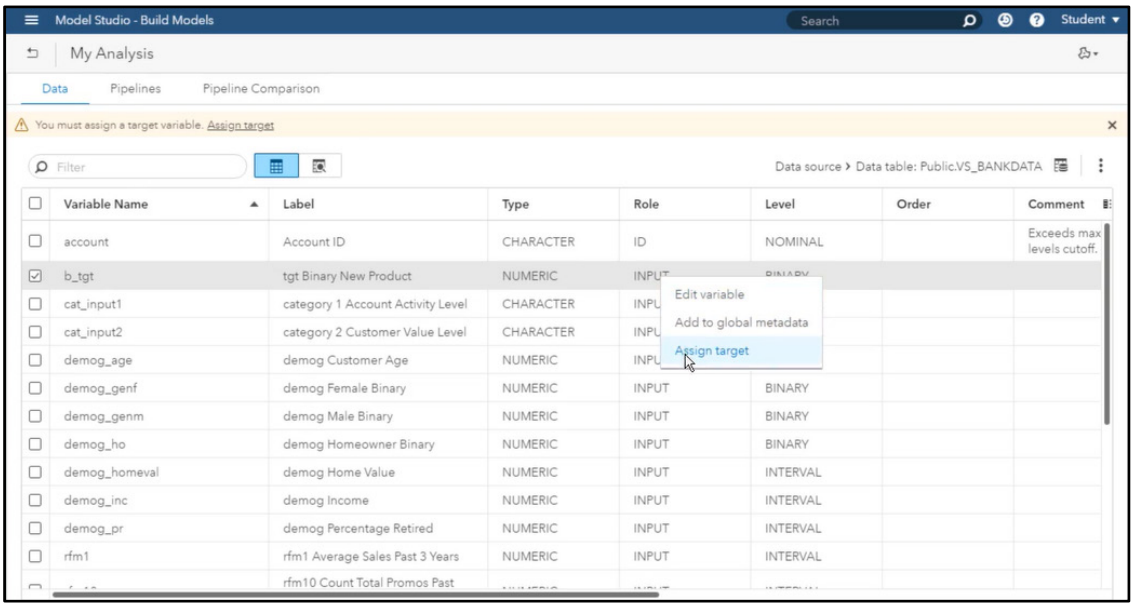
Figure 4.5: New Project Window



Assign a Target Variable

Model Studio populates the variables table from the specified data source, as shown in Figure 4.6. Here, you must assign a target variable. Our target variable is b_tgt. Under the Role column, right-click the value for the b_tgt variable and select **Assign target** from the drop-down menu.

Figure 4.6: Data Tab



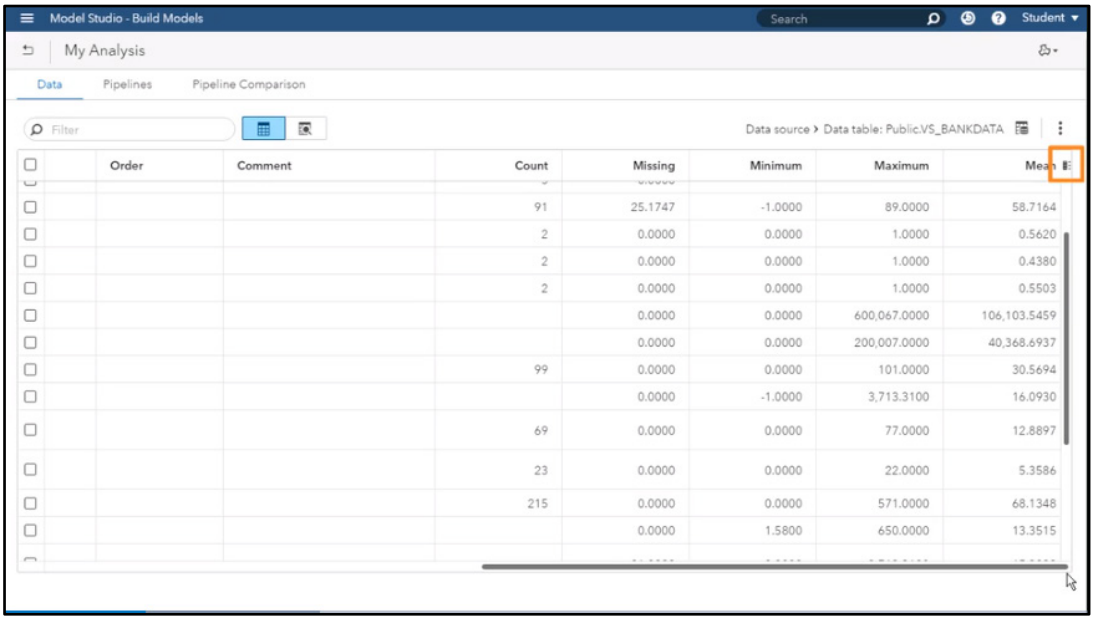
The **Assign Target window** will appear. Select b_tgt from the variable name list, then click **Save**. In the Role column of b_tgt, it should now say TARGET.

Manage Variables

The Data tab shows all the variables that exist in the data set used by this project, along with metadata for each variable. The metadata includes the variable type, level, and role, which control how the variable should be used in your project. We can also see descriptive statistics and information about missing values and the number of unique values.

There are many columns that are not shown by default. You can configure which columns to show using the Options button in the upper right of the Data table, as shown outlined in a box in Figure 4.7.

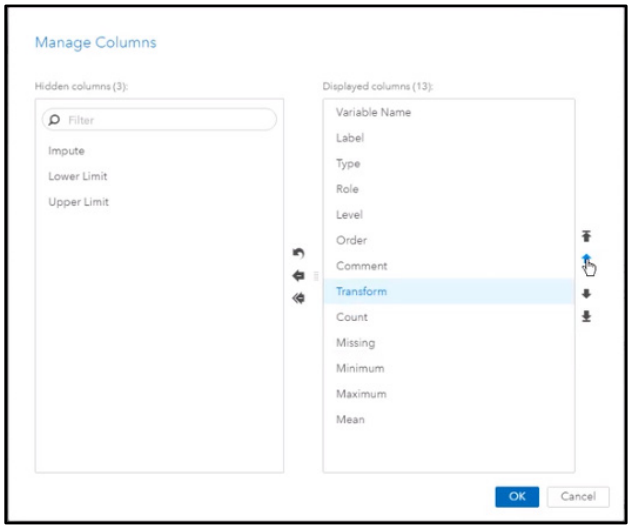
Figure 4.7: Data Tab – Options Button



Click the **Options** button, and then select **Manage columns** from the menu. The **Manage Columns window** will appear, showing the hidden columns and the displayed columns. You can use this window to configure the columns in the Data tab.

For example, if we want to show the Transform column, click **Transform** and then add Transform by clicking on the single arrow in the center of the window. Then, you can move it up so that it appears just after Comment, by using the up and down arrows, as show in Figure 4.8.

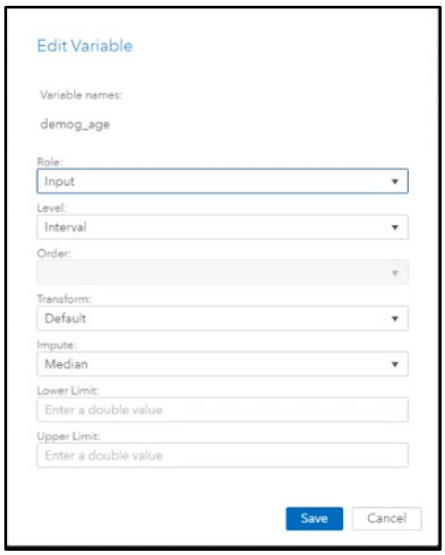
Figure 4.8: Manage Columns Window



Let's hide the columns Label, Order, and Comment. Select these columns and then click the single arrow in the center to move them into the hidden columns section. Click **OK** to exit the window.

In the Data tab, you can also select variables and edit the metadata associated with them to control how they should be used in the pipelines in this project. For example, you might decide that the variable DEMOG_AGE should not be used as a model input, so you can set its role to Rejected. Right-click on the variable name, then select **Edit Variable** from the menu. The **Edit Variable** window appears, as shown in Figure 4.9.

Figure 4.9: Edit Variable Window



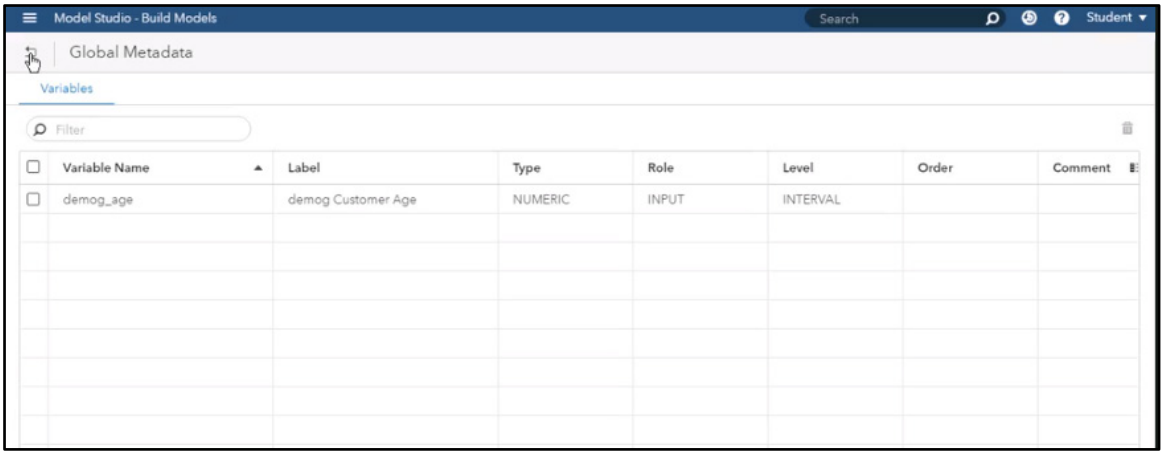
In this window, you can change the role if you would like. In addition, you might know that certain variable needs a transformation applied to it, whether it's to standardize the values, adjust the distribution, or to bin values. You can also define a specific method for imputing missing variables for this individual variable, which

will be discussed in the next section. You can enter lower and upper limits to use for filtering or replacing values. Click Cancel to exit the window.

Note that none of these settings are applied to the data immediately. Rather, we are simply defining metadata, or information about the variables, to be used by nodes in your pipeline. This metadata is used by the nodes to perform preprocessing actions on the data. By defining metadata on the Data tab, you avoid the need to define it within all of your pipelines.

If you want the metadata for a certain variable to be applied to variables with this same name and type in future projects, you can add this variable to a global metadata registry. Right-click on the variable name and select Add to global metadata from the menu. Global metadata can be managed in the toolbox, which is accessed from the Model Studio project home screen. Click the return button in the upper left corner of the screen to return to the projects home page. In the Toolbox panel on the right, click the **View My Tools** button. This opens the **Toolbox** window. In the upper right, click the **Global Metadata** button. Notice that the metadata for demog_age has been added to the global metadata list, as shown in Figure 4.10.

Figure 4.10: Global Metadata List



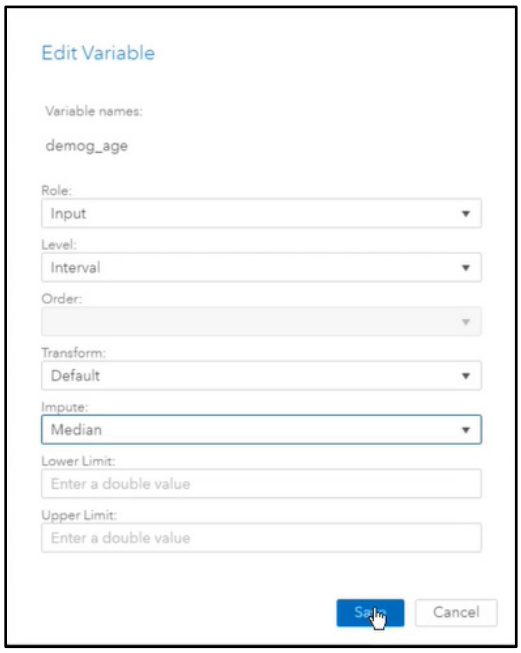
Click the return button and open the My Analysis project again to view the Data tab.

Impute Missing Values Method

The variables table provides general summary statistics. One statistic is the percentage of missing values for each variable. Scroll all the way to the right to see the Missing column. Notice that approximately 25% of the values for Customer Age (demog_age) and approximately 21% of the values for average sales (rfm3) in the past three years are missing.

To impute the missing values for Customer Age, right-click the row and select **Edit variable** from the drop-down menu. In the Edit Variable window, under **Impute**, select Median, as shown in Figure 4.11. Click **Save** to save the changes.

Figure 4.11: Edit Variable Window



Setting the imputation method at this stage in the process does not implement the imputation. Instead, it specifies the method to be used when the Imputation node is run. The purpose of setting the imputation method at this stage in the process is to allow the use of different imputations for different input variables.

To impute the missing values for average sales in the past three years (rfm3), right-click the row and select **Edit variable** from the drop-down menu. Under the **Impute** option in the Edit Variable window, this time select Mean as the imputation method. Notice that in addition to different imputations, you could also specify different transformation methods for each variable under the Transformation option. Click **Save** to save the changes in the window.

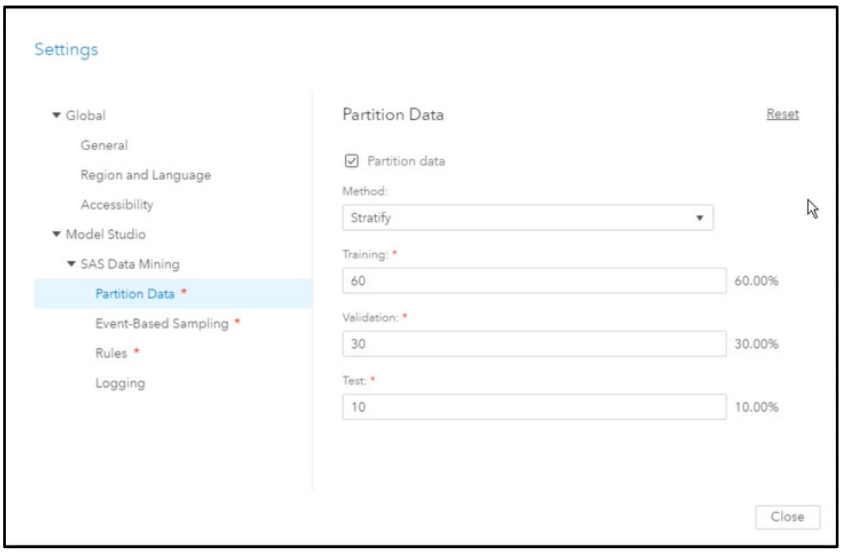
Let's also change the level of two count variables that were designated as Nominal to the Interval measurement level. Right-click the Level column for the rfm5 variable. Select **Edit variable** from the drop-down menu. In the Edit Variable window, change the **Level** option from Nominal to Interval. Click **Save**. Repeat this process for the rfm7 variable.

Set Sampling Criteria

When we observe the Mean column for the target variable (b_tgt), notice that the event rate is approximately 20%. Given that only 20% of the observations experienced the event, we are going to take a balanced sample of the target for data efficiency purposes.

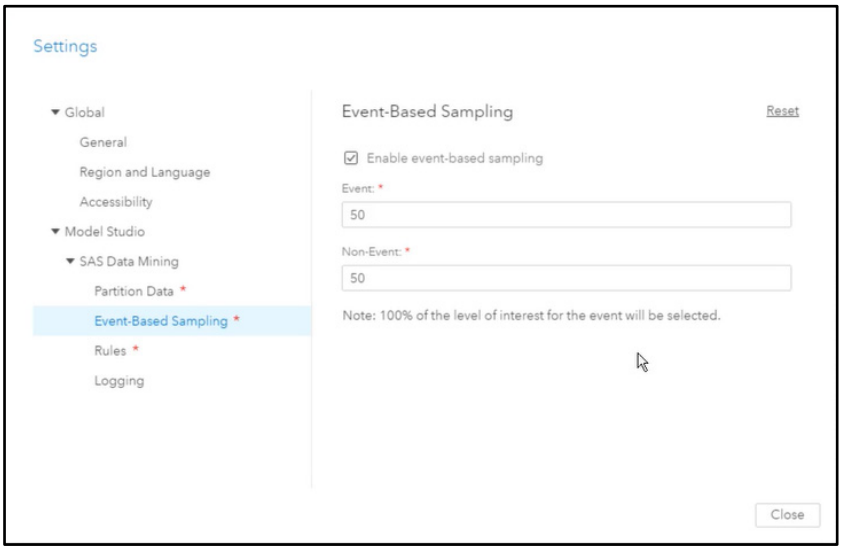
To set the sampling criteria, select the project Settings. Click on your user name in the upper right corner and click **Settings** from the drop-down menu. In the Settings window, we want to view the partition allocations. To do that, select **Partition Data**, as shown in Figure 4.12. The window shows that we are going to allocate 60% of our data to the Training partition, 30% to Validation, and 10% to Test.

Figure 4.12: Settings Window – Partition Data



To set the event-based sampling, select the **Event-Based Sampling** option, as shown in Figure 4.13. We are going to keep the defaults of 50% Events and 50% Non-Events. Note that 100% of the events will be selected. Close the Settings window.

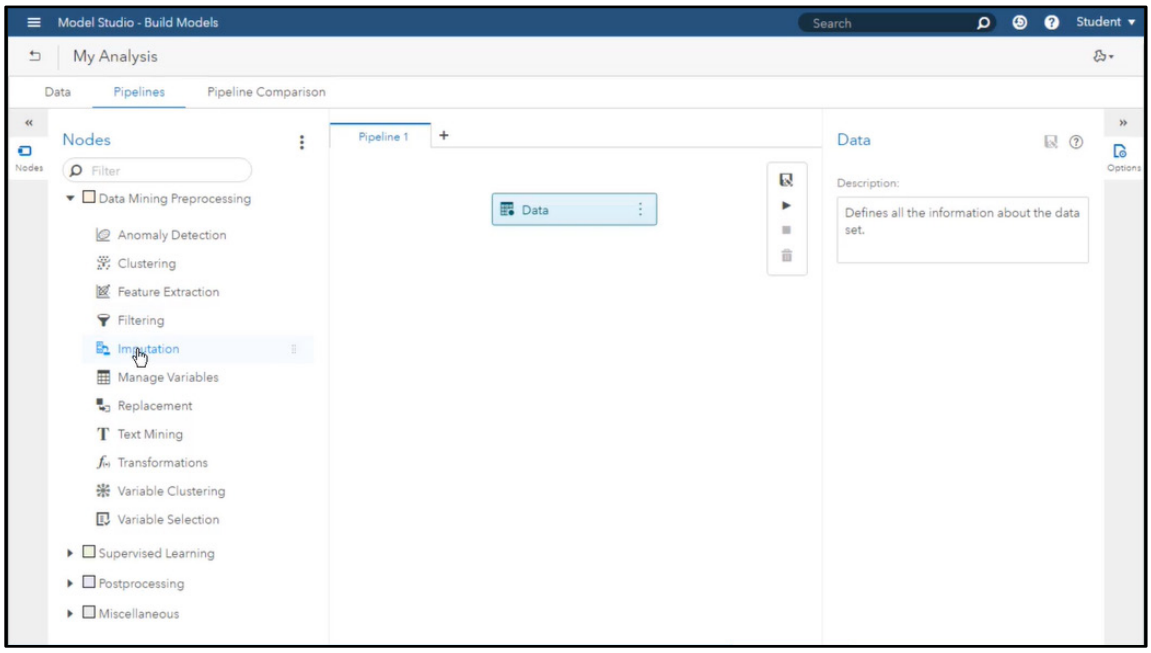
Figure 4.13: Settings Window – Event-Based Sampling



Build Pipeline

Now we are ready to begin building our pipeline. To get to the pipeline, click the **Pipelines** tab near the top of the screen. Notice that the data are already loaded in the pipeline, as shown in Figure 4.14.

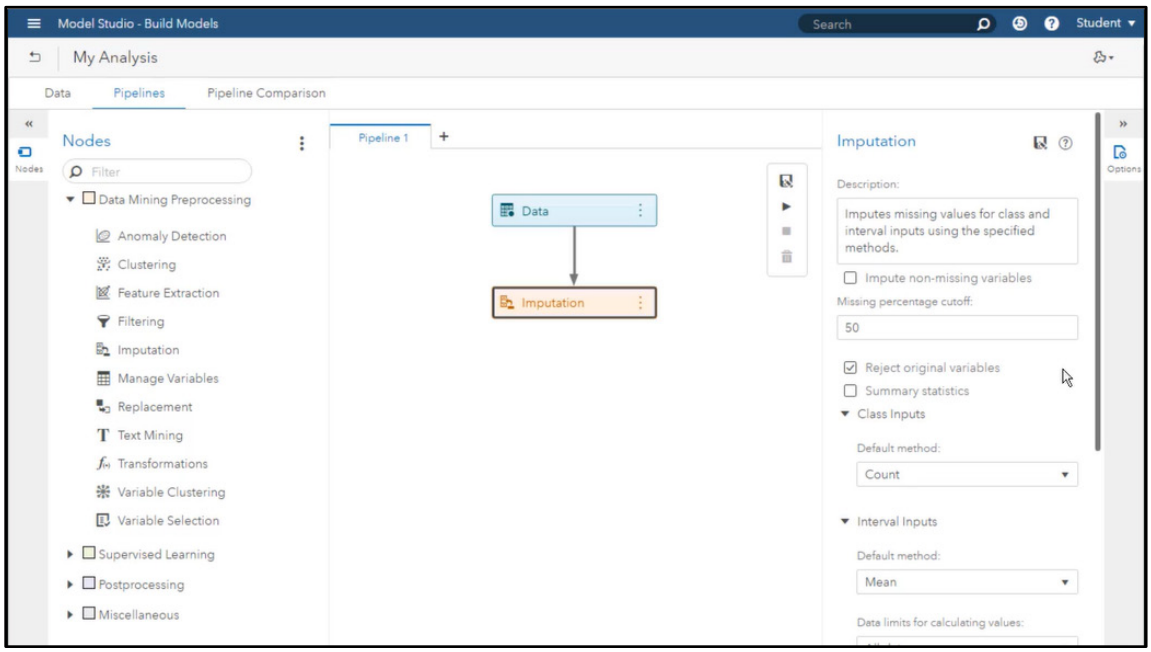
Figure 4.14: Pipeline Tab



Select **Nodes** on the farthest left panel. We will select Nodes to customize the features of the pipeline. Given that we know the data have missing values, we will select the Imputation node under the Data Mining Preprocessing group.

There are several ways to add a node to the pipeline. In this instance, we will select the **Imputation** node and drag it on top of the data source in the Pipeline window. Notice that the Imputation node options appear to the right, as shown in Figure 4.15.

Figure 4.15: Imputation Node



We will leave the **Missing percentage cutoff** at 50%. The box for **Reject original variables** is selected by default. We will select the box for the **Summary statistics** option, which generates summary statistics. There are imputation methods for both class and interval variables. Given that we already selected the imputation methods in the variables table, we will select None for both class and interval variables under default method.

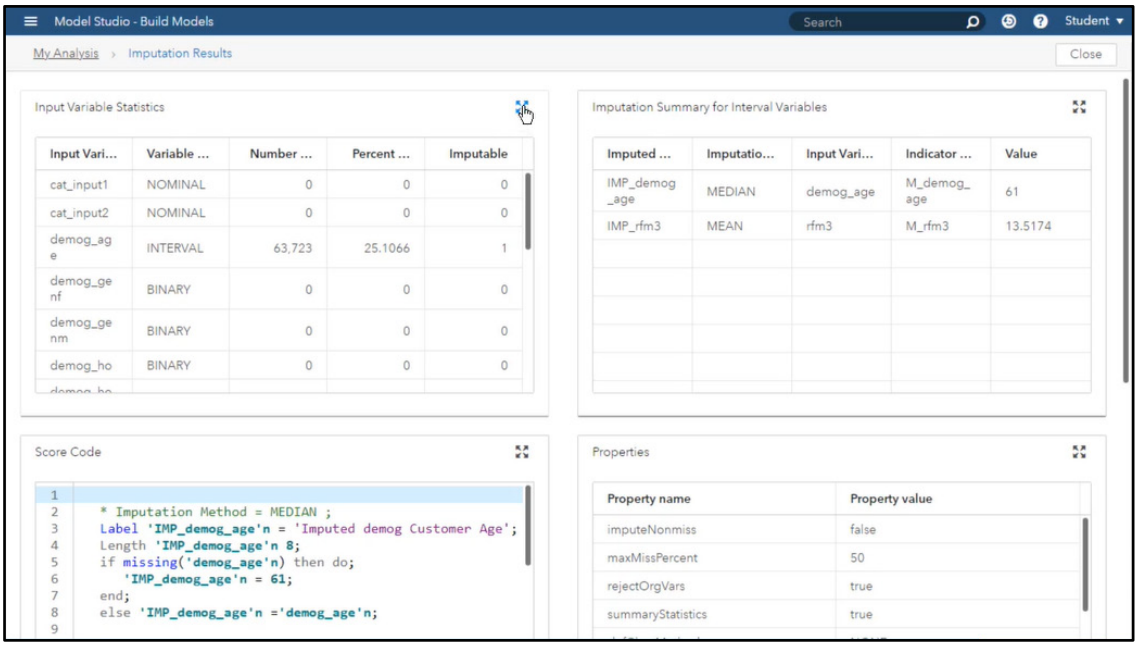
If we had not selected the imputation method in the variables table, then we would select them in the Imputation node, which allows only one imputation method for the class variables and only one for the interval variables. However, the Imputation node does enable you to create Single indicator and Unique indicator variables to model input missingness as it relates to the target. This functionality is not available in the variables table. The **Single indicator** option creates a variable that counts the number of inputs that were imputed. The **Unique indicator** option creates a separate variable for each input being imputed. Select both check boxes, then set the **Indicator subject** property to Missing variables and the **Indicator role** property to Input.

Now we are ready to run the analysis. Right-click the **Imputation** node in the pipeline and select **Run** from the drop-down menu. When the analysis is finished, right-click the Imputation node again and select **Results** from the drop-down menu.

Results

The Results load in a new window, as shown in Figure 4.16. Click on the expand button in the upper right corner of any table to view it in the full window.

Figure 4.16: Imputation Results



The Input Variable Statistics table shows the number of missing values and the percent missing for the variables. The demog_age and rfm3 variables were the only variables that were imputed. They are imputable because the percent missing was less than 50%.

The Imputation Summary for Interval Variables table shows the new imputed variables, the imputed method used on each variable, the indicator variables that were created, and the value the missing value was replaced with. The Score Code window shows the score code that was generated to impute the missing values. The Properties table shows information such as the random seed value and whether the missing indicator variables were created. Finally, scroll down to see the Output table, which shows descriptive statistics of the input variables.

Feature Engineering

In this section, we will learn how to use Model Studio to transform your variables and create new features. To begin, on the Home Page, open the My Analysis project that we created in the previous section. Data has already been uploaded into the project.

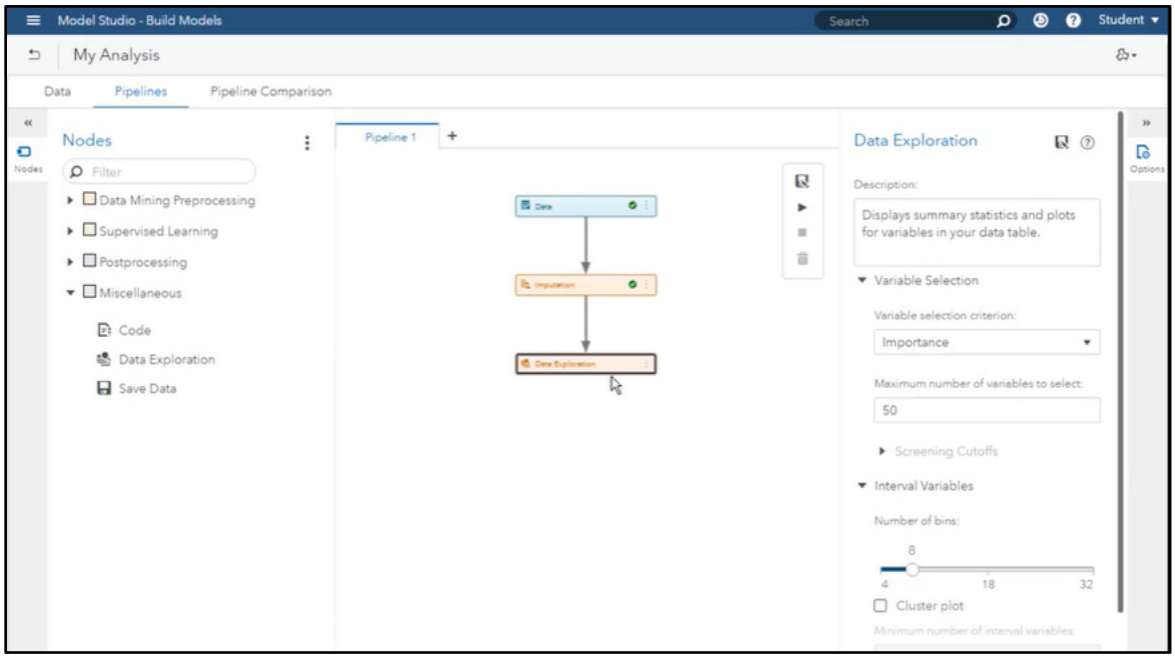
Select the **Pipelines** tab to go to the pipeline. We can see that data has been uploaded and that imputations have already been applied. Select **Nodes** from the far left panel and then select the **Data Exploration** node under the **Miscellaneous** group.

Data Exploration

Build Pipeline

There are several ways to add a node to the pipeline. In this instance, select the **Data Exploration** node and drag it on to the Imputation node. Notice that the Data Exploration options appear on the right, as shown in Figure 4.17.

Figure 4.17: Data Exploration Node

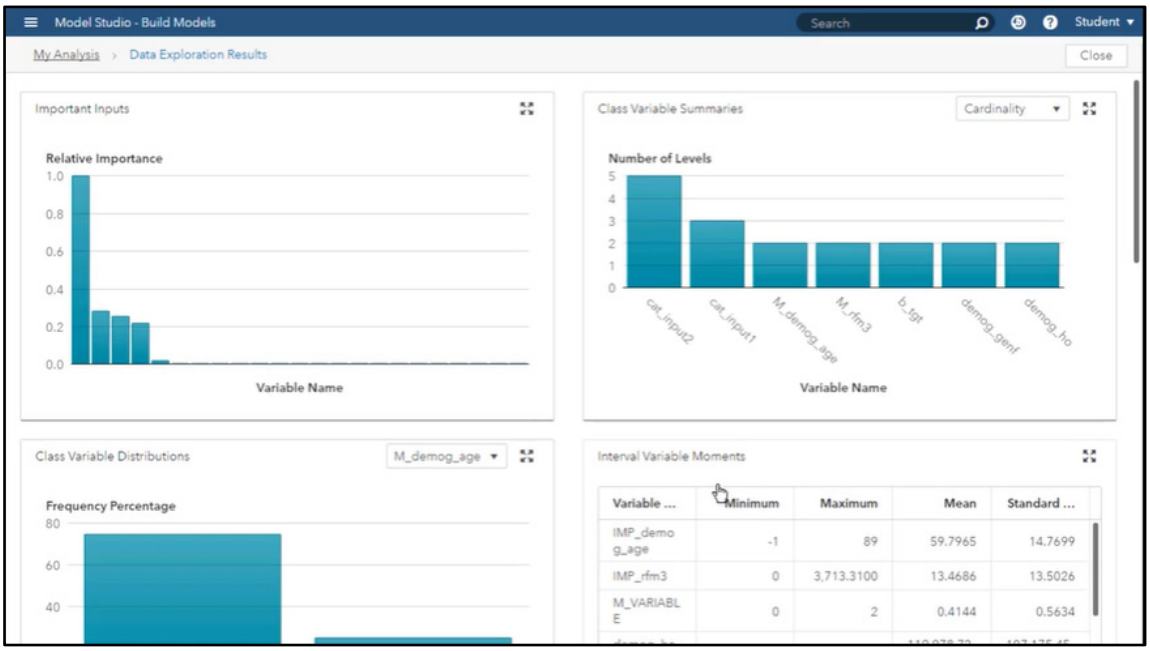


To run the node, right-click the **Data Exploration** node in the pipeline and then select **Run** from the drop-down menu. When the node is finished running, right-click again and select Results from the drop-down menu.

Results

The **Results** window is populated with graphs that describe variable distributions and tables that provide summary statistics, as shown in Figure 4.18.

Figure 4.18: Data Exploration Results



The Relative Importance bar chart provides an indication of important predictors. When you hover your cursor over the tallest bar, you can see that RMF5 is the most important predictor.

The Class Variables Summaries chart provides the cardinality of categorical variables by default. It appears that CAT_INPUT2 has the largest number of distinct class values. We can also use the Class Variables Summaries chart to explore mode percentages. In the drop-down menu in the upper right corner of the chart, select Mode Percent to view a bar chart of the mode percentages.

Scroll down to the Interval Variable Summaries graph to observe the deviation from normality. The variables RFM1, IMP_RFM3, and RFM4 appear to have a high kurtosis and skewness.

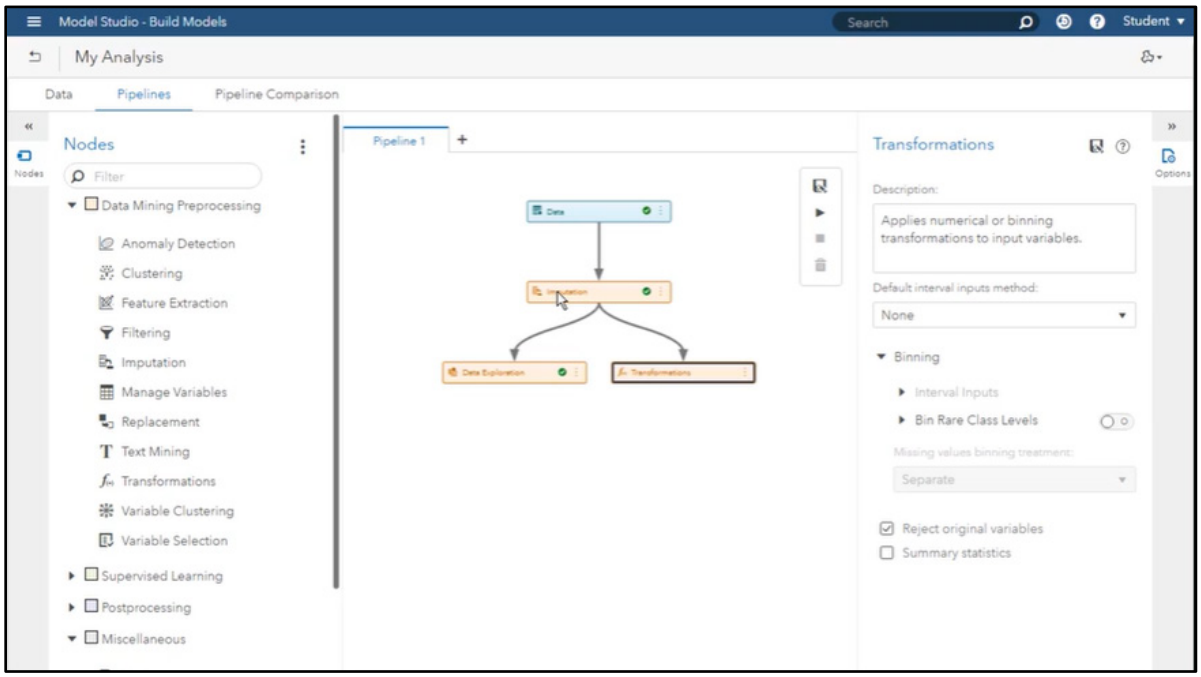
Transformation

Now we are going to apply a log transformation to our input variables. Return to the Pipeline tab.

Build Pipeline

Select the **Transformations** node under the **Data Mining Preprocessing** group. Click and drag the Transformations node on to the Imputation node to create a link between the two nodes, as shown in Figure 4.19.

Figure 4.19: Transformation Node



If we wanted to apply a different transformation to each variable, we would have to assign the transformations in the variables table. You can do this by editing a variable on the Data tab before you have run a pipeline that includes a Transformations node, or from a Manage Variables node in your pipeline before the Transformations node.

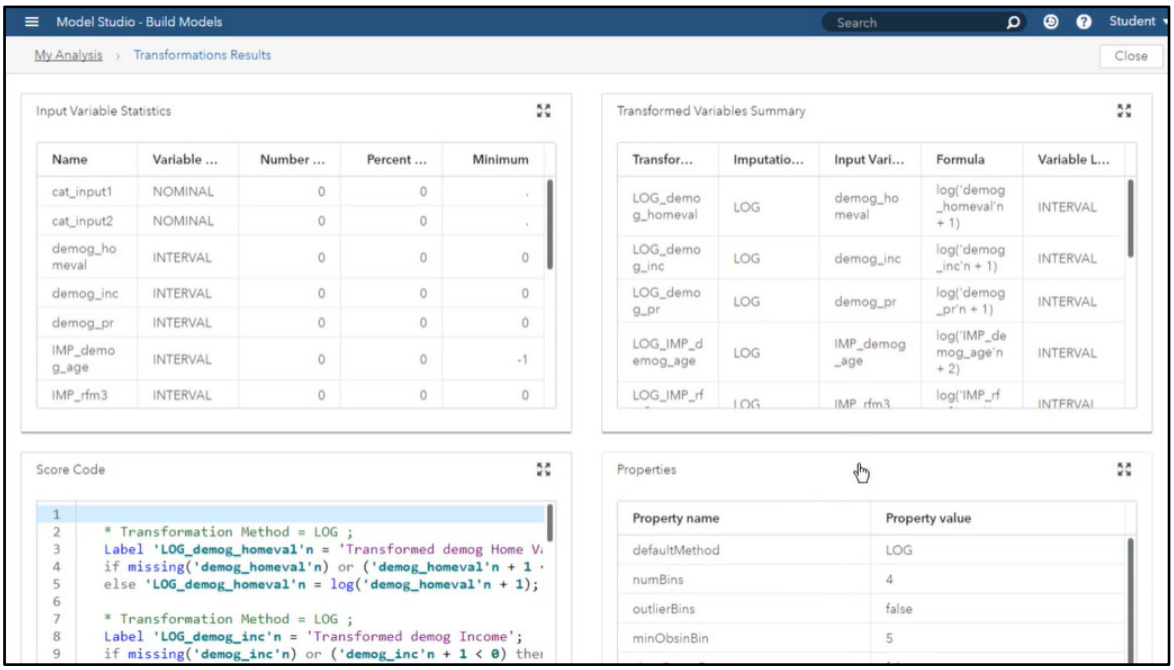
Notice that the Transformations node options appear to the right of the pipeline. Change the **Default interval inputs method** property from None to Log. The original variables will be rejected as indicated by the selected check box for the property Reject original variables.

Right-click the **Transformations** node and select **Run**. When the Transformations node is finished running, right-click on the node again and select **Results** to see the results.

Results

The Results window contains information about the log transformations, summary statistics, and score code, as shown in Figure 4.20.

Figure 4.20: Transformation Results



The Transformed Variables Summary table provides detailed information about the transformation, including the formula. Notice the offset applied to each log transformation. The imputed demographic age variable has an offset of 2, implying that the minimum value for age is -1. The oddity is confirmed when we examine the Input Variable Statistics table. This is an indication of a possible data issue.

Close the Results window to return to the pipeline.

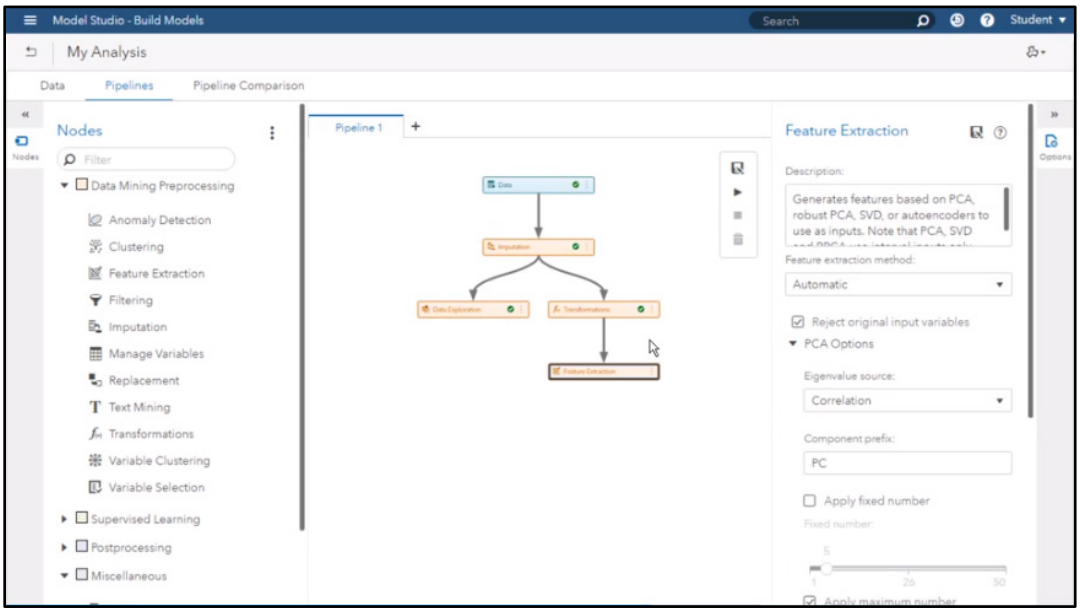
Feature Extraction

We are now going to create two new sets of features. The first set of features will be extracted using singular value decomposition. The second set of features will be extracted using an auto encoder neural network.

Build Pipeline

Select **Nodes**, then select the **Feature Extraction** node under the **Data Mining and Preprocessing** group. Click and drag the Feature Extraction node on top of the Transformations node to create a link between the two nodes, as shown in Figure 4.21.

Figure 4.21: Feature Extraction Node

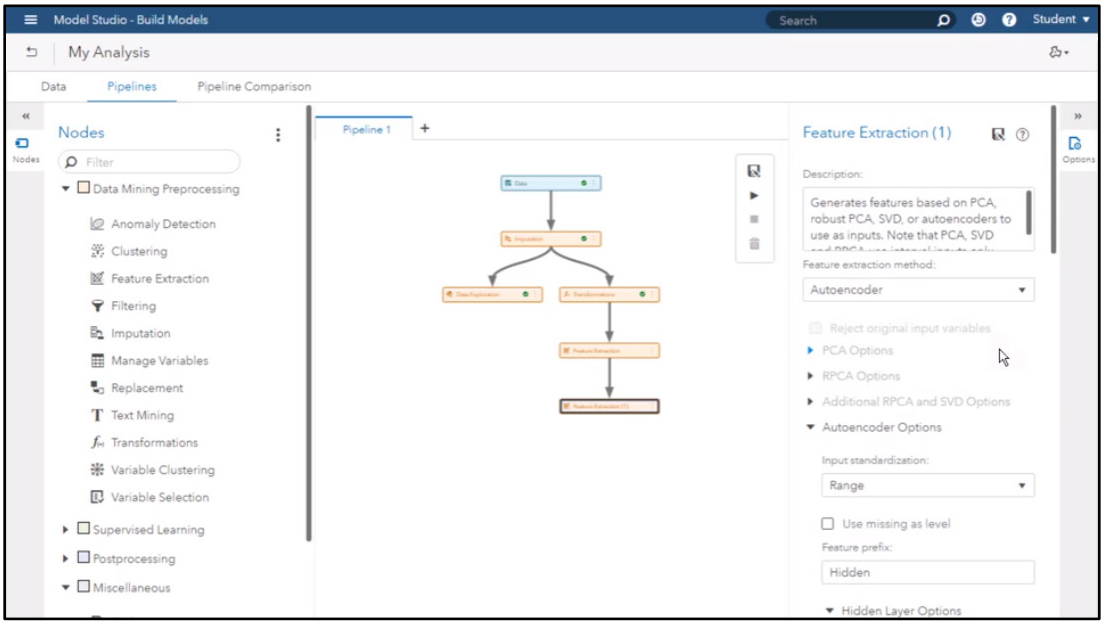


In the right-hand pane, we can see the options for the Feature Extraction node. Let's change the **Feature extraction method** from Automatic to Singular value decomposition. Notice that the original variables will be rejected as indicated by the selected check box for the property **Reject original input variables**.

Under the **Additional RCPA and SVD** options, use the **Maximum rank** property to choose the maximum number of latent variables to pass forward in the pipeline. We will leave the property set to 100. The **Component prefix** property sets the prefix for the latent variables created by the singular value decomposition transformation.

We need to add a second Feature Extraction node. Click and drag the **Feature Extraction** node on top of the existing Feature Extraction node in the pipeline to create a link between the two nodes, as shown in Figure 4.22.

Figure 4.22: Second Feature Extraction Node



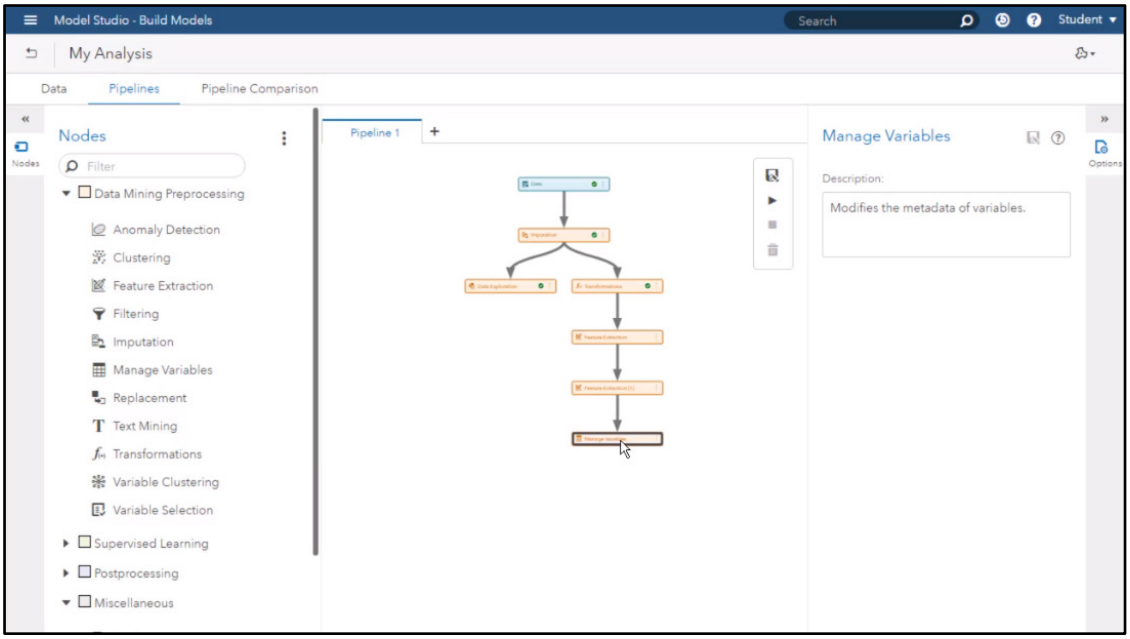
This time, we will use the Feature Extraction node to create encoders. Therefore, in the **Feature Extraction** options, change the **Feature extraction method** from Automatic to Autoencoder. This time, we will not reject the incoming variables. Deselect the check box for the **Reject original input variables** property. Under the

Autoencoder Options section, we will leave the **Input standardization method** set to Range. The **Feature prefix** property specifies the prefix used for the encoders. We will leave this property set to Hidden. The **Number of hidden layers** property specifies the number of hidden layers to use. We will leave this property set to 3. We will change the number of hidden neurons in the Middle hidden layer property from 10 to 25. This means that as many as 25 encoders will be passed forward in the pipeline. Next, change the **Hidden layer activation function** from the hyperbolic tangent denoted as Tanh to the rectified linear activation function denoted as ReLU.

Under the **Stochastic Gradient Decent Optimization** (SGD) options, change the Minibatch size property from 10 to 100. This means the weight vector be updated each iteration using 100 observations sampled from the data. Assign the momentum rate to be .9 to ensure a smoother progression to the error minimum.

Lastly, we need to add a **Manage Variables** node to the pipeline that we can use to reject any encoders that have a unary measurement level. Select **Nodes**, then select the **Manage Variables** node under the **Data Mining Preprocessing** group. Drag it on to the last **Feature Extraction** node in the pipeline, as shown in Figure 4.23.

Figure 4.23: Manage Variables Node

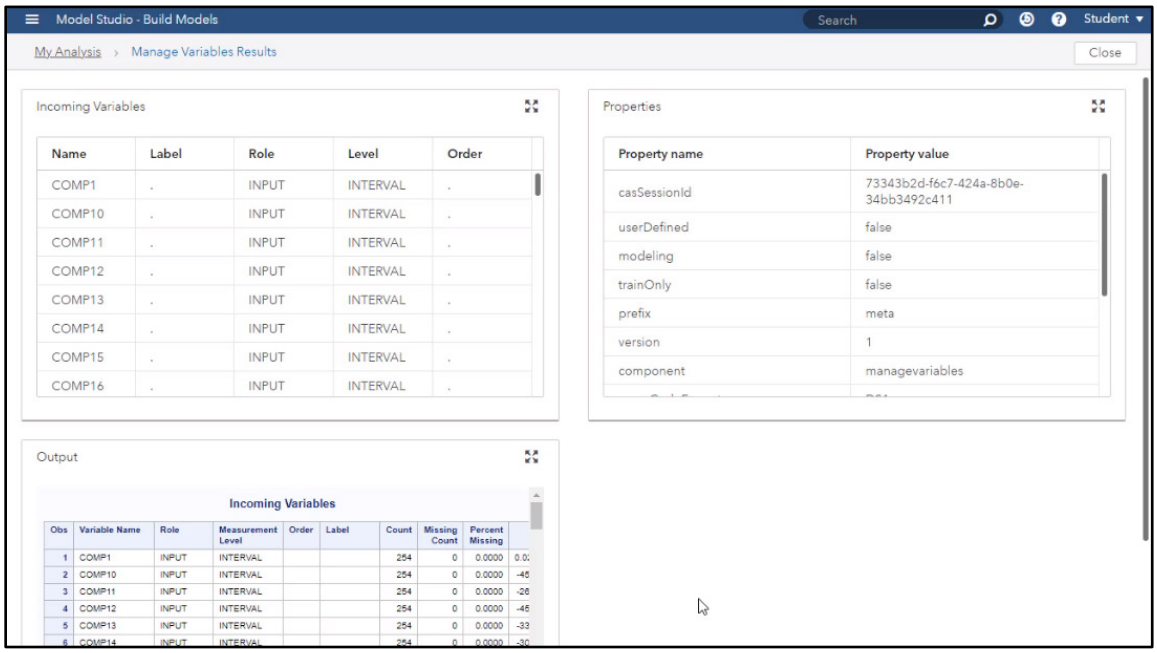


Right-click the **Manage Variables** node in the pipeline and select **Run**. When the node is finished running, right-click it again and select Results.

Results

The Manage Variables results open in a new window, as shown in Figure 4.24.

Figure 4.24: Manage Variables Results



Notice that the variables created by the singular value decomposition method are designated with the prefix COMP in the Incoming Variables table. Scroll down in the Incoming Variables table to see that the encoders that were created have the prefix Hidden. Close the Results window.

Now you have seen how to transform your variables and create new features in Model Studio.

Variable Selection

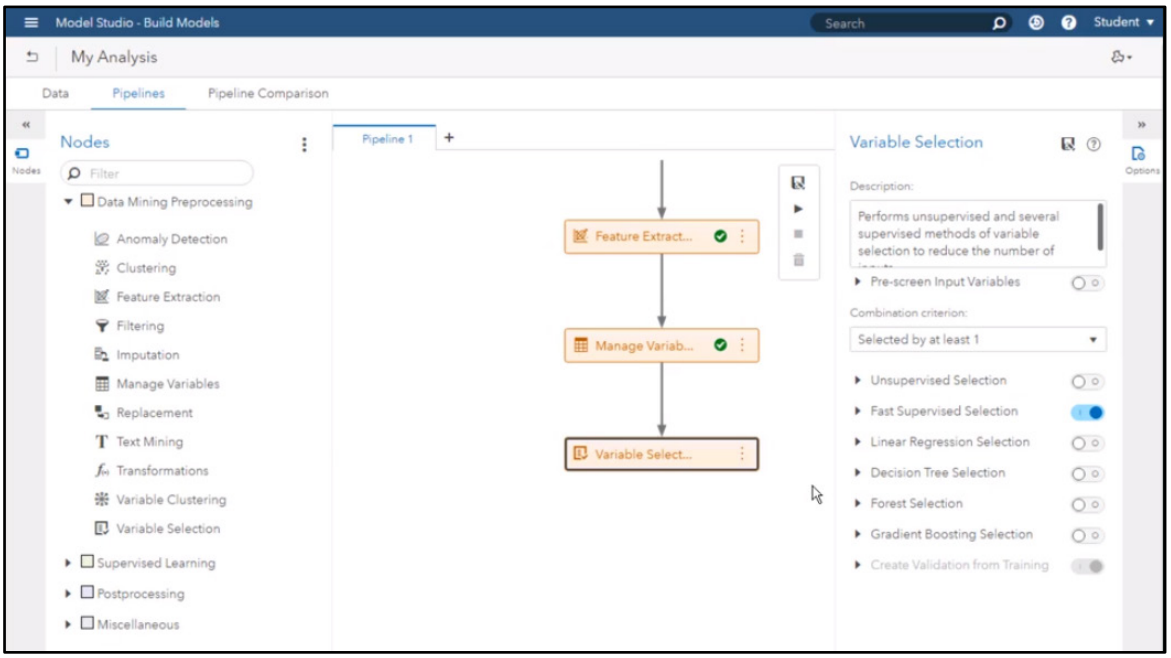
In this section, we will learn how to use Model Studio to select variables for a predictive analysis. To begin, on the Home Page, open the My Analysis project that we created in the previous sections. Data has already been uploaded into the project.

Select the **Pipelines** tab to go to the pipeline. We can see that data has been uploaded, imputations have already been applied, and the data has been enriched with additional features. To select useful inputs, we are going to add a Variable Selection node.

Build Pipeline

Select **Nodes** from the far-left panel. Then choose the **Variable Selection** node under the **Data Mining Preprocessing** group. Click and drag the Variable Selection node on top of the Manage Variables node to create a link between the two nodes, as shown in Figure 4.25.

Figure 4.25: Variable Selection Node



Notice in the **Variable Selection** panel, there are six variable selection methods that be used individually or combined. Given that we are going to combine multiple methods, we need to specify a rule that will determine the final selection of inputs. For example, the default rule as given under the Combination criterion is to pass forward any input that was chosen by at least one method. We are going to use three methods:

- Unsupervised Selection
- Fast Supervised Selection
- Forest Selection

First, switch on the **Unsupervised Selection** property. Expand the Unsupervised Selection section and change the **Selection process** property from Perform sequential selection, which specifies to use Unsupervised Selection before Supervised Selection, to Combine with supervised method(s). The Combine with supervised method(s) property combines unsupervised selection with supervised selection methods via the combination criterion.

Fast Supervised Selection is selected by default. Therefore, no changes are needed to that property. The fast supervised selection identifies a set of variables that jointly explain the maximum amount of variance contained in the target.

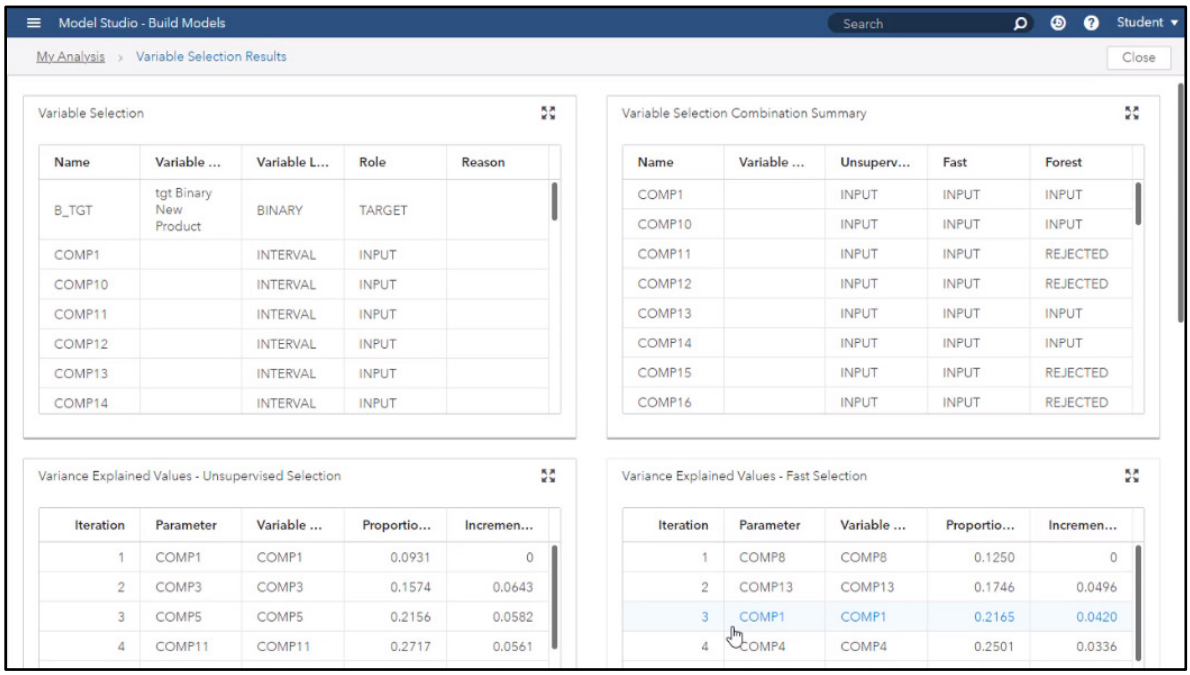
Next, switch on **Forest Selection**. Notice when you expand the section that the **Relative importance cutoff** property is set to .25. This property specifies the relative importance cutoff used to determine whether an input is rejected. Any input with a relative importance below this cutoff is rejected. Let's reduce the threshold from .25 to .1.

Now we are ready to run the Variable Selection Node. Right-click the node and select **Run** from the drop-down menu. When the node is finished running, right-click it again and select Results.

Results

In the Variable Selection results, each variable selection generates a table describing the respective results, as shown in Figure 4.26.

Figure 4.26: Variable Selection Results



The Variable Selection Combination Summary table displays a summary of the variables selected by all of the specified methods. We can sort by the Unsupervised column by clicking on the column header. We can see that the Unsupervised method selected a number of variables. We can also sort by the Fast and Forest columns to examine the inputs chosen for each method. The Input column shows the number of methods that chose an input. Sorting the Input column from largest to smallest shows that feature COMP13 was chosen by all three methods! The last column in the table, labeled Output Role, indicates which variables are going to be passed forward as inputs in the pipeline. Sorting by this column shows the variables that were selected and will be passed forward in the pipeline.

Since we used the Combination criterion Selected by at least 1, a variable would have to be rejected by all methods to have its output role set to Rejected. If one method flags a variable as an input in our example, it will be kept.

Gradient Boosting Model

In this section, you will learn how to build a gradient boosting model in Model Studio and use the autotune feature to build a better gradient boosting model. To begin, let's start in the Model Studio home page. Double-click on the existing project called My Analysis that was created in a previous section. Recall that the data set used for this project is called VS_BANKDATA and it identifies customers who have purchased a banking product. The binary target variable is b_tgt.

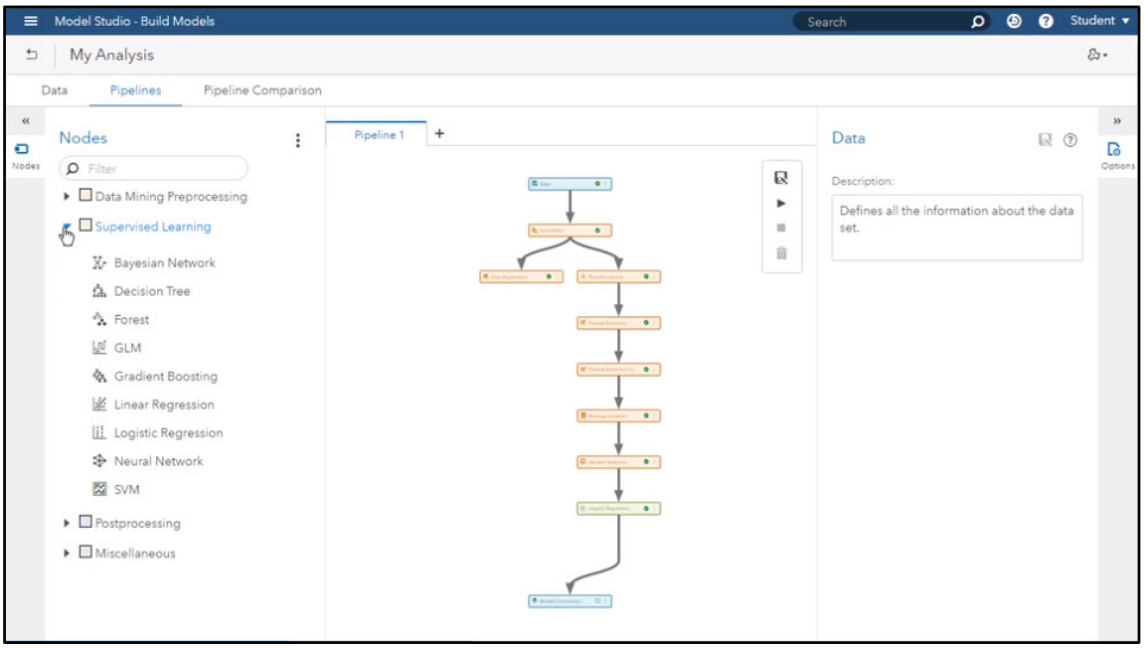
Train Model

Build Pipeline

Click the **Pipelines** tab to view the pipeline that was created in the previous sections. In preparation for model building, imputation of missing values, feature extraction, and variable selection have already been performed.

Click **Nodes** in the panel on the left. Expand the **Supervised Learning** group. Drag and drop the **Logistic Regression** node onto the last node in the pipeline, the Variable Selection node. Notice that in addition to the Logistic Regression node, a Model Comparison node also appears at the end of the pipeline by default, as shown in Figure 4.27. All modeling nodes that you add to the pipeline will automatically feed results into this node for easy comparison of models.

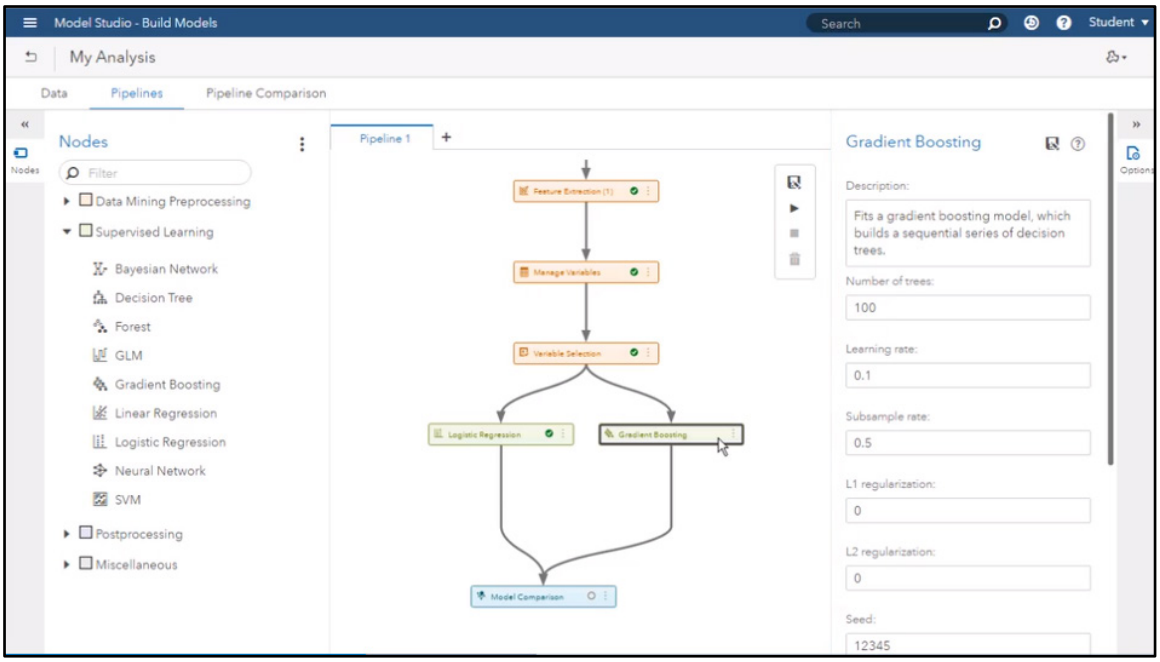
Figure 4.27: Logistic Regression Node



Next, let's build a gradient boosting model, which is based on a more sophisticated machine learning algorithm compared to the logistic regression model. Then we can compare the two models.

To create a gradient boosting model, return to the **Nodes** panel in the left pane and choose **Gradient Boosting** from the **Supervised Learning** group. Drag and drop the Gradient Boosting node on top of the Variable Selection node, as shown in Figure 4.28.

Figure 4.28: Gradient Boosting Node



Notice that when the Gradient Boosting node is added, it's automatically connected to the Model Comparison node. Click the Gradient Boosting node. The Gradient Boosting options panel appears to the right. Some of the general properties define:

- **Number of trees** – specifies the number of iterations in the boosting series. The default is 100.
- **Learning rate** – specifies the step size for gradient descent optimization. The default is 0.1.

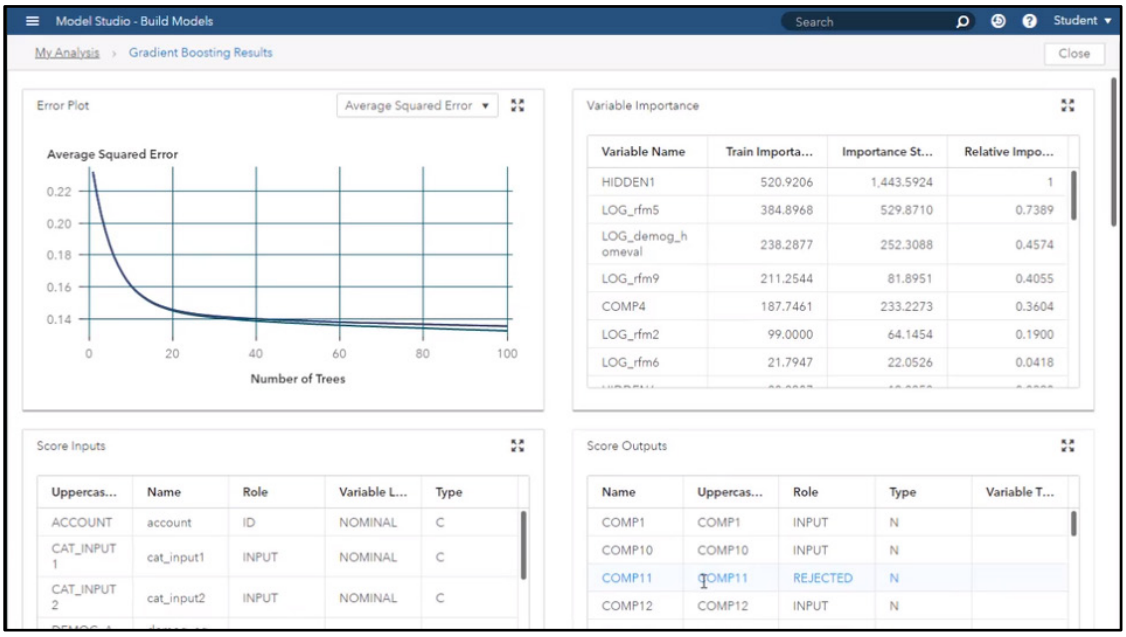
- **Subsample rate** – specifies the proportion of observations used to train a single tree. The default is 0.5.
- **L1 and L2 regularization terms –** specify the L1 and L2 regularization parameters, respectively. The default for each is 0. Regularization is a process used during model training to avoid overfitting the model to the training data. The L1 term is also known as the Lasso regularization. It penalizes the absolute value of the weights. Analogously, the L2 term is also known as the Ridge regularization, and it penalizes the square value of the weights.
- **Seed** – used for generating random numbers. The default is 12345. Note that specifying the Seed does not guarantee repeatability when the training is performed on data distributed among a grid of worker nodes.

Let's run the model with the default settings. This means that 100 trees will be used in the series. Click the Run shortcut button (▶) in the box in the upper right corner of the Pipeline window. When the run is complete, right-click the **Gradient Boosting** node, then select **Results** from the drop-down menu.

Results

In the Results window (Figure 4.29), we see an error plot, a table showing variable importance, plots for lift and ROC, a table of fit statistics, and output from the GRADBOOST procedure.

Figure 4.29: Gradient Boosting Results



The Fit Statistics table shows that Average Squared Error for the final model on the test data set is 0.1844. Let's see if we can make a few changes to the node properties and beat the default gradient boosting model. Close the Results window to return to the pipeline.

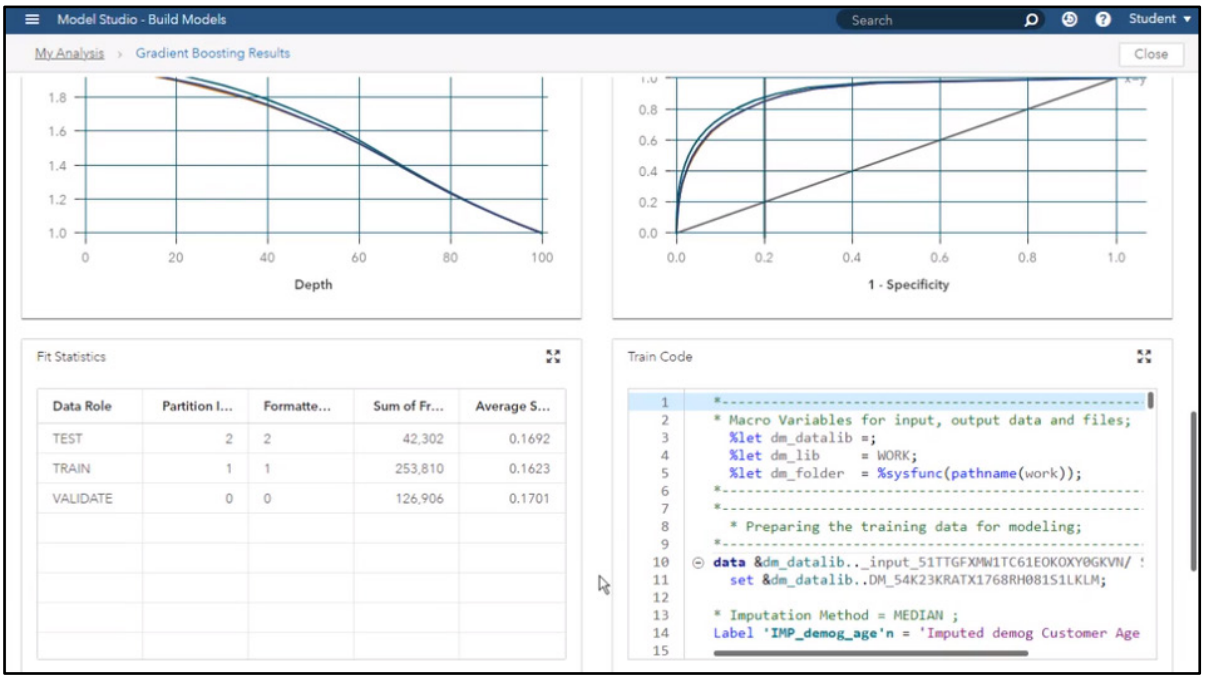
Modify Node Properties

In the Gradient Boosting properties panel, change the **Number of trees** to 200, and increase the **Learning rate** to 0.2. We will re-run the pipeline from the Model Comparison node. Right-click the **Model Comparison** node and select **Run**. When the run is complete, right-click the **Gradient Boosting** node and select **Results**.

Modified Results

Scroll down in the results to see the Fit Statistics table. As shown in Figure 4.30, the changes that we made did improve the performance of the model. Average Squared Error on the test data set dropped to 0.1692. Close the Results window.

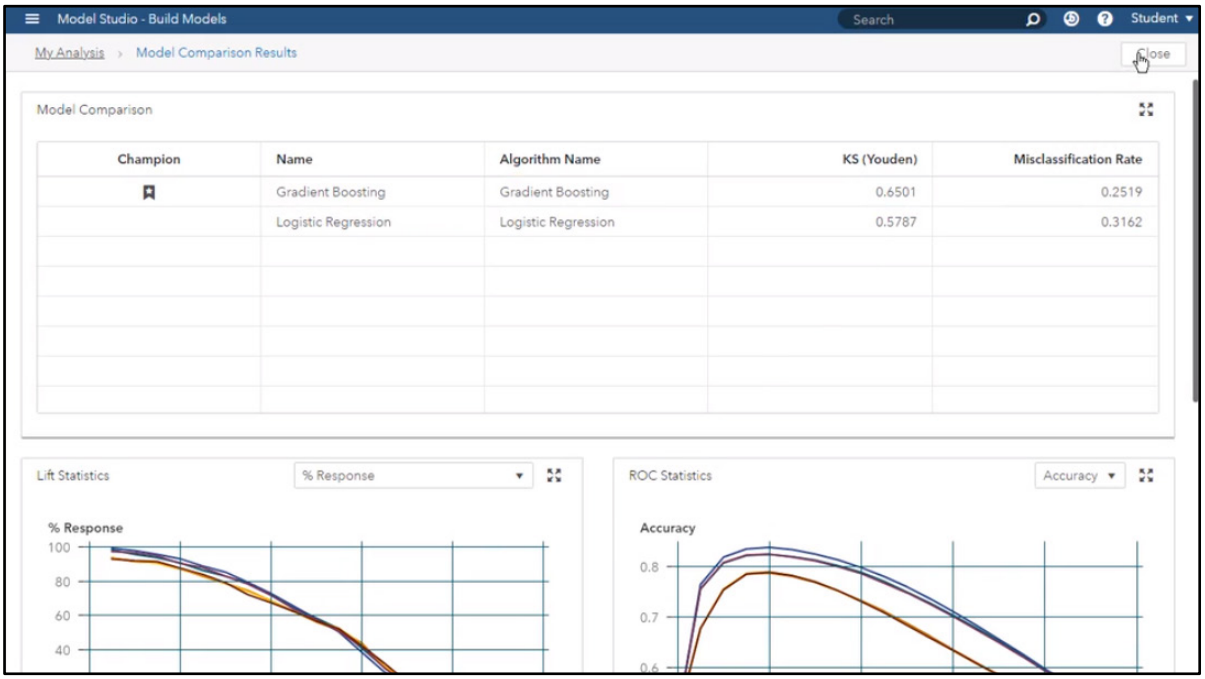
Figure 4.30: Modified Gradient Boosting Results



Model Comparison

Let's look at which model has been selected as the champion. Right-click the **Model Comparison** node and select **Results**. The results of the Model Comparison node in Figure 4.31 show that the Gradient Boosting model is selected as the champion. Close the Results window to return to the pipeline.

Figure 4.31: Model Comparison Results



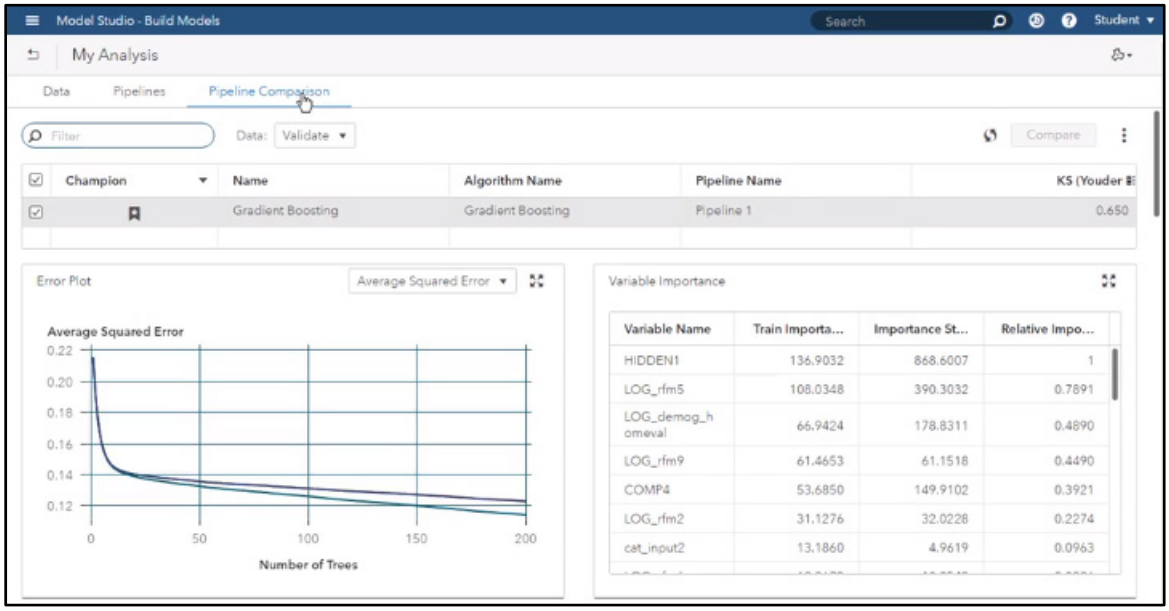
When you are satisfied with your model, you can download the score code to use for scoring future data. Right-click the **Gradient Boosting** node and select **Download score code** from the menu. The score code downloads as a ZIP file for future use.

You can also save a node to reuse the configuration in other pipelines. Right-click the Gr**adient Boosting** node. Notice that you have the option to **Save as…** in the menu.

Pipeline Comparison Tab

Click the **Pipeline Comparison** tab at the top of the screen. The best (or champion) model from your pipeline will show up as a champion on the Pipeline Comparisons tab, as shown in Figure 4.32.

Figure 4.32: Pipeline Comparison Tab



Click the **Pipeline** tab to return to the pipeline. If you want to include a non-champion model on the Pipeline Comparison tab, you can add a challenger model for the model that you want to add. Right-click the model node on the pipeline and select **Add challenger model** from the menu. When you return to the Pipeline Comparison tab, you can see that the other model has been added.

Autotune Model

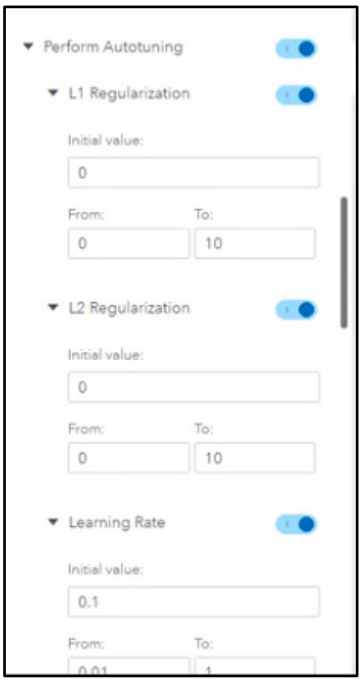
Click the **Pipelines** tab to view the pipeline that was created in the previous section. Let's focus just on the Gradient Boosting model. Right-click on the **Logistic Regression** node and select **Delete** from the menu to remove it from the pipeline.

Autotune Options

Click the **Gradient Boosting** node. Notice in the Gradient Boosting panel on the right that toward the bottom of the panel there is an option called **Perform Autotuning**. Turn this property on to activate a search strategy that will try many different combinations of option values to attempt to find a better model. Note that when you select the Perform Autotuning feature, new properties become visible and many of the original properties for the node become inactive because they are controlled by the autotuning process.

Notice in Figure 4.33 that the autotune feature will try out a range of values for several of the gradient boosting properties. Autotuning can be turned on or off for each individual property. By default, it is turned on for all properties.

Figure 4.33: Gradient Boosting Panel – Perform Autotuning Option



All properties that are autotuned have an initial starting value and a range within which to search. The starting values and the From and To values can be changed for these properties.

The L1 and L2 Regularization terms specify L1 and L2 Regularization parameters. The L1 and L2 terms were defined in the previous section. The Learning Rate specifies the step size for gradient descent optimization. The Number of Inputs per Split determines the number of input variables randomly sampled to use per split. The Number of Iterations determines the number of trees in the boosting series. The Subsample Rate determines the proportion of training observations to train a tree with.

In addition to these properties, the autotuning process has a number of Search Options and General Options that can be set. The Search Options enable you to specify the search method and any options associated with the selected search method. The default search method uses a genetic algorithm optimization method to intelligently search for the best combination of hyperparameter values. This method and the Bayesian method evaluate numerous candidate models over multiple iterations. Thus, you can specify the number of evaluations in each iteration, the maximum number of iterations, and the maximum number of evaluations. The Latin hypercube sample and Random methods enable you to specify the sample size.

The General Options specify the general properties for autotuning, regardless of the search method used. Because autotuning works by evaluating each candidate model using validation data, you can specify whether to use partitioning (along with the corresponding validation data proportion), or cross validation (along with the number of folds). You can specify the objective function to use for evaluating models for either nominal or interval targets, and the maximum time that you want the autotuning to run. Notice that for class targets, the default objective function is Misclassification rate. This means that for the VS_BANKDATA data set with the binary target b_tgt, the resulting gradient boosting model is the one with the minimum Misclassification rate on the validation data set. We will leave all properties for autotuning turned on and set at their default values.

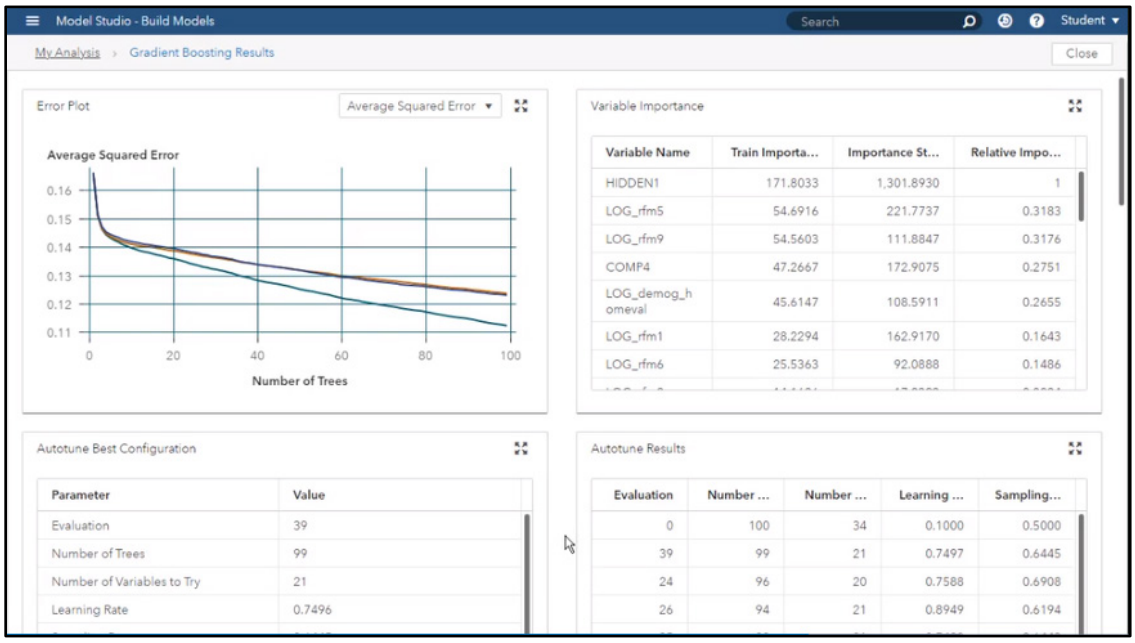
Keep in mind that while performing autotuning can substantially increase the run time because it's training many candidate models, the autotuning process will attempt to train as many in parallel as your environment will allow.

To run the Pipeline, click the **Run** shortcut button (▶). When the run is complete, right-click the **Gradient Boosting** node and then select **Results** from the menu.

Results

The results shown in Figure 4.34 include many fit statistics and charts to describe the best model found by autotuning including an error plot, a table of variable importance, plots for lift and ROC, a fit statistics table, and output from the GRADBOOST procedure.

Figure 4.34: Gradient Boosting Results



The Fit Statistics table shows that the final model had an Average Squared Error value of 0.1669 on the test data set. Recall, however, that by default, misclassification rate is used to select the final model.

The Autotune Best Configuration table shows the resulting values of properties that were autotuned. The table shows that the best configuration came from evaluation 39, which used 99 trees in the ensemble and where a random sample of 21 input variables was tried at each split.

The Autotune Results table displays the top 10 models found by the autotuning process, enabling you to consider tradeoffs among models of varying complexity with similar accuracy. Recall that misclassification was used to determine the optimal model.

If you want to see the entire history of all candidate models considered in the autotuning process, you can expand the window showing the ODS output results for the GRADBOOST procedure and scroll down to see the Tuner History table shown in Figure 4.35.

Figure 4.35: Tuner History Table

Tuner History
All Evaluated Configurations

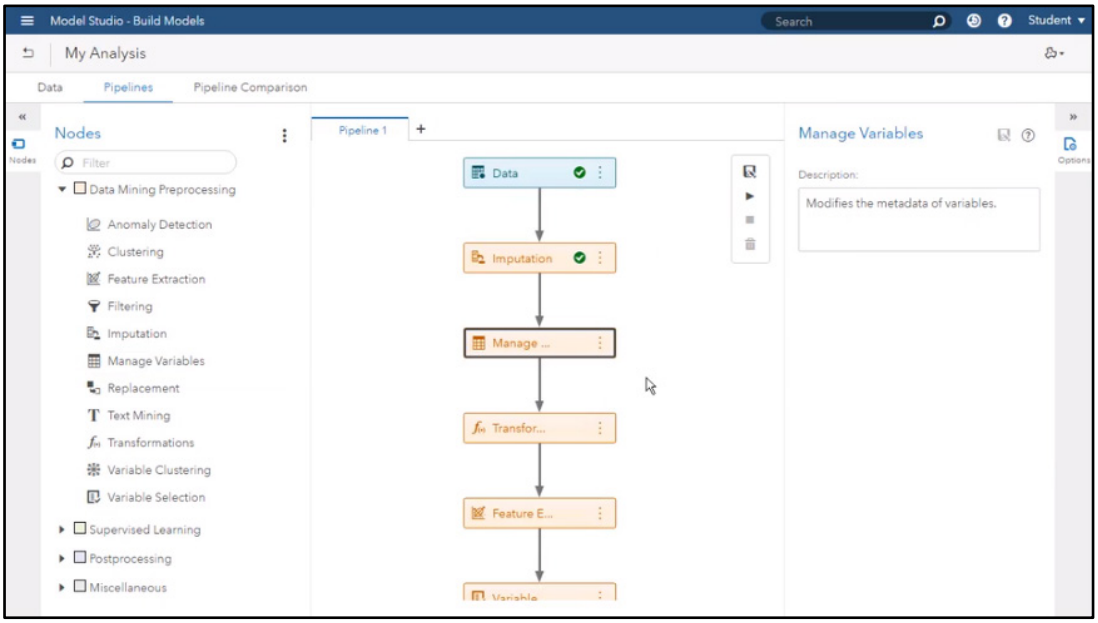
Evaluation	Iteration	Number of Trees	Number of Variables to Try	Learning Rate	Sampling Rate	Lasso	Ridge	Misclassification Error Percentage	Time In Seconds
0	0	100	34	0.100000	0.500000	0	0	19.36	51.34
1	1	100	34	0.100000	0.500000	0	0	19.36	0.00
2	1	63	1	0.230000	0.900000	3.333333	3.333333	20.21	83.78
3	1	107	30	0.890000	0.300000	1.111111	8.888889	18.46	146.06
4	1	150	27	0.120000	0.500000	0	0	18.82	204.11
5	1	121	19	0.460000	0.400000	10.000000	10.000000	18.36	164.58
6	1	20	8	0.560000	0.100000	4.444444	5.555556	20.47	26.78
7	1	78	12	0.780000	1.000000	6.666667	6.666667	18.07	100.99
8	1	34	23	0.010000	0.600000	8.888889	1.111111	20.76	46.28
9	1	92	16	0.340000	0.800000	5.555556	7.777778	18.81	79.39
10	1	49	5	1.000000	0.700000	2.222222	4.444444	19.51	42.18
11	2	127	15	0.547950	0.372014	7.425480	7.032077	18.04	160.66
12	2	106	30	0.197950	0.472014	2.798570	2.798570	18.80	137.54
13	2	111	27	0.775499	0.326023	3.424267	9.178033	18.22	143.65
14	2	149	34	0.811113	0.100000	0.314277	10.000000	20.73	58.57
15	2	91	23	0.921113	0.413140	1.425388	7.631779	18.32	120.30
16	2	73	32	1.000000	0.140321	1.998215	10.000000	20.74	95.85
17	2	116	29	0.734783	0.340321	0.887104	7.096832	17.85	135.54
18	2	108	29	1.000000	0.271418	1.269900	10.000000	18.96	122.27
19	2	101	33	0.212899	0.471418	0.158789	1.270315	18.73	107.72
20	3	97	34	0.226198	0.470687	0	0	18.67	122.61

Manage Variables

Earlier in this chapter, you learned how to manage variables by using the Edit Variable feature in the Data tab to edit a variable's metadata. In cases where you want to use different inputs for some models, or apply custom metadata that's different from what is defined on the Data tab, you can use the Manage Variables node to modify the variable metadata that will be used for subsequent nodes in the pipeline.

In any existing pipeline, click **Nodes** in the left panel. Under the **Data Mining Preprocessing** group, select the **Manage Variables** node. Drag it between the nodes where you want to place it. In the example shown in Figure 4.36, we will drag it between the Imputation and Transformation nodes.

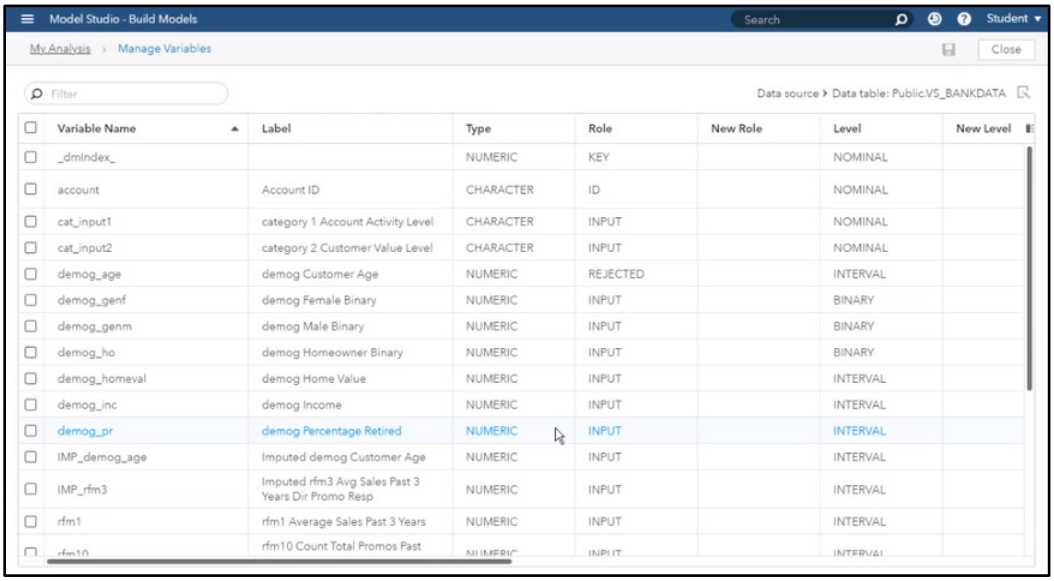
Figure 4.36: Manage Variables Node



It's important to note that you must first execute the Manage Variables node after you insert it so that the node can import the existing variables and their metadata at that point in the pipeline, enabling you to make modifications to their current definition.

Right-click the **Manage Variables** node and select **Run**. After the node has finished running, right-click it again and select **Manage Variables** from the menu. Now you can bring up the editor (see Figure 4.37) to modify the variable metadata to be used for the rest of this branch of the pipeline.

Figure 4.37: Manage Variables Window



Variable Name	Label	Type	Role	New Role	Level	New Level
dmIndex		NUMERIC	KEY		NOMINAL	
account	Account ID	CHARACTER	ID		NOMINAL	
cat_input1	category 1 Account Activity Level	CHARACTER	INPUT		NOMINAL	
cat_input2	category 2 Customer Value Level	CHARACTER	INPUT		NOMINAL	
demog_age	demog Customer Age	NUMERIC	REJECTED		INTERVAL	
demog_genf	demog Female Binary	NUMERIC	INPUT		BINARY	
demog_genm	demog Male Binary	NUMERIC	INPUT		BINARY	
demog_ho	demog Homeowner Binary	NUMERIC	INPUT		BINARY	
demog_homeval	demog Home Value	NUMERIC	INPUT		INTERVAL	
demog_inc	demog Income	NUMERIC	INPUT		INTERVAL	
demog_pr	demog Percentage Retired	NUMERIC	INPUT		INTERVAL	
IMP_demog_age	Imputed demog Customer Age	NUMERIC	INPUT		INTERVAL	
IMP_rfm3	Imputed rfm3 Avg Sales Past 3 Years Dir Promo Resp	NUMERIC	INPUT		INTERVAL	
rfm1	rfm1 Average Sales Past 3 Years	NUMERIC	INPUT		INTERVAL	
rfm10	rfm10 Count Total Promos Past	NUMERIC	INPUT		INTERVAL	

Note that just as with the metadata defined on the Data tab, any variable metadata defined in the Manage Variables node will override the settings in subsequent preprocessing nodes. Close the window to return to the pipeline.

You can insert multiple Manage Variables nodes in different branches of your pipeline to affect the variables used in each branch separately. Or, you can build different pipelines that use different sets of variables and variable metadata for the entire pipeline.

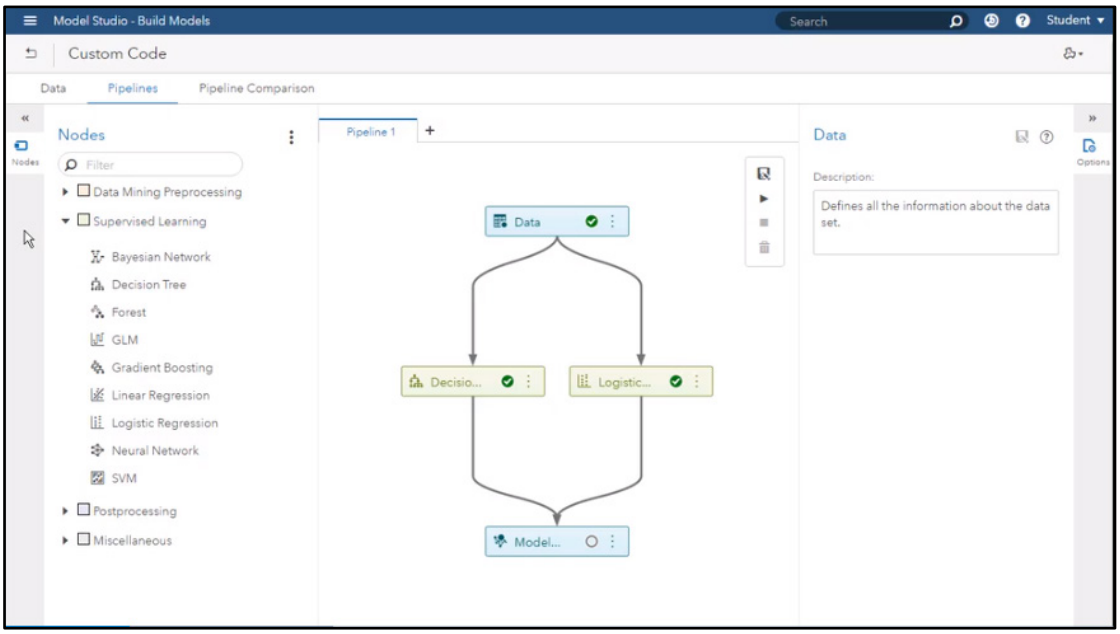
To start a new pipeline from inside the Pipelines tab, click the + sign next to the last existing pipeline to open a new pipeline. Name your new pipeline using the **New Pipeline** window. Click **Save**. Because the Data tab does not allow editing variables after a pipeline is executed (to ensure consistent results), it's a good practice to insert a Manage Variables node at the beginning of new pipelines immediately after the Data node in case you want to edit the metadata later.

As you can see, by defining variable metadata for the entire project on the Data tab and modifying the variable metadata within pipelines using the Manage Variables node, you have ultimate flexibility in building automated machine learning pipelines.

Add Custom Code

In this section, you will learn how to incorporate custom SAS code into your projects in Model Studio. For this example, we will be using a pipeline that has already been constructed in a project titled "Custom Code." The pipeline shown in Figure 4.38 builds two different predictive models using nodes that are available in the Supervised Learning group category.

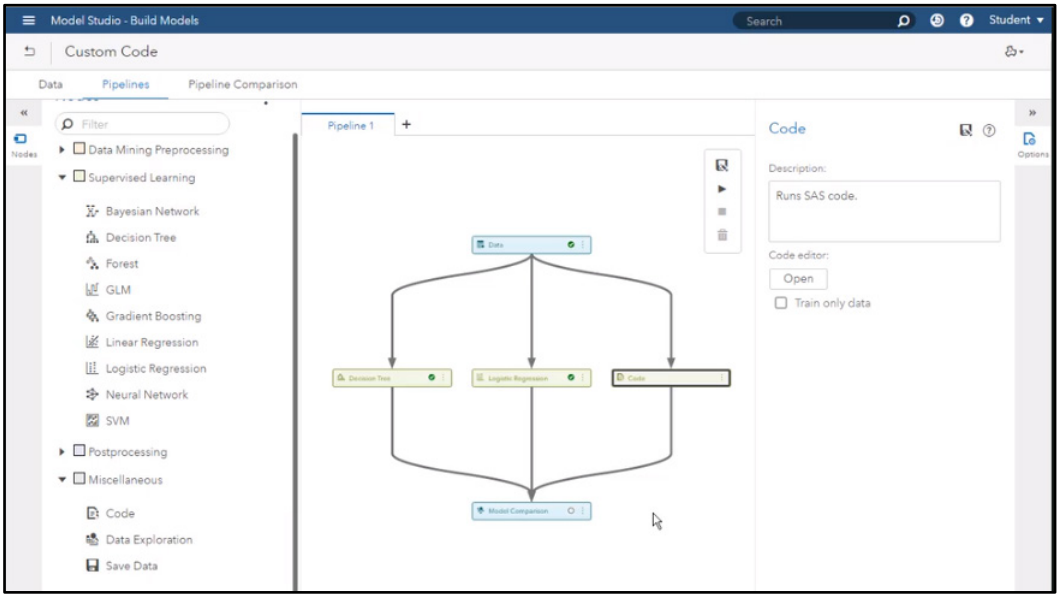
Figure 4.38: Example Pipeline



We want to compare these models against a model that's not supported by any of these existing supervised learning. To do this, we can use the Code node to write SAS code that will be executed as part of the pipeline.

The Code node can be found in the Miscellaneous category. Drag and drop the **Code** node onto the Data node to add it to the pipeline. Notice that it is a different color than the Supervised Learning nodes and is separated from them. By default, a Code node is considered to be a preprocessing node because you could write any SAS code to execute it. To indicate that this node produces a model, right-click the node and in the menu select **Move ▶ Supervised Learning**. Notice that the appearance of the Code node changes to match the other supervised learning nodes and is automatically connected to the Model Comparison node, as shown in Figure 4.39.

Figure 4.39: Modified Pipeline

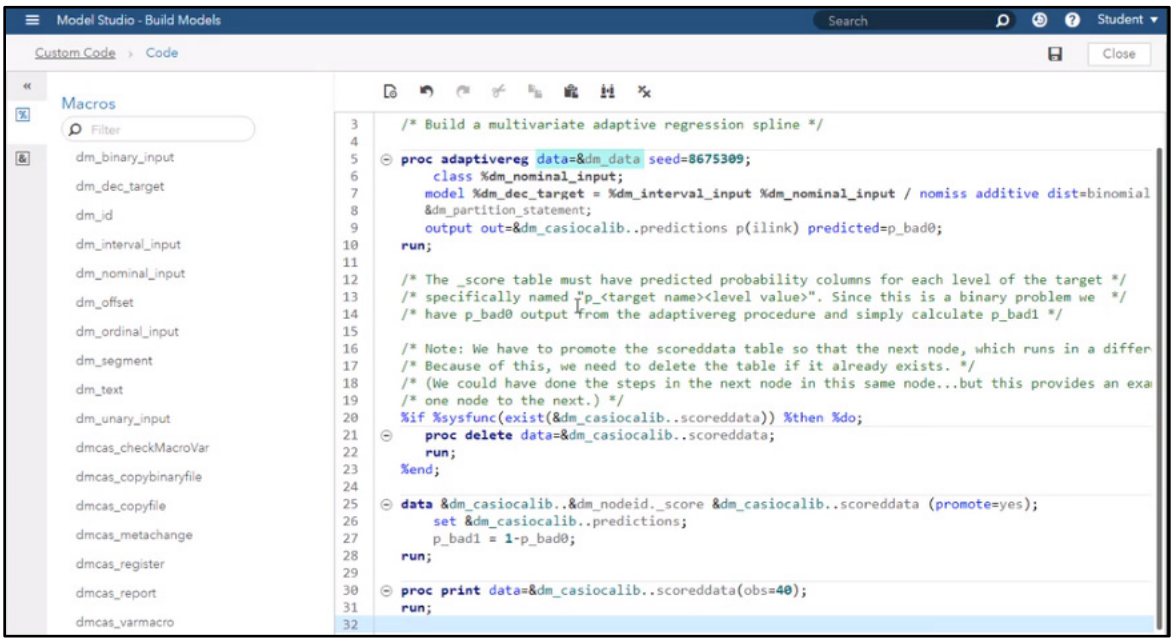


We are ready to enter the code for this node. Right-click the node and select **Open** from the menu to open the editor. The **Code Editor** provides an area for composing your code and includes helpful features such as syntax coloring, code insight prompting to present supported options, and autocomplete. It also provides a list of

macros and macro variables in the left panel to give you access to various data tables, variable metadata, and other information about the project and pipeline that might be useful in your program.

In this example, we will use an existing procedure, PROC ADAPTIVREG, to build an adaptive regression spline model to compare with the others. You can enter the code directly or paste it in from an external source, as shown in Figure 4.40.

Figure 4.40: Code Editor – PROC ADAPTIVREG



In line 5 of the code shown in Figure 4.40, we use the macro variable dm_data to reference the training data passed into the node. Notice in lines 6 and 7 of the code that the available macros were used to insert lists of variables based on roles and levels. In line 9, we output the predictions to a "predictions" table in the project caslib.

The built-in model assessment mechanism in Model Studio requires a predicted probability column for each target event. However, PROC ADAPTIVREG provide the predictions only for a specified event level, so in line 27 we calculate the non-event probability and add it to the scored data table.

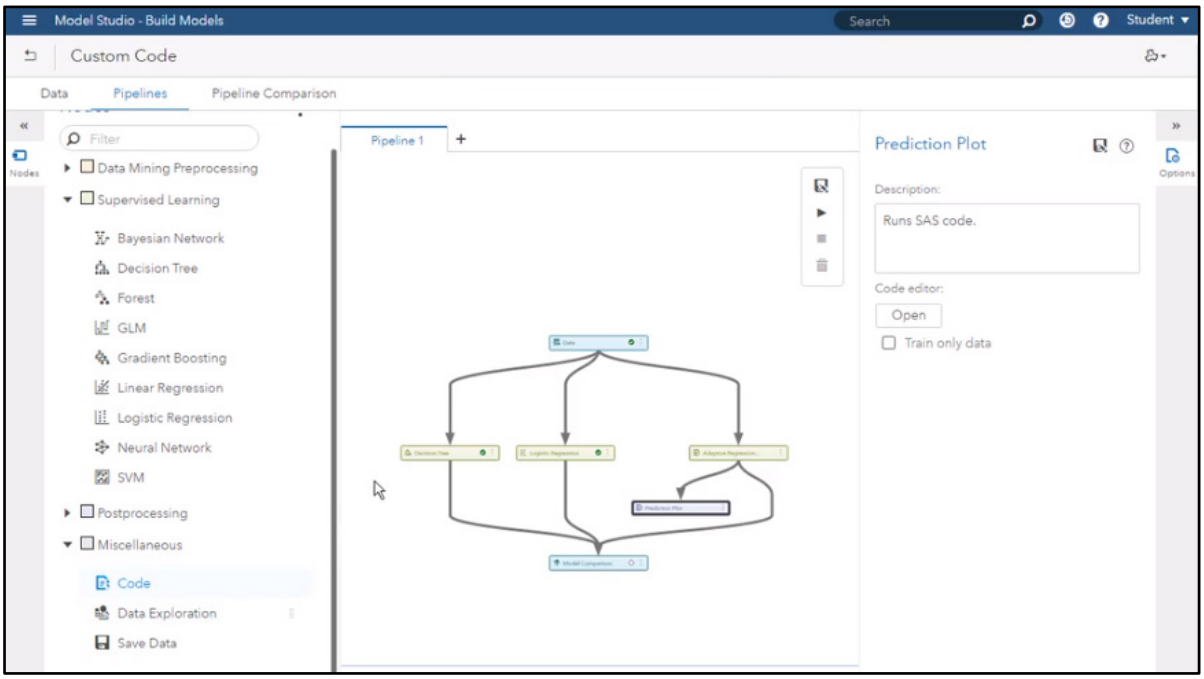
We also want to use this table in a downstream node in our pipeline. Because each node runs its own CAS session, we need to promote the table so that the downstream node can access it (line 25) and delete any table of the same name that already exists (lines 20–23).

Finally, in lines 30 and 31, we print 40 observations from the resulting table as a test. Click **Save** in the upper right corner of the window to save your code before clicking the **Close** button to close the Code Editor.

Back in the pipeline, let's rename the Code node. Right-click the node and chose **Rename…** from the menu. Type in the name Adaptive Regression Spline so that its purpose is more obvious.

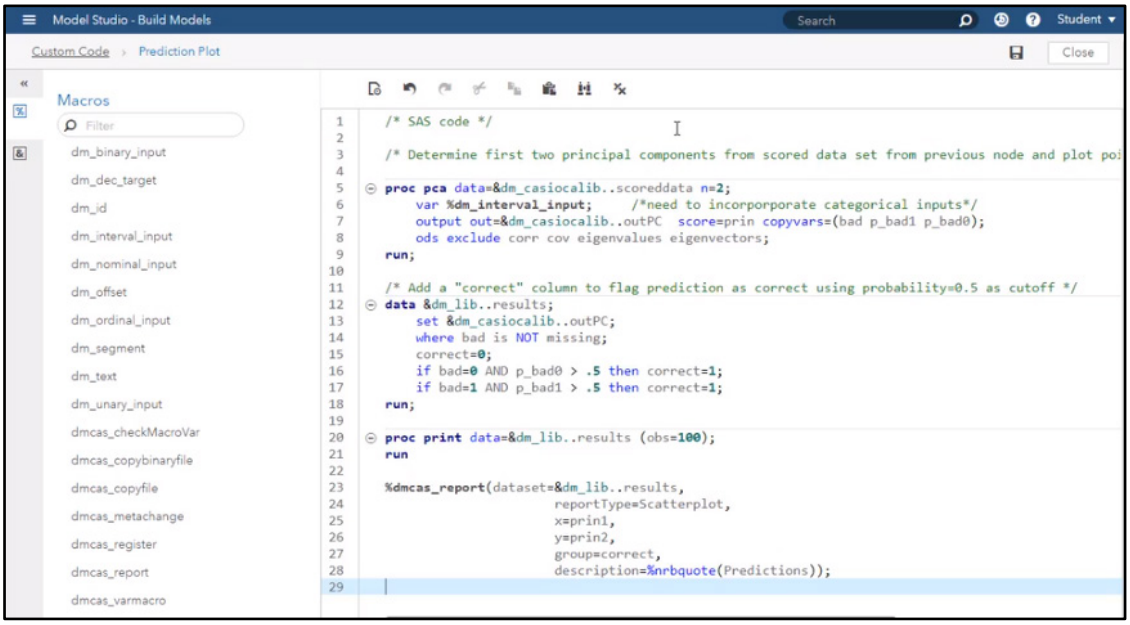
We also want to see a graphical representation of the model predictions from this node. We can do this by using a dimension reduction technique and some custom code. Add another Code node branching off the Adaptive Regression Spline node. Click the **Code** node in the **Miscellaneous** group, then drag and drop it on top of the Adaptive Regression Spline Node. Right-click the new **Code** node and rename it Prediction Plot. (See Figure 4.41.) Right-click the **Prediction Plot** node and click **Open** to open the Code Editor.

Figure 4.41: Modified Pipeline



In the editor, we will paste in the code that we want to execute, shown in Figure 4.42.

Figure 4.42: Code Editor – PROC PCA

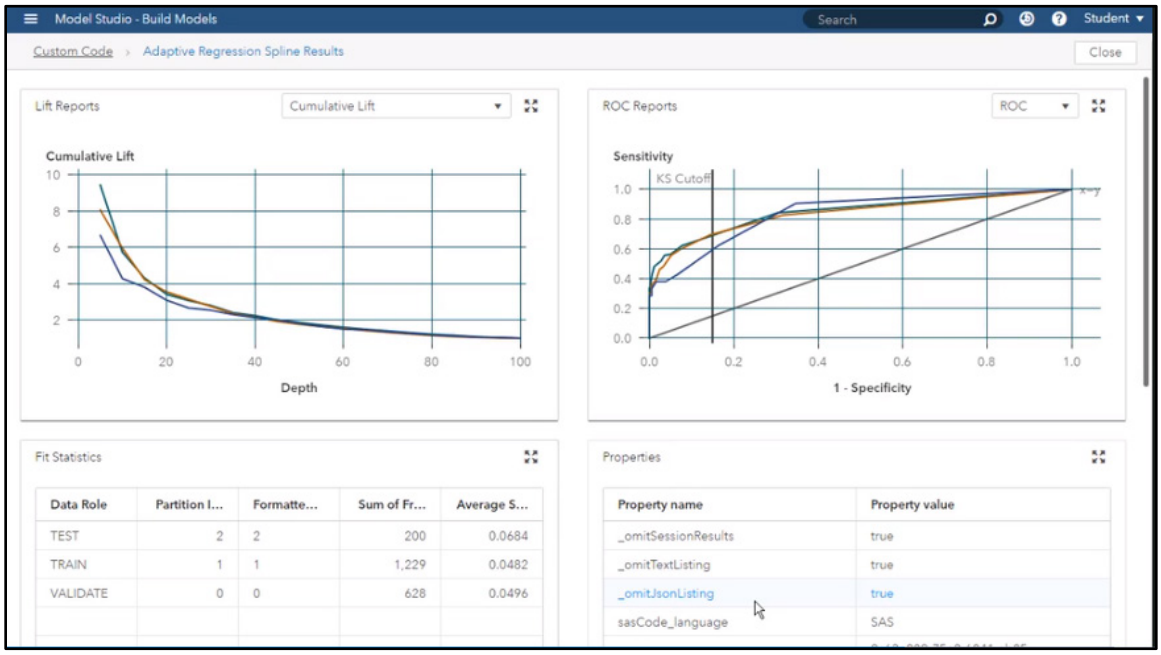


In lines 5–9, we run PROC PCA on the scored data set from the previous node to determine the first two principal components. That way, all our interval input information is represented in two dimensions. In lines 12–18, we want to plot points as correct or incorrect, so we add a variable named "correct" that is calculated for each observation based on the prediction and the actual target value. Finally, in lines 23–29, we use a specially provided macro, dmcas_report, to create a scatter plot of the two specified columns in the specified data set. Click Save, then close the window.

Now let's run the pipeline. Click the **Run** shortcut button (▶) in the pipeline window. Note that the supervised learning nodes that already ran do not re-execute, but the Model Comparison node does because the new adaptive regression node supplies it with a new model.

When the pipeline finishes executing, we can look at the results. First, let's explore the results for the Adaptive Regression Spline node. Right-click on the node and choose **Results** from the menu. Because this node is designated as a Supervised Learning node, all the assessment plots and fit statistics that are reported for any supervised learning node are automatically provided, as shown in Figure 4.43. In addition, the ODS output for all the SAS code is presented, including output from the ADAPTIVREG procedure and the PROC PRINT call.

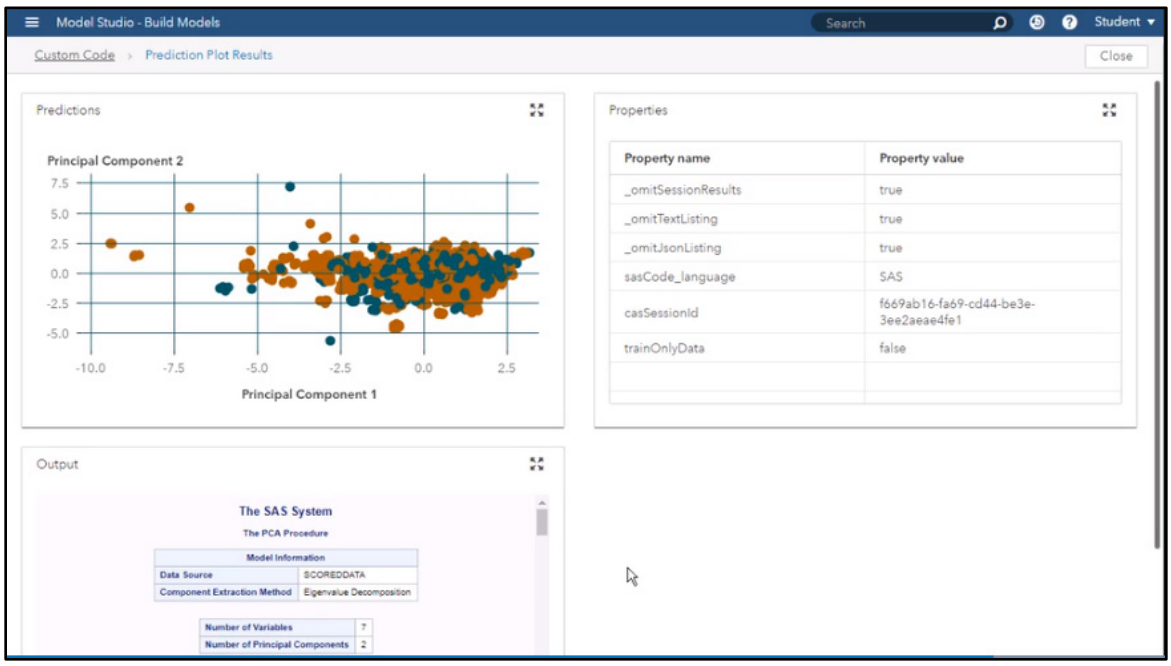
Figure 4.43: Adaptive Regression Spline Results



Click **Close** to return to the pipeline.

Now, let's look at the Prediction Plot node results. Right-click the **Prediction Plot** node and select **Results** from the menu. In the Prediction Plot results (Figure 4.44), we see the results of the PCA procedure and the scatter plot that we specified. We can see that our model seems to have reasonable prediction power, with many more correct predictions than incorrect.

Figure 4.44: Prediction Plot Results

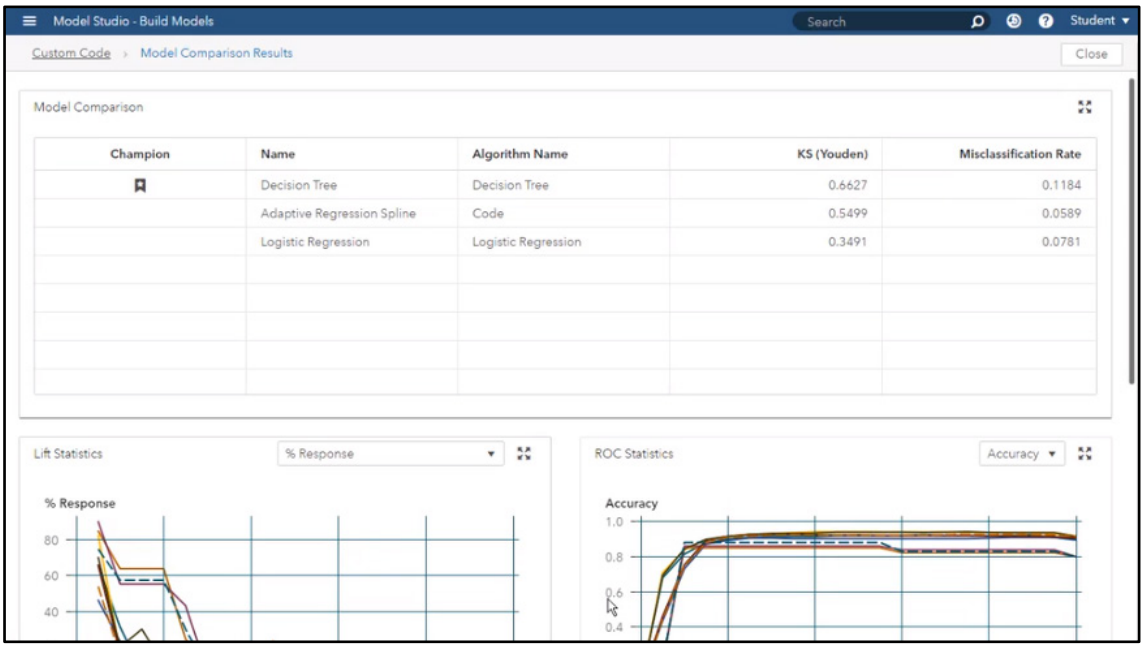


Close the window to return to the pipeline.

Finally, let's look at the Model Comparison node results. Right-click the **Model Comparison** node and select **Results** from the menu.

In the Model Comparison results in Figure 4.45, we see that the Adaptive Regression Spline has a very good misclassification rate compared to the other models, but the Decision Tree is still selected as the champion because our project was specified to use the KS statistic as the metric for selecting a champion model.

Figure 4.45: Model Comparison Results



At this point, we would want to take a closer look at our models and possibly tune and retrain them before deciding on a model that we are comfortable deploying. As you can see, the Code node provides great flexibility and functionality in automated machine learning pipelines.

Save Data for Use in Other Applications

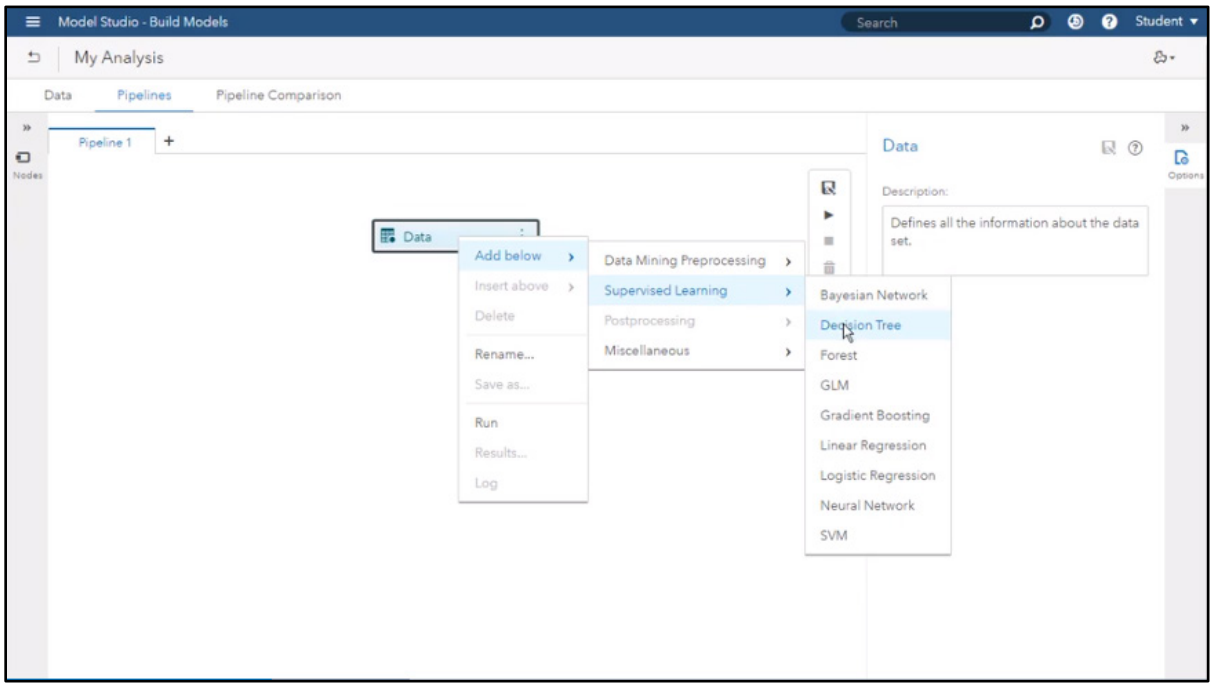
In this section, you will learn how to use the Save Data node in Model Studio. After you save a data set, you can use it later in other applications – for example, in SAS Visual Analytics.

To begin, let's start in Model Studio and open the My Analysis project that we have been modifying throughout this chapter. Click the **Pipelines** tab and click the (+) to open a new pipeline. Currently, the Data node is the only node in the pipeline. Let's learn another way to add nodes to a pipeline.

Build Pipeline

Right-click the **Data** node. From the menu choose **Add below ▶ Supervised Learning ▶ Decision Tree** to add a Decision Tree node to the pipeline, as shown in Figure 4.46.

Figure 4.46: Adding a Node to a Pipeline

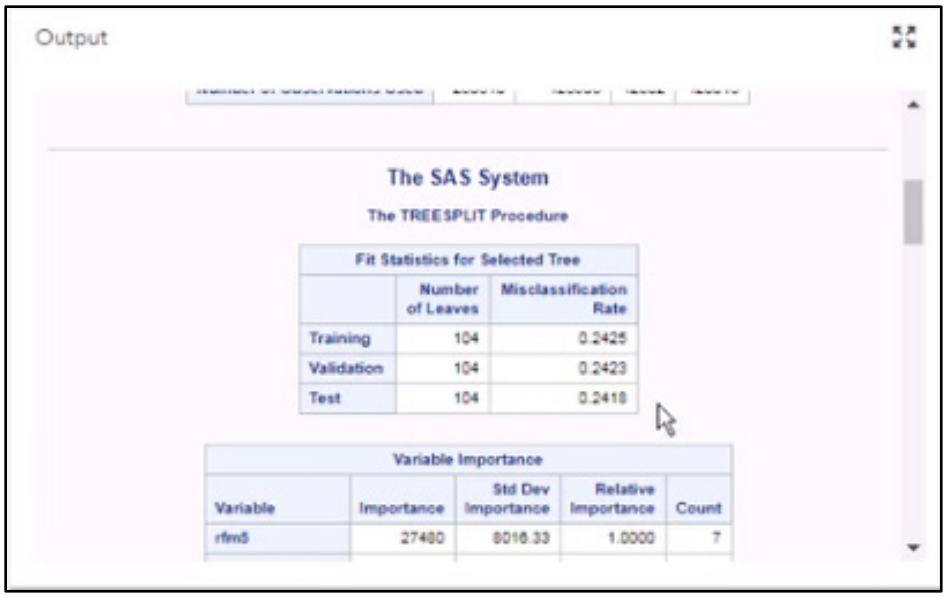


A Decision Tree node is added and connected to the Data node, and a Model Comparison node is added and connected to the Decision Tree node. Any other models you want to add to the project will automatically be compared with each other.

Results

Let's run the Decision Tree model with the default settings. Right-click the Decision Tree node and select Run. When the execution is done, right-click the Decision Tree node again and select Results. Let's scroll down in the results until we see the output for the TREESPLIT procedure (Figure 4.47).

Figure 4.47: Decision Tree Results



The misclassification rate for the test data is 0.2418. That means the model made wrong decisions in almost 25% of the cases. However, out of the 25% of misclassified cases, we don't know the proportion of false positives and false negatives.

The target b_tgt refers to the customers who purchased a new bank product (1) and the customers who did not purchase a new bank product (0). If the model classifies a customer as a 1 (likely to purchase) and the customer was observed as a 0 (did not purchase), the bank would waste money by sending an offer to that particular customer – a false positive. On the other hand, if the model classifies a customer as 0 (unlikely to purchase) and the customer was observed as 1 (actually did purchase), the bank would miss the opportunity to sell a new bank product to this customer – a false negative. This is illustrated in the confusion matrix in Table 4.1.

Table 4.1: Confusion Matrix

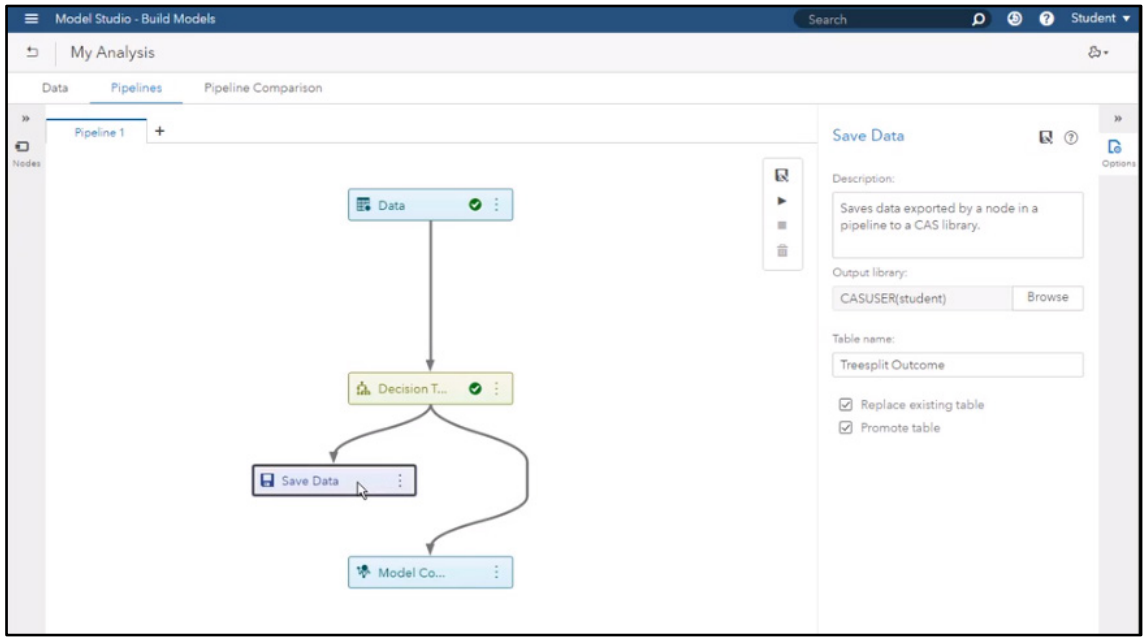
Classified	Observed	Result	Case
1	0	Bank wastes money by sending an offer to a customer who doesn't purchase	false positive
0	1	Bank misses the opportunity to sell a new product to this customer	false negative

Both false cases are important, but they have different financial impacts during a sales campaign. To evaluate the impact of all possible combinations in terms of predicted and observed cases (false positives, false negatives, true positives, and true negatives), we need to explore the final output in a different way.

Save Data Node

Let's save the output from the Decision Tree node so that we can use it in another application. Close the results of the decision tree model to return to the pipeline. Right-click the **Decision Tree** node, then select **Add Below ▶ Miscellaneous ▶ Save Data**. A Save Data node is added and connected to the Decision Tree node as shown in Figure 4.48. You can also add nodes by dragging and dropping them from the list in the left pane.

Figure 4.48: Modified Pipeline



Under the **Save Data** options pane on the right side of the screen, click **Browse** to choose the **Output library**. Expand cas-shared-default. Select CASUSER(student) and then click OK. This is the caslib that we are going to use to save the data. Later, we'll use this library to retrieve the data for further analysis.

In the **Table name property**, enter Treesplit Outcome as the new table name, then select the option for **Replace existing table**, which overwrites any existing table with that name. Also select **Promote table**, which enables this table to be seen and used in other sessions.

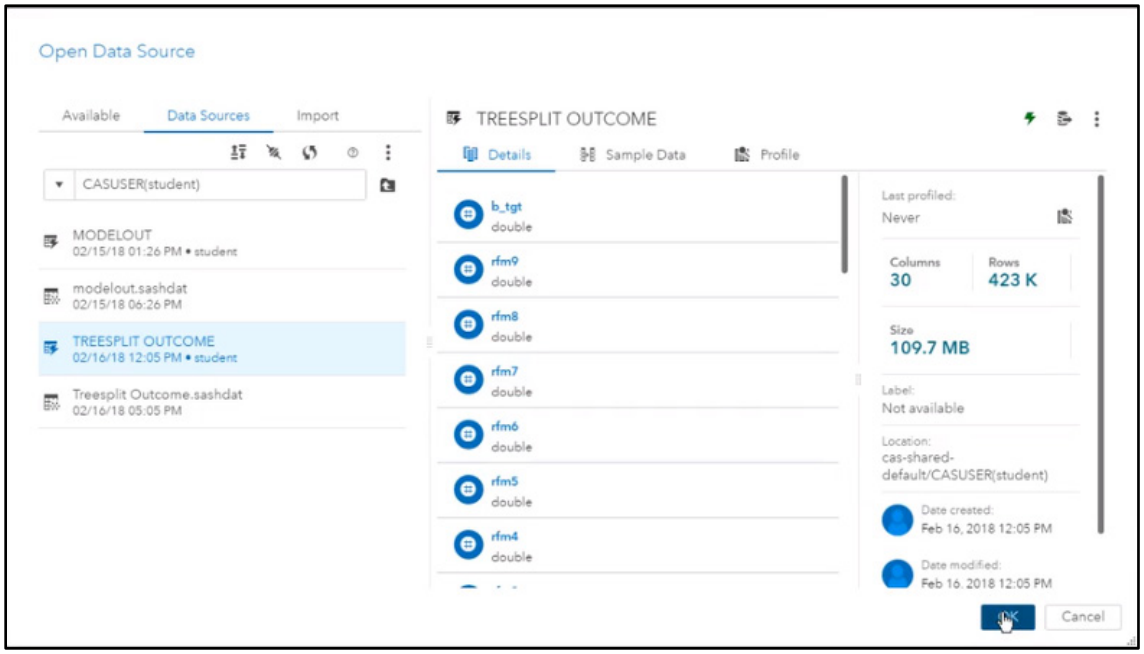
Right-click the **Save Data** node and select **Run**. Let's now look at the saved results data by using SAS Visual Analytics.

SAS Visual Analytics

From Model Studio, you can open Visual Analytics by clicking the applications menu in the top left corner. Select **Explore and Visualize Data**.

In the SAS Visual Analytics home screen, select **Data** from the far left panel. In the **Open Data Source** window, select the **Data Sources** tab. Expand cas-shared-default ▶ CASUSER(student) and then select the Treesplit Outcome data set, as shown in Figure 4.49. Click **OK**.

Figure 4.49: Open Data Source Window

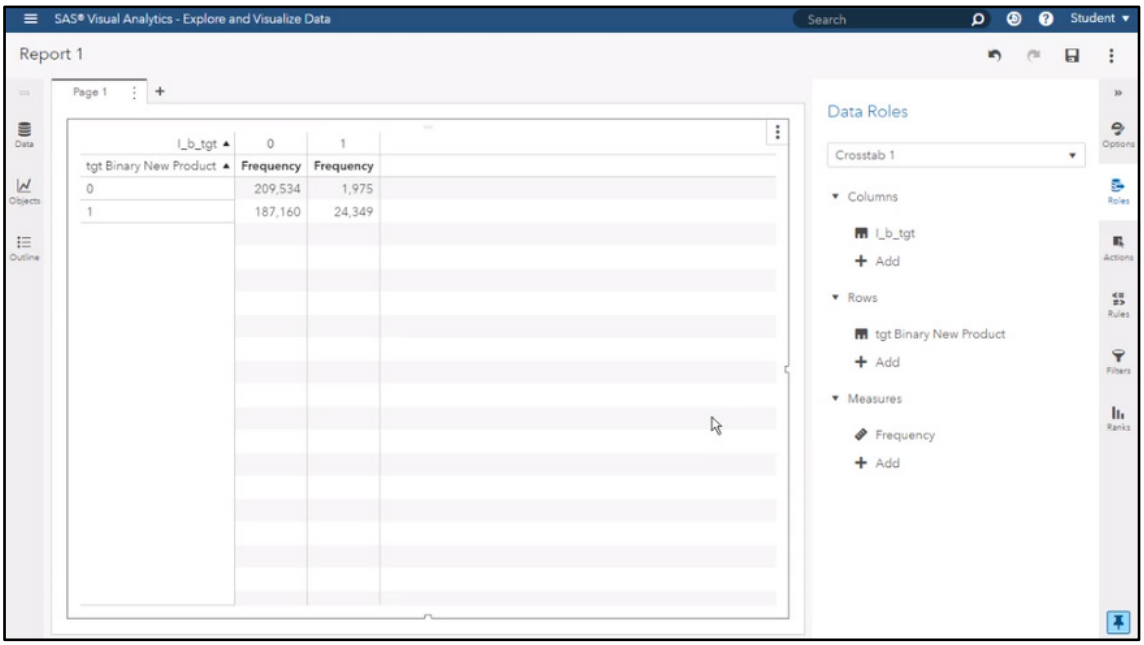


Now your data is loaded to memory and is ready to be explored. Notice the categorical variable l_b_tgt in the Data pane on the left side of the screen. This is the class predicted for each observation by the decision tree model. Scroll down to the end of the variable list find the observed class, tgt Binary New Product. This variable is numeric, so it's automatically defined as a measure.

To create a cross tabulated table to represent a confusion matrix, we need two categorical variables. Let's change the level of measurement for tgt Binary New Product from numeric to category. Click the button that has double downward arrows next to the variable. Select **Category** under the **Classification** property. The new variable now moves to the top of the list under Category.

In the left pane, click **Objects**. Under **Table**, double-click **Crosstab**. In the right pane, under Columns, click **Add**. Select l_b_tgt and then click **OK**. Under Rows, click **Add**. Select tgt Binary New Product and then click **OK**. The cross tabulated results are now ready, as shown in Figure 4.50.

Figure 4.50: Crosstab Table



The observed class is on the left and the predicted class is on the top. For instance, out of the 211,509 cases of non-purchase (0), the model correctly classified 99% (209,534) and misclassified only 1% (1,975).

Thinking about the confusion matrix, we only have 0.5% false positives out of the entire population. That means the bank would send a promotion to a small number of customers who would not be willing to purchase the product.

However, out of the 211,509 customers who actually did purchase, the model correctly classified only 12% (24,349). That means the model incorrectly classified 88% of the cases (187,160). Therefore, we have 44% false negatives based on the population data. Those cases are assigned to the missing opportunity. The bank would think that those customers were unlikely to purchase and would not add them to the campaign. Because of these results, we might consider building a different model in an attempt to improve performance.

In this section, you have seen how easy it is to save data in Model Studio and use model prediction results to perform additional analysis and data exploration in SAS Visual Analytics.

Resources

This chapter is based on the "Data Mining and Machine Learning on SAS Viya" videos in SAS® Viya® Enablement, a free course available from SAS Education.

You might find the following documentation and resources helpful as you learn more about SAS Visual Data Mining and Machine Learning in Model Studio:

- SAS® Visual Data Mining and Machine Learning 8.4: User's Guide
  Model Studio 8.4: Getting Oriented
- Link to download VS_BankData data set link: https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/bank

Chapter 5: SAS Visual Data Mining and Machine Learning in SAS Visual Analytics

Introduction

In this chapter, you will learn about key features of SAS Visual Data Mining and Machine Learning in SAS Visual Analytics. SAS Visual Data Mining and Machine Learning is an integrated add-on to SAS Visual Analytics that offers rich data mining and machine learning capabilities. This solution provides forward-looking insights with more information to support better decisions.

In this single, web-based environment, data scientists, citizen data scientists, and business analysts can prepare data, train models, and assess models. Multiple users have concurrent access to the same in-memory data with no conflicts. In this easy-to-use, drag-and-drop interface, you can interact with and analyze complex and large data sets visually, create predictive models without writing code, and see results instantly.

Predictive Models

The predictive models available in SAS Data Mining and Machine Learning in SAS Visual Analytics are:

- Factorization Machines
- Forest
- Gradient Boosting

- Neural Networks
- Support Vector Machines (SVM)

SAS Visual Data Mining and Machine Learning is designed to make model development easy and straightforward.

SAS Visual Analytics Quick-Start Guide

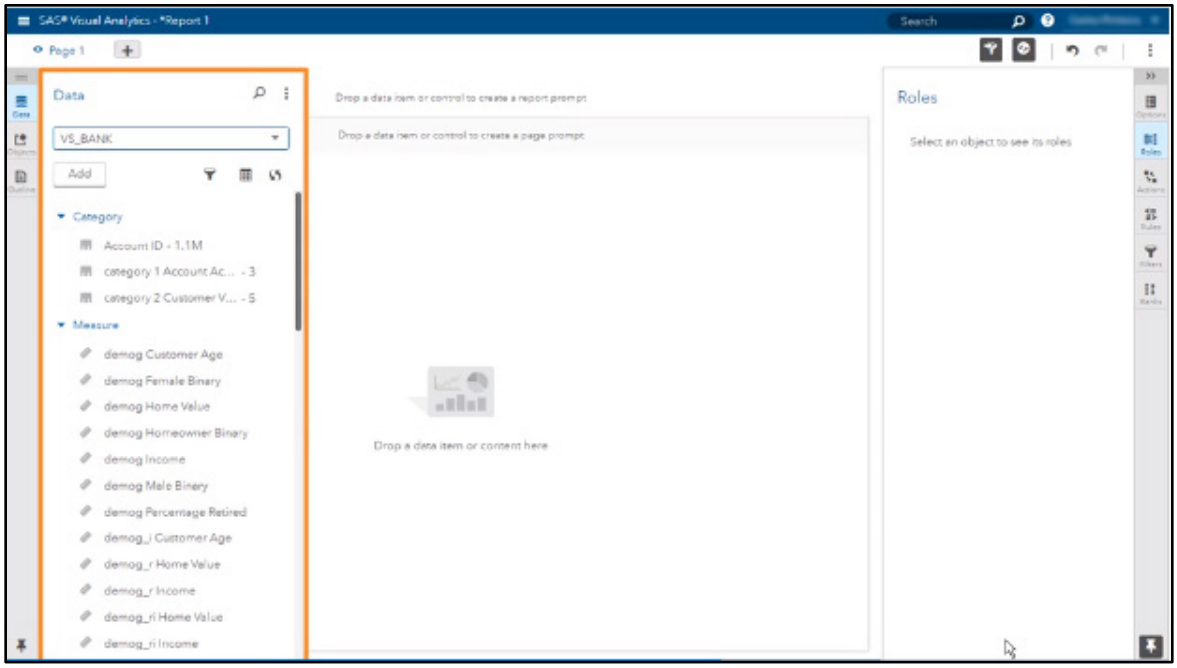
To begin, sign in to SAS. Click the **SAS Visual Analytics** tile on your home screen to access a single interface for designing reports and exploring data.

In the Welcome to **SAS Visual Analytics** window, click **Data**. In the **Open Data Source** window, make sure that the **All** option is selected. All the data sources available for analysis and model development are listed in this window. Find and select the data source that you want to use. Then click **OK**.

Data and Objects

In general, data-related tasks are initiated from the left-hand pane, also called the Data pane, as shown in Figure 5.1. In the Data pane, you can see all of the columns for the data source, grouped by categorical variables (under Category) and interval variables (under Measure).

Figure 5.1: Data Pane



In the far left pane, select **Objects**. You can see all of the different types of objects available in SAS Visual Analytics. These objects represent actions that you can perform for data preparation, data analysis, descriptive statistics, statistical tasks, and data mining and machine learning procedures.

Scroll down to find the SAS Visual Data Mining and Machine Learning group. You can see all of the models available, which we will discuss in this chapter. Using a particular predictive model against the selected data is quite easy. Just select a model and drag it into the workspace in the center pane. Some graphs are displayed along with an alert indicating that you need to select the target variable (or response) and the input variables (or predictors).

Roles and Options

In general, presentation-related tasks are initiated from the right-hand pane. Under **Roles**, you need to define the response and the predictors. Once you have defined these variables, the results of the model will be displayed in the workspace. After you have adjusted all of your options the way you would like, you can save your report.

You have now learned the basics of how to use SAS Visual Analytics to develop a predictive model using the SAS Visual Data Mining and Machine Learning capabilities. Let's take a deeper dive into each of the models in the rest of this chapter.

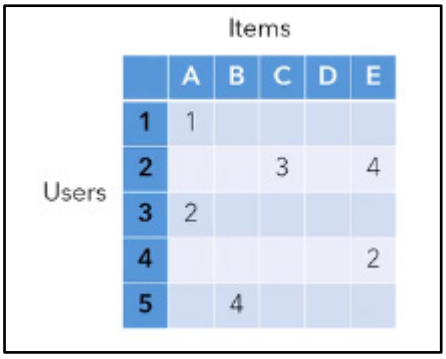
Factorization Machine

In this section, you will learn how to build a factorization machine using the SAS Visual Data Mining and Machine Learning feature in SAS Visual Analytics.

Overview

A factorization machine is a relatively new and powerful tool for modeling high-dimensional, sparse data. For example, consider a recommendation problem where users rate items that they have purchased. This data produces a matrix where the rows correspond to users and the columns correspond to items. There are many users and many items, so the matrix is large. However, the ratings can be very sparse because not all of the users give ratings for all of the items. As a consequence, the matrix can have several missing values, as shown in Figure 5.2.

Figure 5.2: Sparse Matrix



The main goal of factorization machines is to predict the missing entries in the matrix so that it is possible to find items that each user is likely to rate highly and then recommend those items to each user. Factorization machines model the ratings by summing the average rating over all users and items, the average ratings given by a user, the average ratings given to an item, and a pairwise interaction term that accounts for the affinity between a user and a particular item.

The affinity between users and items is modeled as the inner product between two vectors of features: one for the users, and another for the item. These features are collectively known as factors.

Data Source

The data source REVIEW contains information about ratings given by users to items. Both users and items are identified by an anonymous numeric ID. A rating is a numeric value between 0 (very bad) and 5 (very good). The data source contains 10,000,000 rows and 3 columns. There are 10,000 users and 10,000 items, so 10,000,000 reviews out of 100,000,000 are possible. The matrix produced by this data contains 90% missing values, so it is very sparse. A similar data set of movie ratings is available to download at http://grouplens.org/datasets/movielens.

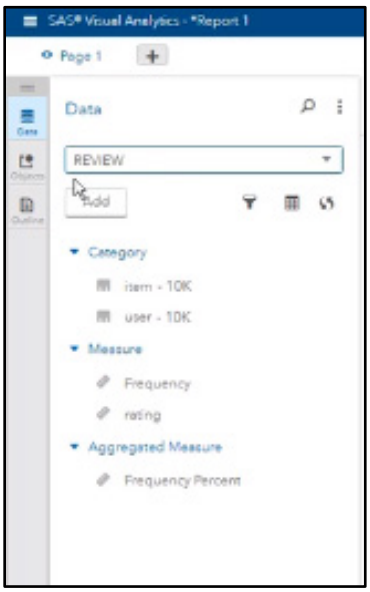
Build Model

To begin, sign in to SAS. Select the **Visual Analytics** tile on the home screen. In the **Welcome to SAS Visual Analytics** window, click **Data**. In the **Open Data Source** window, make sure that the **All** option is selected. Select the REVIEW data source (which has already been loaded into CAS) and then click OK.

In the **Data** pane, you can see all of the columns in the REVIEW data source, grouped by categorical variables (under Category), and interval variables (under Measure), as shown in Figure 5.3. Notice that item and user are

categorical variables, and each has 10,000 values. Rating and Frequency are interval variables or, in other words, measures.

Figure 5.3: Data Pane



In the left pane, select **Objects**. Scroll down to the SAS Visual Data Mining and Machine Learning group. To add a factorization machine to the diagram, you can either double-click **Factorization Machine** or drag and drop Factorization Machine onto the workspace. Once you have added the object, you can see three graphs with an alert indicating that it is time to configure the factorization machine model.

In the right pane, under **Roles**, you need to define the response variable (the target or dependent variable) and the predictors (the input or independent variables). Under **Response**, click Add. In the pop-up window, select the rating variable. Under **Predictors**, click Add. In the pop-up window, select the item and user variables. Click OK.

Notice that the model starts running. When it finishes, you can see the results of the factorization machine model at the top of the workspace, as shown in Figure 5.4.

Figure 5.4: Factorization Machine Model Results

The results display the target variable rating associated with the factorization machine; the average squared error (ASE) for the model, which is around 0.084; and the number of observations used during model development: 10,000,000.

On the left side of the workspace is the iteration plot based on the loss function. The iteration plot shows the loss function at each step of the optimization process for the factorization machine. During the optimization process, the task is to minimize the sum of losses over the observed data. In this example, the procedure computes the biases and the factors by using the stochastic gradient descent (SGD) method. In this method, each iteration attempts to minimize the root mean square error.

On the right side of the workspace are two assessment plots. On the top, the Scored Response plot shows the performance of the model by comparing the predicted values and the observed values. This plot is based on 20 bins by default. The plot shows the number of cases that fall in a particular combination of a predicted value range and an actual value range. On the bottom is the Assessment plot. This plot shows the overall performance of the model by comparing the predicted average to the observed average. In other words, this plot shows the difference between how the model estimates a case and what the case was observed to be in reality.

Modify Model

Let's try to improve the model's performance. In the far right pane, click **Options**. Under Factorization Machine, change **Factor count** to 20, **Max iteration count** to 50, and **Learn step** to .01.

Notice that when you change those options, the model starts running automatically. At the end of the run, you can see the model's performance at the top of the workspace, as shown in Figure 5.5.

Figure 5.5: Modified Factorization Machine Model Results

The final ASE for the new model is around 0.079, which is 6.1% better than the original one.

Forest

In this section, you will learn how to build a forest model using the SAS Visual Data Mining and Machine Learning feature in SAS Visual Analytics.

Overview

A forest consists of several different decision trees that differ from each other in two ways:

1. The training data for each tree is a sample without replacement from all available observations.
2. The input variables that are considered for splitting a node are randomly selected from all available inputs.

In other respects, trees in a forest are trained like standard trees. For an interval target, the procedure averages the predictions of the individual trees to predict an observation. For a categorical target, the posterior probabilities in the forest are the averages of the posterior probabilities of the individual trees.

Data Source

We will use a data source named VS_BANK_PARTITION for this example. The data source contains over one million rows and 57 columns. It consists of observations taken from account holders at a large financial services firm. The accounts represent consumers of home equity lines of credit, automobile loans, and other short- to medium-term credit instruments. This data source contains the original data in its raw form and additional inputs created during data preparation, which includes imputation, replacement, and log transformation.

The raw data describes both customer and account properties as well as some RFM metrics. The target variable indicates whether the account holder purchased a new product from the bank in the past year.

Build Model

To begin, sign in to SAS. Select the **Visual Analytics** tile on the home screen. In **the Welcome to SAS Visual Analytics** window, click **Data**. In the **Open Data Source** window, make sure that the **All** option is selected. Find and select the VS_BANK_PARTITION data source (which has already been loaded into CAS) and then click OK.

In the **Data** pane, you can see all of the columns in the VS_BANK_PARTITION data source, grouped by categorical variables (under Category), and interval variables (under Measure). To avoid overfitting the model, as a best practice, we train the model based on one data set (called the training partition) and validate the model based on another data set (called the validation partition).

For this example, we have already created a column named GroupDesc - 2 that splits the data source into training and validation partitions. For each observation, the value of GroupDesc - 2 is either Training or Validation.

Now, let's assign the observations to training or validation based on the value of GroupDesc - 2. Under Category, right-click GroupDesc - 2 and select **Set as partition column** from the menu. In the **Edit Partition Column** window, make sure that the Training data value is set to Training and the Validation data value is set to Validation. Click OK to confirm this assignment.

In the far left pane, select **Objects**. These objects correspond to the actions that you can perform for data preparation, data analysis, descriptive statistics, statistical tasks, and data mining and machine learning procedures. Find the SAS Visual Data Mining and Machine Learning group. To add a forest to the diagram, you can double-click **Forest** or drag and drop Forest onto the workspace. After you add the object to the workspace, you can see three graphs with an alert indicating that it is time to configure the forest model.

In the far right pane, click **Roles**. Under Roles, you need to define the response (the target or dependent variable) and the predictors (the input or independent variables). Under **Response**, click Add. In the pop-up window, select the variable tgt Binary New Product. Under **Predictors**, click **Add**. In the pop-up window, select the predictors from the list. In this example, there are 20 predictors. Click **OK**. Notice that the model starts running. When it finishes, you can see the results of the model as shown in Figure 5.6.

Figure 5.6: Forest Model Results

At the top of the workspace, the results display the target variable tgt Binary New Product; the average squared error for the model, which is around 0.121; and the number of observations used during model development, which is over a million.

On the left side of the workspace in Figure 5.6 is the variable importance table, which lists inputs that contribute most to the classification. You can see that the variable rfm5 Count Purchased Past 3 Years was the most important variable for this model. Six variables have a reasonable importance, and another six have a small importance.

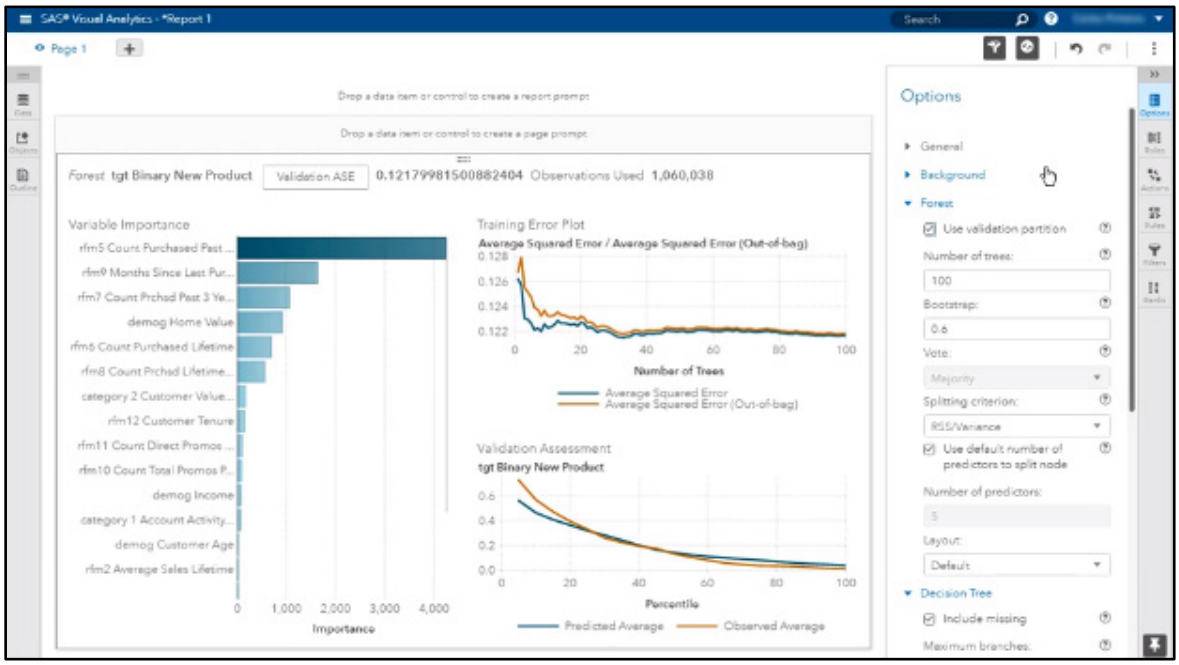
On the top right of the workspace is the error plot based on the average squared error and the average squared error (out-of-bag). The average squared error is the sum of squared errors on the final model divided by the number of observations. The average squared error (out-of-bag) is the mean of the average squared error for each tree during the training process. The forest ran based on 100 trees. We see the performance of the final model during training as more trees are added.

On the bottom right of the workspace is the assessment plot. The assessment shows the overall performance of the model, comparing the predicted average to the observed average. In other words, this plot shows the difference between how the model classified the case and how the case was observed in reality.

Modify Model

To improve model performance or make the model more robust or generalized (for example, to avoid overfitting), you can change some of the model parameters. In the far right pane, click **Options**. Under **Forest**, select **Use validation partition**. This option tells the model to use all of the cases where the GroupDesc -2 value is Training for the training process, and all of the cases where the GroupDesc - 2 value is Validation for the validation process. The model runs, and when it finishes, you can see the results as shown in Figure 5.7.

Figure 5.7: Modified Forest Model Results



Notice at the top of the workspace that the title now specifies Validation ASE, indicating that the model was assessed by the average squared error based on the validation data partition. The average squared error is slightly worse. Let's try to improve the model's performance a bit more.

Under **Forest**, change the **Number of trees** option to 200 and the **Bootstrap** option to 0.8. Under **Decision Tree**, change **Maximum levels** to 12, **Leaf size** to 25, and **Predictor bins** to 100. When you change these options, the model runs again. When it finishes running, you can see the results, as shown in Figure 5.8.

Figure 5.8: Further Modified Forest Model Results

The final average squared error for the new model is around 0.100, which is 17.4% better than the first model without any modifications.

Gradient Boosting

In this section, you will learn how to build a gradient boosting model for digital clickstream data using the SAS Visual Data Mining and Machine Learning feature in SAS Visual Analytics. In this example, the business use case is to identify important attributes that are related to whether a customer wants to download a white paper from a high-tech company's website.

Overview

In gradient boosting, the data set is resampled several times to generate results that form a weighted average of the resampled data. This model creates a series of decision trees that together form a single predictive model. A tree in the series is fit to the residuals of the predictions from the earlier trees in the series. The residuals are defined in terms of the derivative of a loss function. The successive samples are adjusted to accommodate previously computed inaccuracies.

Build Model

To begin, sign in to SAS. Select the **Visual Analytics** tile on the home screen. In the **Welcome to SAS Visual Analytics** window, click **Data**. In the **Open Data Source** window, make sure that the **All** option is selected. Find and select DIGITAL_CLICK_PART data source (which has already been loaded into CAS) and then click OK.

In the **Data** pane, you can see all of the columns in the DIGITAL_CLICK_PART data source, grouped by categorical variables (under Category), and interval variables (under Measure). Before we build our model, we are going to set our partition variable so that we can distinguish between training and validation sets. Scroll down to find the variable Partition Indicator. Right-click and select **Set as partition column** from the menu. In the **Edit Partition Column** window, make sure that the Training data value is set to Training and the Validation data value is set to Validation. Click **OK** to confirm this assignment.

Click **Objects** in the far left pane. Let's drag the **Gradient Boosting** model from the SAS Visual Data Mining and Machine Learning group into the visualization pane. Click **Role** in the far right pane. The target or response variable is going to be the white paper download indicator. This is a binary variable that tracks whether a visitor downloaded a white paper from the website. The values are yes or no. Under **Response**, click Add. Choose White Paper DLs ind from the list of variables.

Click **Options** in the far right pane. Under **Gradient Boosting**, select **Use validation partition**. Next, click **Data** in the far left pane. We are going to select certain attributes into the model. Use the search bar at the top of the data pane to search for the word "interest" because in our data, we have several variables that contain the word interest that we want to include in the model. They include cloud computing, database, Internet of Things, mobile and wireless, security, and storage. Click the check boxes next to these variable names to select the variables, as shown in Figure 5.9.

Figure 5.9: Adding Predictors from the Data Pane

With the variables selected, click and drag them into the visualization pane. The software will now begin to build a model for us. But we can still add more variables. Clear out the search in the data pane search bar. Let's also add Continent, as well as the duration of how long visitors were on our website (Active 10s to 30m, Active 10 to 30s, and so on) Click and drag these variables into the visualization pane as well.

Now you can see we have built a Gradient Boosting model as shown in Figure 5.10.

Figure 5.10: Gradient Boosting Model Results

But there are still more things that we need to change. Firstly, we need to model our event level as Yes. In the top row of the diagram, click in the rectangle labeled (event = no). From the drop-down, select **Change event level**. In the **Select Event Level** dialog box, change the radio button selected from no to yes. Click **OK** to close.

Next, we will look at the misclassification rate. Right now, the top row of the diagram indicates that our Validation Misclassification is 0.142. You can also look at the training misclassification. Click the rectangle

labeled **Validation Misclassification**. From the menu, select **Misclassification**. That number is 0.144. Reset the misclassification to show the Validation Misclassification.

Modify Model

Click **Options** in the far right pane. Under **Gradient Boosting**, we can see that we are only using 10 trees right now. Let's increase that number to 100. Hit enter. The diagram refreshes, and we can see significant improvement in the Validation Misclassification rate, which is now 0.133. You can also see from the Training Iteration Plot that the Misclassification Rate is dropping dramatically until we reach about 72 trees, as shown in Figure 5.11.

Figure 5.11: Modified Gradient Boosting Model Results

We can also change things like the **Learning Rate**, which is how quickly the model is learning. If you change that number for 0.1 to 0.2, we can see a little bit more improvement in the misclassification rate. It is now 0.1324.

We can also look at the **Subsample Rate**, which is the percent of data that is used to build each tree. Change that number from 0.8 to 0.6, and we can see an even better improvement in the misclassification rate. It is now 0.1319.

Scroll down in the Options pane to see the controls for the individual decision trees – 2 branch splits, a maximum number of 6 levels deep, and a minimum leaf size of 5. We'll leave those alone.

Now let's look at changing some variables. Click **Roles** in the far right pane to see the list of predictor variables. Let's remove the variable Continent. Right-click the variable name and select **Remove Continent** from the menu. When the model updates, we can see that it takes a bit of a hit, but not much. The misclassification rate is now 0.134.

If we wanted to, we could add Continent in as a filter instead. Click **Filters** in the far right pane. Click **Add.** Select Continent from the list of variables. Now we can see what each Continent's model might look like. Select only North America from the Continent list to see just those results.

In summary, we can see that the gradient boosting model helps to accurately identify the likelihood of downloading a white paper.

Export Model

Now, what do we do with this model? How do we take action on it? The first thing we can do is export the model to a table. In the upper right corner of the diagram workspace, click the double arrows (<<) and then click the three vertical dots to see a drop-down with more options. Choose **Export model…** from the list. In the **Export Model** window, click OK. This creates an Analytical Store table, which is a binary table that represents the entire model. We can take the intelligence of that model using that table and operationalize it to score new visitors. We can identify whether the visitor is interested in downloading a white paper, and we can deliver that information to our marketing peers.

You can also export the data into an Excel spreadsheet. In the upper right corner of the diagram workspace, click the double arrows (<<) and then click the three vertical dots to see the drop-down with more options. Choose **Export data…** from the list. In the **Export Data** window, click **OK**.

Neural Network

In this section, you will learn how to build a Neural Network model for digital clickstream data using the SAS Visual Data Mining and Machine Learning feature in SAS Visual Analytics. In this example, the business use case is to identify important attributes that are related to whether a customer wants to download a white paper or not from a high-tech company's website.

Overview

A neural network is a computational approach that was originally developed by researchers trying to mimic the neurophysiology of the human brain by combining many simple computing elements (neurons) into a highly interconnected system. Neural networks incorporate methods from statistics and numerical analysis into their training process.

Build Model

To begin, sign in to SAS. Select the **Visual Analytics** tile on the home screen. In the **Welcome to SAS Visual Analytics** window, click **Data**. In the **Open Data Source** window, make sure that the **All** option is selected. Find and select DIGITAL_CLICK_PART data source (which has already been loaded into CAS) and then click **OK**.

In the **Data** pane, you can see all of the columns in the DIGITAL_CLICK_PART data source, grouped by categorical variables (under Category), and interval variables (under Measure). Before we build our model, we are going to set our partition variable so that we can use training and validation sets when we model. Scroll down to find the variable Partition Indicator. Right-click and select **Set as partition column** from the menu. In the **Edit Partition Column** window, make sure that the Training data value is set to Training and the Validation data value is set to Validation. Click OK to confirm this assignment.

Click **Objects** in the far left pane. Look through the modeling library until you find **Neural Network** in the SAS Visual Data Mining and Machine Learning group. Drag the Neural Network object into the center Visualization pane.

Now we are ready to add our variables. The target or response variable of the neural network analysis is the white paper download indicator. Click **Roles** in the far right pane. Under **Response**, click **Add**. Find the variable White Paper DLs Ind and select it. This is a binary variable that tracks whether a visitor downloaded a white paper or not from the website. The values are yes and no.

Next, let's add certain attributes that we want to add into the analysis. These attributes or indicators are mainly related to what people were interested in when they visited the website. In the far left pane, click **Data**. In the search bar, type in the word "interest." We are quickly presented with all of the variables that have the word "interest" in their titles. These include cloud computing, database, Internet of Things, mobile and wireless, security, and storage. Click the check boxes next to all of these variables to select them, then drag them into the visualization pane.

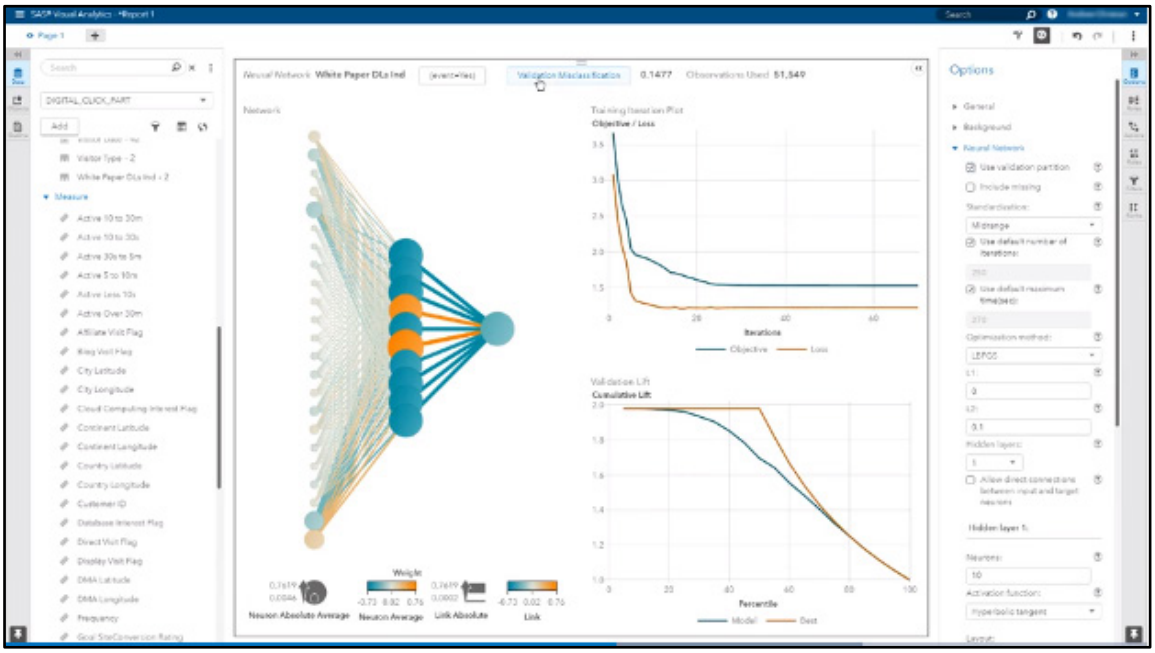
A neural network model quickly builds, but we are not done adding variables yet. Clear the search bar at the top of the Data pane. Click on the variable Continent to select it. Scroll down to the Measures group. We want to

choose variables related to the duration of the visit. We want to include all of the various time buckets. Click all those variables, then drag them into the visualization pane.

After the model finishes running, you can see it has built a neural network model for us. There are still a few more things we need to change. Right now, it is modeling on the event=No. We are going to change that to event=Yes. At the top of the diagram, click on the text **event=No**. From the menu, select **Change event level**. Click the radio button **Yes**, then click **OK**.

Next, click **Options** in the far right pane. Under **Neural Network**, select the option **Use validation partition**. The model now looks like Figure 5.12.

Figure 5.12: Neural Network Model Results



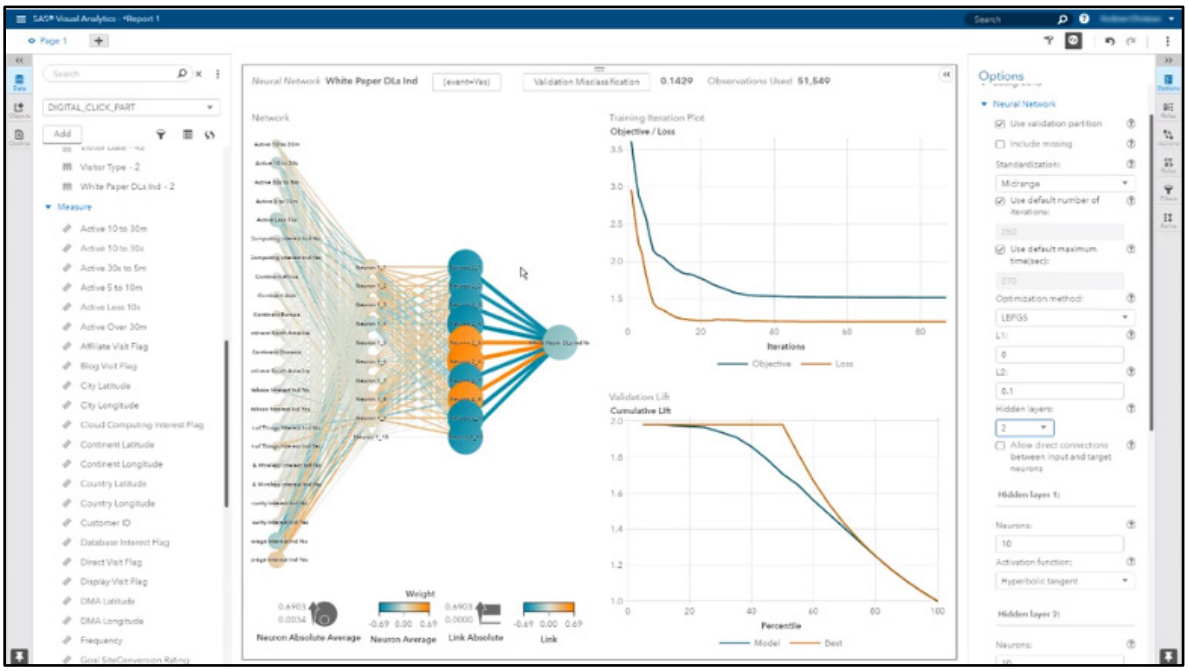
As you can see, we have a Validation Misclassification rate of 0.1477. Let's zoom in on the network and take a look. In the Options pane, scroll down to the Network Diagram section. Click **Neuron labels** to isolate hidden relationships.

The Iteration Plot indicates that after about 25 iterations, the objective or loss function flattens. For those who would like to go further in their analysis, there are several model assessment visualizations such as the Lift chart, ROC curve, and Misclassification chart.

Modify Model

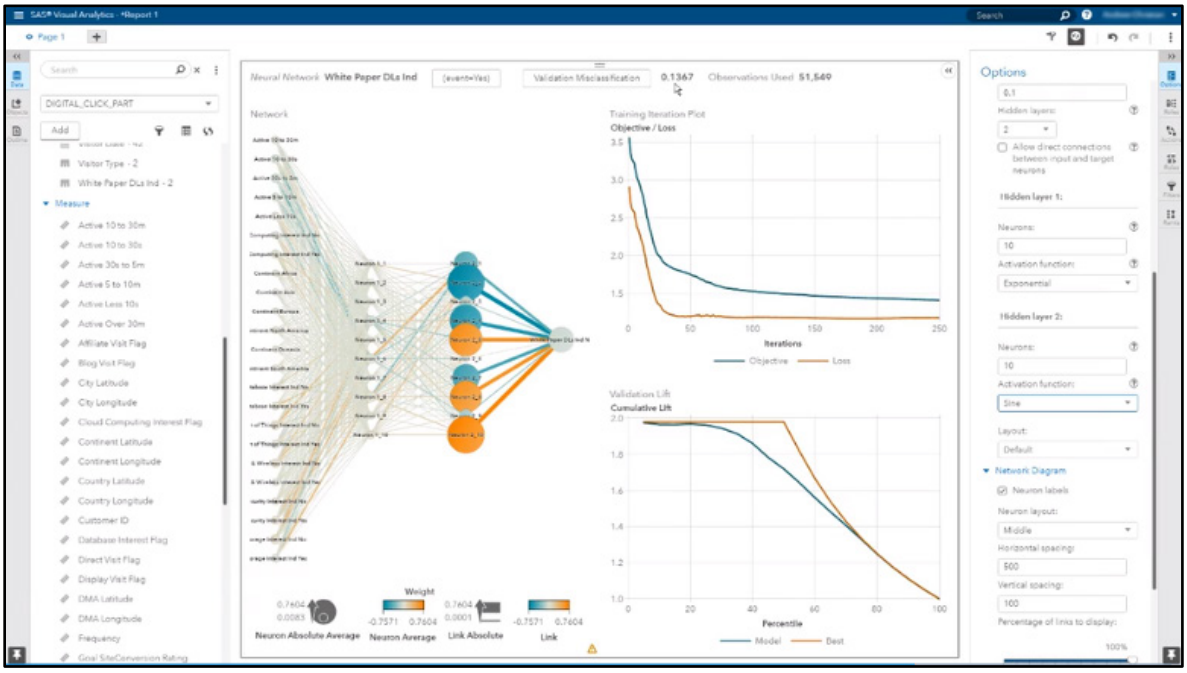
Let's go back to the Options pane and add another hidden layer. In the **Neural Network** section under **Hidden Layers**, change the number from 1 to 2. Notice how the results update automatically, and the misclassification rate improves slightly to 0.1429, as shown in Figure 5.13.

Figure 5.13: Modified Neural Network Model Results



In the **Options** pane, you can also change the optimization method and adjust your regularization parameters, L1 and L2. You can also change the **Activation functions by layer**. In this case, for the first layer, let's change the **Activation function** property from Hyperbolic tangent to Exponential. For the second layer, let's change it from Hyperbolic tangent to Sine. The diagram updates as shown in Figure 5.14.

Figure 5.14: Further Modified Neural Network Model Results



Notice that the Validation misclassification has dropped to 0.1367. This tells us that this neural network model accurately identified the likelihood to download a white paper.

Export Model

If we want to download the model, in the upper right corner of the diagram workspace, click the double arrows (<<) and then click the three vertical dots to see a drop-down with more options. Choose **Export model…** from the list. When we export the model, this produces SAS DATA step code representing the entire neural network model. We can take the intelligence of this neural network model and operationalize it to score new visitors. We can identify whether the visitor is interested in downloading a white paper, and we can deliver that information to our marketing peers.

You can also export the data as an Excel spreadsheet. In the upper right corner of the diagram workspace, click the double arrows (<<) and then click the three vertical dots to see the drop-down with more options. Choose **Export data…** from the list. In the **Export Data** window, click **OK**.

Support Vector Machine

In this section, you will learn how to build a support vector machine model by using the SAS Visual Data Mining and Machine Learning feature in SAS Visual Analytics.

Overview

A support vector machine is a supervised machine-learning method that is used to perform a classification analysis based on slack variables, which create a soft margin classifier. The standard support vector machine model solves binary classification problems that produce a binary output (positive 1 or -1) by constructing a set of hyperplanes that maximize the margin between the two classes. Posterior probabilities of the event can also be estimated.

Data Source

We will use a data source named VS_BANK_PARTITION, which contains observations taken from account holders at a large financial services firm. The accounts represent consumers of home equity lines of credit, automobile loans, and other short-to-medium term credit instruments. This data source contains the original data in its raw form and additional inputs created during data preparation, which includes imputation, replacement, and log transformations.

The raw data describes both customer and account properties and includes recency, frequency, and monetary (RFM) metrics. The target variable indicates whether the account holder purchased a new product from the bank in the past year. The data source contains over a million rows and 57 columns. The original VS_BANK_DATA data set is available to download from GitHub here: https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/bank.

Build Model

To begin, sign in to SAS. Select the **Visual Analytics** tile on the home screen. In the **Welcome to SAS Visual Analytics** window, click **Data**. In the **Open Data Source** window, make sure that the **All** option is selected. Find and select the VS_BANK_PARTITION data source (which has already been loaded into CAS) and then click **OK**.

In the **Data** pane, you can see all of the columns in the VS_BANK_PARTITION data source, grouped by categorical variables (under Category), and interval variables (under Measure). To avoid overfitting the model, as a best practice, we train the model based on one data set (called the training partition) and validate the model based on another data set (called the validation partition).

For this example, we have already created a column named GroupDesc - 2 that splits the data source into training and validation partitions. For each observation, the value of GroupDesc - 2 is either Training or Validation.

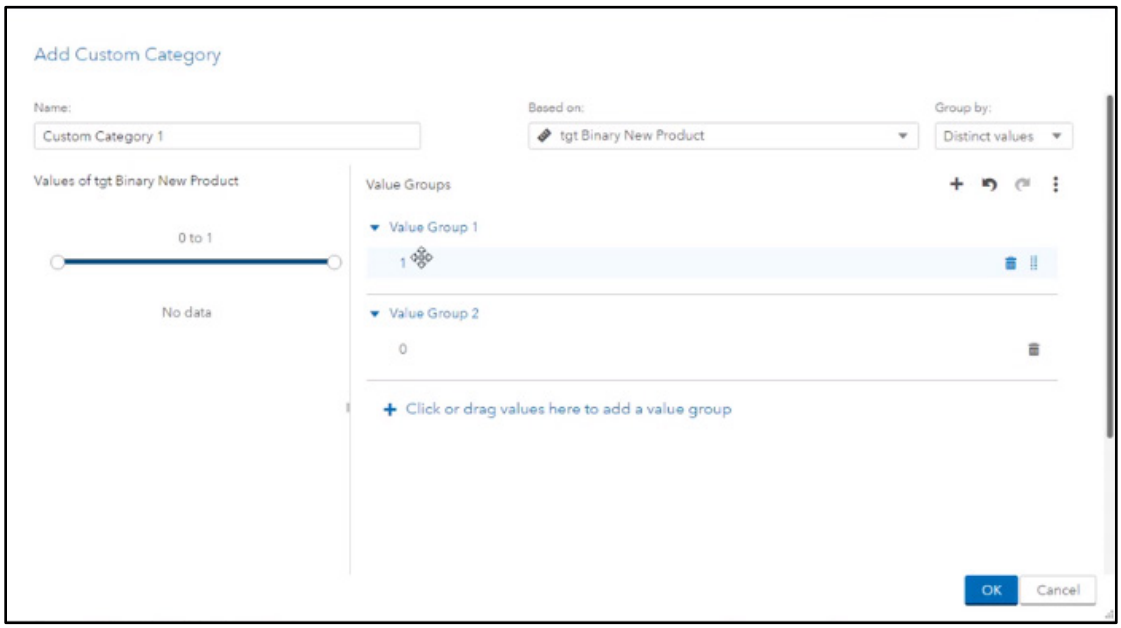
Now, let's assign the observations to training or validation based on the value of GroupDesc - 2. Under **Category**, right-click GroupDesc - 2 and select **Set as partition column** from the menu. In the **Edit Partition**

Column window, make sure that the Training data value is set to Training and the Validation data value is set to Validation. Click **OK** to confirm this assignment.

Support vector machines (SVMs) are classifier models. Based on a set of training cases for a binary target, the SVM algorithm builds a model that assigns new cases to one category or the other. An SVM is, therefore, a non-probabilistic binary linear classifier. The target in the VS_BANK_PARTITION data source is numeric, and it has two possible values. SVM models require categorical targets, so you need to convert the binary target to a categorical target.

In the Data pane, below the VS_BANK_PARTITION data source drop-down, click the **Add** button, and then select **Add Custom Category**. At the top of the **Add Custom Category** window, for **Based on**, select the variable tgt Binary New Product. This target has two values: 0 for non-buyers and 1 for buyers. Therefore, you need two categories to represent the target. Select **+ Click or drag values here to add a value group**. You will now see that a Value Group 2 is available. From the left side of the window, select 0 and drag it under the property **Value Group 2**. Select 1 on the left and drag it under Value Group 1. Your screen should now look like Figure 5.15.

Figure 5:15: Add Custom Category Window



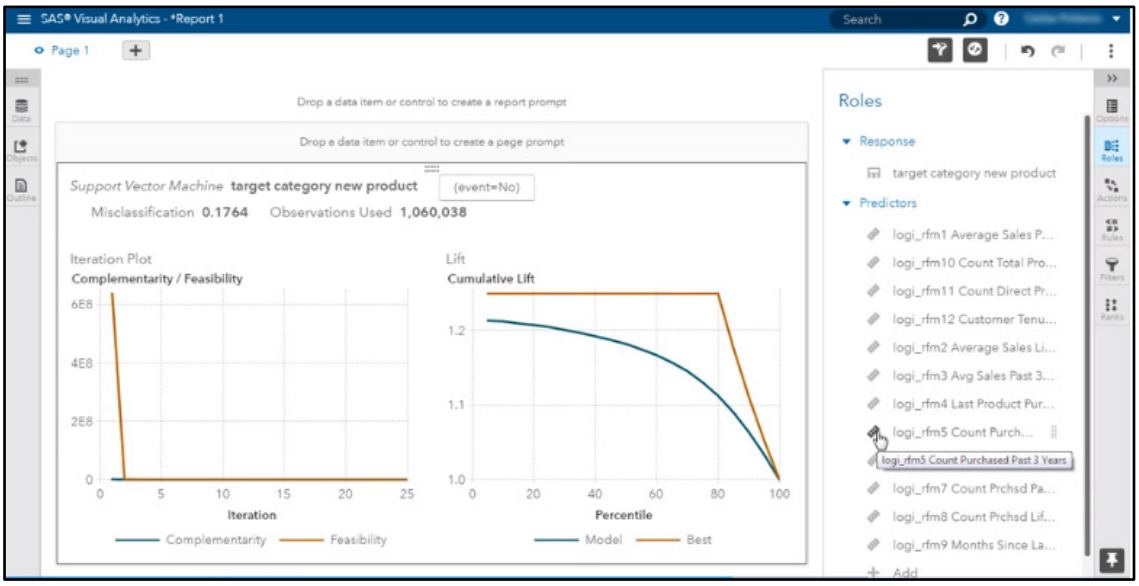
Let's change the names of the groups. Right-click Value Group 1 and select **Edit group name.** Enter "Yes" for the **Name** property and click OK. Right-click Value Group 2 and select **Edit group name**. Enter "No" for the **Name** property and click **OK**.

Let's also change the name of the category. In the top left corner of the window, under **Name**, enter "target category new product" and click **OK** in the bottom right of the **Add Custom Category Window** to close the window.

In the far left pane, select **Objects**. These objects correspond to the actions that you can perform for data preparation, data analysis, descriptive statistics, statistical tasks, and data mining and machine learning procedures. Find the SAS Visual Data Mining and Machine Learning group. To add a support vector machine to the diagram, you can double-click **Support Vector Machine** or drag and drop it onto the workspace. After you add the object to the workspace, you can see graphs and plots and an alert indicating that it is time to configure the support vector machine model.

In the far right pane, click **Roles**. Under Roles, you need to define the response (the target or dependent variable) and the predictors (the input or independent variables). Under **Response**, click **Add**. In the pop-up window, select the variable tgt category new product. Under **Predictors**, click **Add**. In the pop-up window, select the 12 variables from the list that have a logi_rfm prefix. Click **OK**. Notice that the model starts running. When it finishes, you can see the results of the support vector machine model as shown in Figure 5.16.

Figure 5.16: Support Vector Machine Model Results



The results of the support vector machine model are shown at the top of the workspace. You can see the target variable tgt category new product; the target level of interest (which was set to No by default due to alphabetical ordering); the misclassification rate, which is around 0.1764; and the number of observations used during model development, over one million.

On the left side of the workspace in Figure 5.16 is the Iteration Plot based on Complementarity and Feasibility. Complementarity conditions are optimization constraints required for optimal nonlinear programming solutions. The feasibility is a gap tolerance to check for optimization convergence. Slack variables are added to allow for errors to the objective function, and the approximate the model if the problem is infeasible.

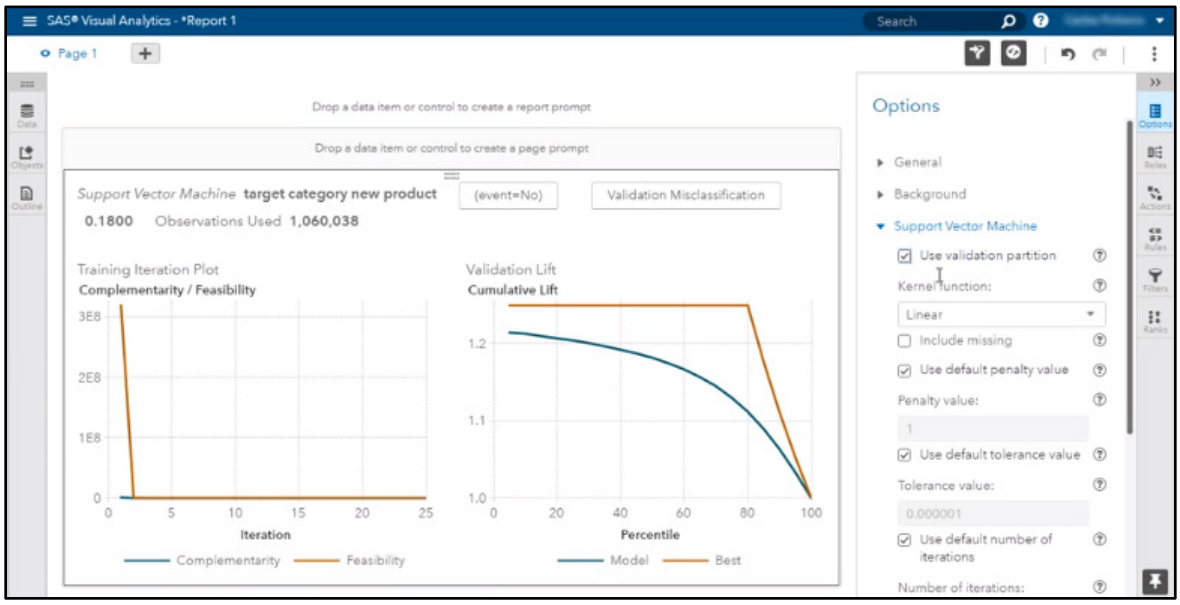
On the right side of the workspace in Figure 5.16 is the Cumulative Lift plot. The cumulative lift shows the cumulative ratio of the percentage of captured responses within each decile. The percentage captured response is the proportion of total responders in each decile. In this case, the cumulative lift is shown against the optimal model, the best possible model instead of the baseline.

Modify Model

To improve model performance or to make the model more robust or generalized (for example, to avoid overfitting), you can change some of the model parameters. In the right pane, select **Options**. Under the **Support Machine Vector** section, select **Use validation partition.** This option tells the model to use all of the cases where the GroupDesc - 2 value is Training for the training process, and all of the cases where GroupDesc – 2 is Validation for the validation process.

The model updates, and when it finishes, you can see the final results in the workspace, as shown in Figure 5.17.

Figure 5.17: Modified Support Vector Machine Model

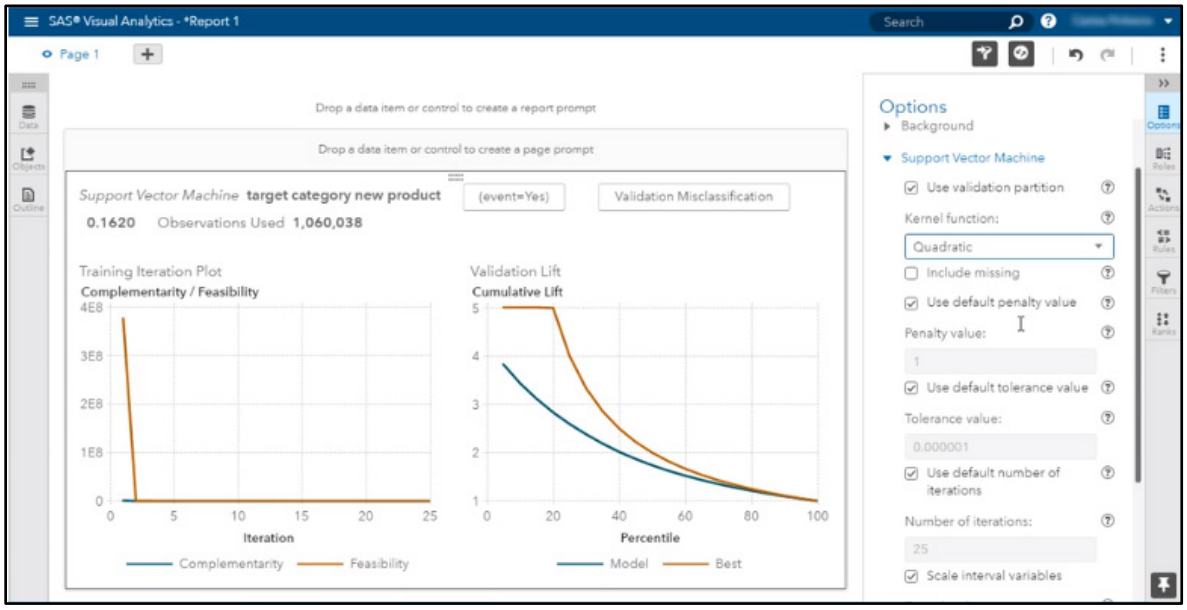


At the top of the workspace, notice that the title now indicates that the model was assessed by the misclassification rate based on the validation data partition. The misclassification rate has changed a bit. It is now 0.1800.

You can also change the event level of the target for the classification model. In the right pane, in the **Support Vector Machine** section, scroll down to find the **Event level** option. Click **Change.** In the pop-up list of levels for the categorical target, select Yes, and click **OK** to close the window. The model runs again. The model's performance is exactly the same with a validation misclassification rate of 0.1800.

Let's try to improve the model's performance further. In the right pane, in the **Support Vector Machine** section, find the **Kernel function** option. Change it from Linear to Quadratic. After you confirm this option, the model begins to run. When it finishes, you can see the results in the workspace, as shown in Figure 5.18.

Figure 5.18: Further Modified Support Vector Machine Model



At the top of the workspace, you can see the final results of the model. The final validation misclassification rate is 0.1620, which is almost 10% better than the initial SVM model.

Model Comparison

In this section, we will look at how to build a model comparison using SAS Visual Data Mining and Machine Learning in SAS Visual Analytics. In previous sections, we built a Gradient Boosting model and a Neural Network to predict the likelihood of someone downloading a white paper from a high-tech company's website. Now, let's determine which of those models predicts more accurately.

To get started, make sure that the Gradient Boosting and Neural Network models are each open in a separate report. Open a new page in a SAS Visual Analytics report. Click **Objects**. Under the SAS Visual Statistics section, click **Model Comparison** and drag it into the workspace. Notice that an error message pops up on the screen. The message says there are no models in the report that can be compared. We're going to have to fix something!

Go back to the Gradient Boosting model page of the report. The first thing you should notice is that the event level is not the same as the event level for the Neural Network model. In order to successfully perform a model comparison, the data source, partition, response variable, and event level for all models must match. At the top of the workspace, click on the button labeled **(event=No)**. In the dialog box, change the event level to Yes.

Return to your new report page. Try again to drag the Model Comparison object into the workspace. This time it should work. The **Add Model Comparison** window will appear, as shown in Figure 5.19.

Figure 5.19: Add Model Comparison Window

You can see that we have the same data source, the same partition, the same response variable, and the same event level. Below that, you can see all of the models that are available for comparison. Select both of them, and then click **OK**.

The software immediately starts to run a comparison and produces a comparative Lift chart and Fit Statistic bar chart, as shown in Figure 5.20.

Figure 5.20: Model Comparison Results – Misclassification Rate

In the **Options** pane on the right side of the screen, you can select from a number of **Model Comparison statistics.** Currently, it is set for Misclassification. Click the drop-down menu and select K-S Statistic instead. According to this statistic, it appears the Gradient Boosting model is going to be our champion.

Notice that you can also select a percentile to compare. Use the slider to move the **Percentile** to 20. At the 20th percentile, the gradient boosting model is still the best model. At the 35th percentile, it's still the winner. At the 50th it's still the winner, and so on.

Resources

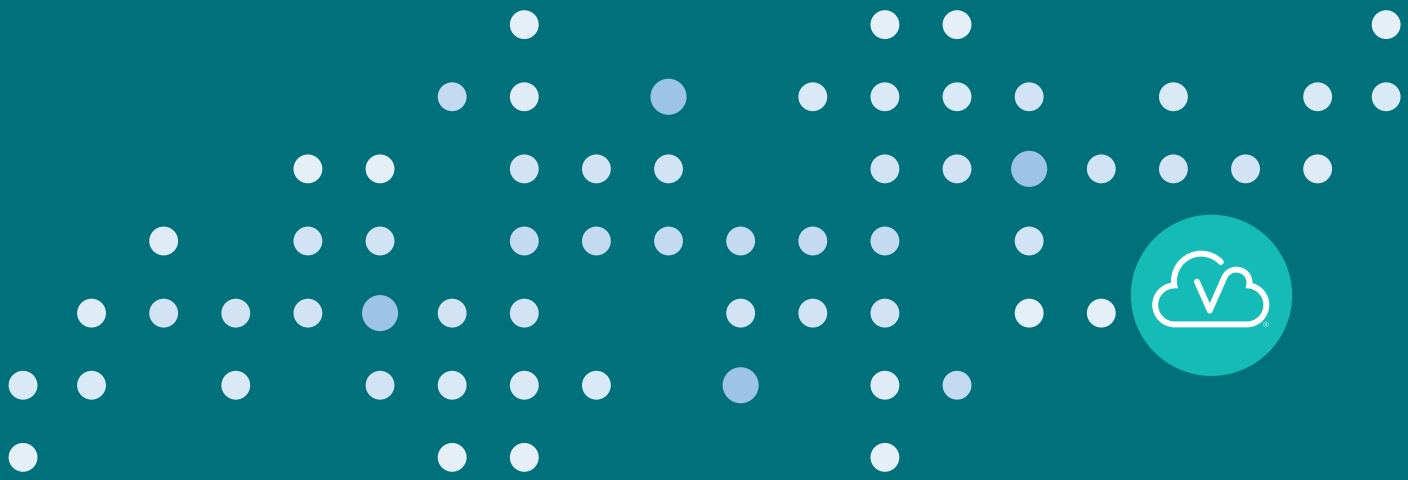
This chapter is based on the SAS Visual Data Mining and Machine Learning videos from the free, online How-To Tutorials provided by SAS at video.sas.com/category/videos/how-to-tutorials.

You might find the following documentation and resources helpful as you learn more about SAS Visual Data Mining and Machine Learning in SAS Visual Analytics:

- SAS® Visual Analytics 8.3: Working with SAS® Visual Data Mining and Machine Learning
- Link to download VS_BankData data set link: https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data/bank
- Link to download *MovieLens* data set: http://grouplens.org/datasets/movielens

Free SAS® Viya® e-Books: Fundamentals

This series is based on content from SAS® Viya® Enablement, a free course available from SAS Education. You can follow along with the examples in real time by watching the videos if you prefer. Topics covered illustrate the features and capabilities of SAS Viya. SAS Viya extends the SAS platform to enable everyone – data scientists, business analysts, developers, and executives alike – to collaborate and realize innovative results faster.



support.sas.com/freesasebooks

sas.com/books
for additional books and resources.

sas
THE POWER TO KNOW®