Getting Started with SAS® Visual Data Mining and Machine Learning 8.2

**Audience**

Data scientists, statisticians, data miners, engineers, researchers, and scientists, who need to analyze large and complex data and build predictive models using a modern, powerful, and customizable in-memory programming language.

**What is Machine Learning?**

Machine learning is a method of data analysis that automates 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.

**Benefits of Using Machine Learning**

- Data scientists can run more model-building scenarios in the same environment, which improves their productivity.
- Data scientists can build sophisticated machine learning pipelines with far less code because the integrated modern machine learning capabilities are available in a higher-level programming language.
- Data science teams can collaborate more easily through a web-based interface and a common machine learning platform.
Workflow for Machine Learning

1 Identify your data sets, load the data sets into SAS Cloud Analytic Services, and perform any data manipulation.

2 Explore the data by creating graphs that visualize any anticipated relationships, unanticipated trends, and anomalies in order to gain understanding and ideas.

3 Prepare the data for model building by splitting the data into training, validation, and test data.

4 Model the data by using the analytical techniques to search for a combination of the data that reliably predicts a desired outcome.

5 Assess the data by evaluating the usefulness and reliability of the findings from the data mining process.

6 Score new data. Assess and monitor the effectiveness of the models.

Access Data, Create Formats, Create Input Data Set

Where Can I Get the Data?
The case study begins with two SAS data sets that are available from the following location in GitHub:

https://github.com/sasssoftware/sas-viya-programming/tree/master/data

The two data sets are named census.sas7bdat and crimes.sas7bdat. Later in the case study, we will show how to join the two data sets to integrate the data as follows:

- The crimes data set identifies the offenses and includes a community area value that identifies the vicinity of the offense.
- The community area value maps to the census data set that contains six socioeconomic indicators of public health, a “hardship index,” and per capita income.
Data Provenance

City of Chicago Crime Data (2001 to present)
The data set that is used in the examples is a subset of the original data set that is available from the City of Chicago. The date range for the sample is January 1, 2001, to February 15, 2016.
The original data is available from https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2.

City of Chicago Census Data (2008 to 2012)
The data set is available from https://data.cityofchicago.org/Health-Human-Services/Census-Data-Selected-socioeconomic-indicators-in-C/kn9c-c2s2.

Disclaimer: SAS may reference other websites or content or resources for use at Customer’s sole discretion. SAS has no control over any websites or resources that are provided by companies or persons other than SAS. Customer acknowledges and agrees that SAS is not responsible for the availability or use of any such external sites or resources, and does not endorse any advertising, products, or other materials on or available from such websites or resources. Customer acknowledges and agrees that SAS is not liable for any loss or damage that may be incurred by Customer or its end users as a result of the availability or use of those external sites or resources, or as a result of any reliance placed by Customer or its end users on the completeness, accuracy, or existence of any advertising, products, or other materials on, or available from, such websites or resources.

Very First Steps

1 If you have not already, download the two data sets. Put them in a directory that is accessible to the SAS Workspace Server.

2 Sign in to SAS Studio.
   a Use the Server Files and Folders section to navigate to the directory that contains the two data sets.
   b Right-click on the directory and select Create → Library. Specify chicago as the name.

3 Use the Libraries section to confirm that the new library is listed.

4 Check that you have a CAS session.
   a Enter the following line in the code editor and click to run the code.

   cas casauto list;

   If you get the error: "ERROR: Request failed. Session CASAUTO not recognized", check with your system administrator to make sure that the server host and port are specified in a configuration file. An alternative is to specify the CASHOST= and CASPORT= system options.
   b Click the Log tab. A note like the following indicates that you have a CAS session.

   NOTE: Session CASAUTO is ACTIVE using port 5570 and host cloud.example.com for user sasdemo.
   The session UUID is 2861d3a4-779b-674c-bb46-31f46aa1daf0.

Create a Format

One of the frequently used variables in the crimes data set is the Fbi_code column. The codes in this column correspond to the uniform crime reporting codes as described at http://gis.chicagopolice.org/clearmap_crime_sums/crime_types.html.
Instead of using the codes such as "06" in displays to indicate larceny, we will create a SAS format that maps the code to the more descriptive text, Larceny. Enter the following code in the Code tab and then click.

Note: The CASFMTLIB= option is introduced in the SAS Viya 3.1 release. It is required to add the format to SAS Cloud Analytic Services.

```
proc format casfmtlib="chicagofmts" sessref=casauto;
  value $fbi
    '01A' = 'Homicide 1st & 2nd Degree'
    '02' = 'Criminal Sexual Assault'
    '03' = 'Robbery'
    '04A' = 'Aggravated Assault'
    '04B' = 'Aggravated Battery'
    '05' = 'Burglary'
    '06' = 'Larceny'
    '07' = 'Motor Vehicle Theft'
    '08A' = 'Simple Assault'
    '08B' = 'Simple Battery'
    '09' = 'Arson'
    '10' = 'Forgery & Counterfeiting'
    '11' = 'Fraud'
    '12' = 'Embezzlement'
    '13' = 'Stolen Property'
    '14' = 'Vandalism'
    '15' = 'Weapons Violation'
    '16' = 'Prostitution'
    '17' = 'Criminal Sexual Abuse'
    '18' = 'Drug Abuse'
    '19' = 'Gambling'
    '20' = 'Offenses Against Family'
    '22' = 'Liquor License'
    '24' = 'Disorderly Conduct'
    '26' = 'Misc Non-Index Offense'
  ;
  value suntosat
    1 = 'Sunday'
    2 = 'Monday'
    3 = 'Tuesday'
    4 = 'Wednesday'
    5 = 'Thursday'
    6 = 'Friday'
    7 = 'Saturday';
run;
```

**Load the Data Into SAS Cloud Analytic Services**

Enter the following on the Code tab and then click.

```
proc casutil;
  load data=chicago.census;
  load data=chicago.crime;
  contents casdata="crime";
  contents casdata="census";
quit;
```

If you did not assign the name *chicago* when you created the SAS library, substitute the value that you used.
Merge the Data Sets

To get a more complete picture of each crime, the census data is joined to the crime table.

TIP The / sessref="casauto" DATA statement option ensures that the DATA step runs on the server. Otherwise, the code can download the data to the client for execution, which reduces performance.

libname mycas cas sessref=casauto;

data mycas.crimeCensus/ sessref=casauto;
merge mycas.crime(in=in1) mycas.census(in=in2);
by community_area;
if in1;
run;

proc casutil;
   contents casdata="crimeCensus";
quit;

Output 1  Table and Column Information for the crimeCensus Table

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NLS encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Repeated Table</th>
<th>View</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRIMECENSU</td>
<td>105951</td>
<td>51</td>
<td>utf-8</td>
<td>30Jun2016 13:04:41</td>
<td>30Jun2016 13:04:41</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Explore Data Using ODS Graphics

Before performing data analysis, it is important to explore and survey the data. With larger data sets, that is done by summarizing the data and then either printing the summary table or creating a plot.
Plot the Overall Arrest Rate

/* overall arrest rate */
ods graphics / width=5in antialiasmax=5600;
proc sgplot data=mycas.crimeCensus;
title "Overall Arrest Rate";
vbar arrest;
run;

Plot the Crime and Arrest Rate by Offense

This plot helps visualize that some types of crime, such as Larceny and Simple Battery, occur much more often than other types of crime.

ods graphics / width=8in antialiasmax=5600;
proc sgplot data=mycas.crimeCensus;
vbar fbi_code / categoryorder=respdesc group=arrest;
xaxis display=(nolabel);
run;
To see the arrest rate as a percentage of crimes for the top 10 crimes, you can run the following code:

```sas
proc mdssummary data=mycas.crimeCensus;
  var arrest_code;
  groupby fbi_code;
  output out=mycas.crimeGrouped;
run;

proc sql outobs=10;
  select fbi_code as "FBI Code",
         _Nobs_ as Crimes,
         _Sum_ as Arrests,
         (_Sum_ / _Nobs_) * 100 format=5.2 as Pct
  from mycas.crimeGrouped
  order by Crimes desc;
quit;
```

Because the MDSUMMARY procedure processes numeric variables only, the arrest_code variable that is coded as numeric 0 and 1 is used. The rows are displayed below:
**Output 3**  Top Ten Crimes and Arrest Rate as a Percentage

<table>
<thead>
<tr>
<th>FBI Code</th>
<th>Crimes</th>
<th>Arrests</th>
<th>Pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larceny</td>
<td>21433</td>
<td>2792</td>
<td>13.03</td>
</tr>
<tr>
<td>Simple Battery</td>
<td>16593</td>
<td>4083</td>
<td>24.61</td>
</tr>
<tr>
<td>Vandalism</td>
<td>12049</td>
<td>894</td>
<td>7.42</td>
</tr>
<tr>
<td>Drug Abuse</td>
<td>11440</td>
<td>11369</td>
<td>99.38</td>
</tr>
<tr>
<td>Misc Non-Index Offense</td>
<td>11105</td>
<td>4508</td>
<td>40.59</td>
</tr>
<tr>
<td>Burglary</td>
<td>6163</td>
<td>411</td>
<td>6.67</td>
</tr>
<tr>
<td>Motor Vehicle Theft</td>
<td>5001</td>
<td>486</td>
<td>9.72</td>
</tr>
<tr>
<td>Simple Assault</td>
<td>4897</td>
<td>1049</td>
<td>21.42</td>
</tr>
<tr>
<td>Robbery</td>
<td>3991</td>
<td>418</td>
<td>10.47</td>
</tr>
<tr>
<td>Fraud</td>
<td>3029</td>
<td>654</td>
<td>21.59</td>
</tr>
</tbody>
</table>

**Plot Larceny and Battery by Hour**

The two largest categories are Larceny and Simple Battery. Is there any relationship between time of day and day of the week and having the crime result in an arrest?

```plaintext
data mycas.larBatByHr / sessref=casauto;
  set mycas.crimeCensus;
  format dow suntosat.;
  if fbi_code in ('06', '08B');
  dow=weekday(date); /* returns 1=Sunday... */
  h=hour(timestamp);
run;

proc sort data=mycas.larBatByHr out=work.sorted;
  by dow;
run;

proc sgplot data=work.sorted;
  heatmap x=dow y=h / colorresponse=arrest_code discretex ;
  yaxis label="Hour of Day";
  xaxis discreteorder=data display=(nolabel);
run;
```
Plot Crime by per Capita Income

The documentation for the census data set indicates the following national averages for a number of statistics. According to United States Census Bureau 2008-2012 American Community Survey 5-year estimates:

- 3.2% of occupied housing units in the US had more than one person per room
- 10.9% of households were living below the federal poverty level
- 9.3% of persons aged 16 years or older in the labor force were unemployed
- 14.2% of persons aged 25 years or older did not have a high school diploma
- 37.2% of the population was under 18 or more than 64 years of age
- per capita income was $28,051

```
proc sgplot data=mycas.crimeCensus;
    title 'Crime incidence by Per Capita Income';
    histogram per_capita_income;
    reline 28051 / axis=x label='U.S. Average';
    xaxis label='Per Capita Income';
run;
```

The plot below shows a cluster of reported crime at and below the national average.
Plot Crime and Arrest over Time

By plotting a time series of the number of reported crimes and arrests, we can see an annual trend and a seasonal trend. We can also see how the number of reported crimes is trending down in recent years compared to the early 2000s.

```plaintext
proc mdsummary data=mycas.crimeCensus;
   groupby date;
   var arrest_code;
   output out=mycas.ts;
run;

proc sgplot data=mycas.ts;
   title "Times Series of Crime and Arrest";
   series x=date y=_sum_/ legendlabel='Arrests';
   series x=date y=_nobs_/ legendlabel='Reported Crimes';
run;
```
Output 6  Time Series Plot of Reported Crimes and Arrests

![Time Series Plot of Crime and Arrests](image)

**Plot Crime Incidents by Hardship Index**

The kernel density estimate shows an increase in reported crime in areas with a hardship index that exceeds 50.

```sas
proc sgplot data=mycas.crimeCensus;
  title 'Percentage of Crime by Hardship Index';
  histogram hardship_index;
  density hardship_index / type=kernel;
run;
```
Perform Data Prep

Split the Data into Training, Validation, and Test

The **PARTITION** procedure is used to perform stratified sampling.

```sas
proc partition data=mycas.crimeCensus partind seed=9878 samppct=30 samppct2=10;
    target arrest;
    output out=mycas.cwpart copyvars=_all_;
run;
```

- This **PARTIND** option adds a partition indicator column to the output table. The column has values of 0, 1, or 2 that correspond to the training, validation, and testing partitions.
- The **SEED=** option specifies 9878 as the random seed to be used in the partitioning process.
- The **TARGET** statement causes the procedure to perform stratified sampling and maintain the ratio of arrest in each partition.
- The **SAMPPCT=** option requests that 30% of the input data be included in the validation partition (1).
- The **SAMPPCT2=** option requests that 10% of the input data be included in the testing partition (2).
- The balance, 60%, is used for the training partition (0).
- The **OUTPUT** statement requests that the sampled data be stored in a table named mycas.cwpart and the **COPYVARS=** option specifies that all variables be copied from mycas.crimeCensus to mycas.cwpart.
The table mycas.cwpart is used as the input table for creating models in the section Create the Models on page 14.

Output 7  Stratified Sampling Results

The PARTITION Procedure

<table>
<thead>
<tr>
<th>Index</th>
<th>Arrest</th>
<th>Number of Obs</th>
<th>Sample Size 1</th>
<th>Sample Size 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>false</td>
<td>74283</td>
<td>22285</td>
<td>7428</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
<td>31668</td>
<td>9500</td>
<td>3167</td>
</tr>
</tbody>
</table>

Output CAS Tables

<table>
<thead>
<tr>
<th>CAS Library</th>
<th>Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASUSERHDFS</td>
<td>CWPART</td>
<td>105951</td>
<td>32</td>
</tr>
</tbody>
</table>

Check the Proportion of the Sampling

The MDSUMMARY procedure creates basic descriptive statistics. We want to verify that the arrest variable was stratified across the three partitions.

```
proc mdsummary data=mycas.cwpart;
  groupby _partind_
  var arrest_code;
  output out=mycas.split;
run;
proc print data=mycas.split;
run;
```

By looking at the _Mean_ column we can verify that all three rows have a value of about 0.29. We can also verify the percentage of observations in each partition by looking at the _NObs_ column.

Output 8  Partial Results for the Mycas.split Table

```
Obs  _Partind_  _Partind_f_  _Column_  _Min_  _Max_  _NObs_  _NMiss_  _Mean_  _Sum_  _Std_  _StdErr_
1    2    2        arrest_code  0    1    10595   0      0.298945624 3167  0.457893627 0.0044476339
2    0    0        arrest_code  0    1    63571   0      0.2988941948 19001 0.457769473 0.0018156177
3    1    1        arrest_code  0    1    31785   0      0.298883121 9500  0.4577576027 0.0025676843
```

Explore the Cardinality

The CARDINALITY procedure determines a variable’s cardinality (the number of levels the variable has) or limited cardinality in SAS Viya 3.1. You can view the results of the CARDINALITY procedure to decide which variables to use as classification or interval variables for model building.

```
proc cardinality data=mycas.cwpart outcard=mycas.card maxlevels=20;
  var _numeric_;
  var _char_;
run;

proc print data=mycas.card;
run;
```
Create the Models

Now you can build candidate models based on the training and validation partition. Model building is also available inside the SAS Visual Data Mining and Machine Learning add-on to SAS Visual Analytics.

Create Macro Variables

It is helpful to set up macro variables for repetitive code. In this example, we create macros for the commonly used user-supplied values.

```
%let dset=mycas.CWpart;
%let outdir=.;
%let target=arrest_code;
%let nom_input=fbi_code location_description domestic beat district ward community_area;
%let int_input=percent: per_capita_income hardship_index;
```

- The DSET macro variable represents the input table, mycas.CWpart. Input data must be an in-memory table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref and the second level must be the table name. In this example, mycas. is the CAS engine libref, so we begin the input table with mycas.

- The OUTDIR macro variable is used to specify an operating system directory for the SAS scoring code that analytic procedures create. The directory must be accessible to the SAS Workspace Server.

- The TARGET macro variable represents the variable whose values the procedures try to predict. Arrest_code was originally the character variable Arrest in the input data. We transformed it to a numeric variable.

- The NOM_INPUT and INT_INPUT macro variables are used to specify the columns to treat as nominal and interval values, respectively. Using a macro variable can speed your development time because the columns are specified in one place and referenced multiple times.

Create a Predictive Model

The FOREST procedure creates a predictive model by averaging many decision trees in SAS Viya. The purpose of a predictive model is to predict a target value from inputs. The following statements train a forest model and score the training data table.

```
proc forest data=&dset. ntrees=10 minleafsize=5 seed=1859355710 outmodel=mycas.model_forest ;
  target &target. / level=nominal;
  input &nom_input. / level=nominal;
  input &int_input. / level=interval;
  partition rolevar=_partind_(train='0' validate='1');
  output out=mycas.ap_scored_forest copyvars=(_partind_ &target);
  title "Random Forest";
run;
```
The TARGET, INPUT, PARTITION, and OUTPUT statements perform the same basic function in all of the procedures in this section.

- The TARGET statement names the variable whose values PROC FOREST tries to predict.
- The INPUT statements specify the nominal and interval input variables. We determined the interval and nominal variables from the PROC CARDINALITY results.
- The PARTITION statement assigns roles to the observations in the input table based on the value of the variable _partind_ in that table. Observations whose value is 0 are assigned for training, and observations with a value of 1 are used for validation.
- The OUTPUT statement creates a new data table mycas.ap_scored_forest that is the result of prediction from using the input data and the model. PROC ASSESS uses this table as the input table in Assess the Models.
### The FOREST Procedure

**Model Information**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
<td>10</td>
</tr>
<tr>
<td>Number of Variables Per Split</td>
<td>4</td>
</tr>
<tr>
<td>Seed</td>
<td>1859355710</td>
</tr>
<tr>
<td>Bootstrap Percentage</td>
<td>60</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>20</td>
</tr>
<tr>
<td>Number of Input Variables</td>
<td>14</td>
</tr>
<tr>
<td>Maximum Number of Tree Nodes</td>
<td>1671</td>
</tr>
<tr>
<td>Minimum Number of Tree Nodes</td>
<td>921</td>
</tr>
<tr>
<td>Maximum Number of Branches</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Number of Branches</td>
<td>2</td>
</tr>
<tr>
<td>Maximum Depth</td>
<td>20</td>
</tr>
<tr>
<td>Minimum Depth</td>
<td>20</td>
</tr>
<tr>
<td>Maximum Number of Leaves</td>
<td>836</td>
</tr>
<tr>
<td>Minimum Number of Leaves</td>
<td>461</td>
</tr>
<tr>
<td>Maximum Leaf Size</td>
<td>9464</td>
</tr>
<tr>
<td>Minimum Leaf Size</td>
<td>5</td>
</tr>
<tr>
<td>OOB Misclassification Rate</td>
<td>0.1438464</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Validation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>63571</td>
<td>31705</td>
<td>95356</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>63571</td>
<td>31705</td>
<td>95356</td>
</tr>
</tbody>
</table>

### Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbi_code</td>
<td>6415.27</td>
<td>448.05</td>
</tr>
<tr>
<td>Location_Description</td>
<td>981.91</td>
<td>26.6718</td>
</tr>
<tr>
<td>Beat</td>
<td>244.61</td>
<td>3.1743</td>
</tr>
<tr>
<td>District</td>
<td>144.82</td>
<td>12.0133</td>
</tr>
<tr>
<td>Ward</td>
<td>117.48</td>
<td>2.0418</td>
</tr>
<tr>
<td>Domestic</td>
<td>114.76</td>
<td>23.5779</td>
</tr>
<tr>
<td>Community_Area</td>
<td>89.1332</td>
<td>4.4366</td>
</tr>
<tr>
<td>percent_household_renter</td>
<td>88.89</td>
<td>7.1264</td>
</tr>
</tbody>
</table>

### Fit Statistics

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>OOB Average Error</th>
<th>Training Average Error</th>
<th>Validation Average Error</th>
<th>OOB Misclassification Rate</th>
<th>Training Misclassification Rate</th>
<th>Validation Misclassification Rate</th>
<th>OOB Log Loss</th>
<th>Training Log Loss</th>
<th>Validation Log Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.124</td>
<td>0.1109</td>
<td>0.124</td>
<td>0.156</td>
<td>0.143</td>
<td>0.156</td>
<td>0.058</td>
<td>0.610</td>
<td>0.872</td>
</tr>
<tr>
<td>2</td>
<td>0.119</td>
<td>0.1030</td>
<td>0.114</td>
<td>0.152</td>
<td>0.136</td>
<td>0.150</td>
<td>0.680</td>
<td>0.359</td>
<td>0.444</td>
</tr>
<tr>
<td>3</td>
<td>0.116</td>
<td>0.1018</td>
<td>0.112</td>
<td>0.149</td>
<td>0.135</td>
<td>0.145</td>
<td>0.553</td>
<td>0.338</td>
<td>0.391</td>
</tr>
<tr>
<td>4</td>
<td>0.114</td>
<td>0.1010</td>
<td>0.111</td>
<td>0.147</td>
<td>0.136</td>
<td>0.143</td>
<td>0.493</td>
<td>0.332</td>
<td>0.373</td>
</tr>
<tr>
<td>5</td>
<td>0.112</td>
<td>0.1011</td>
<td>0.110</td>
<td>0.146</td>
<td>0.134</td>
<td>0.143</td>
<td>0.441</td>
<td>0.330</td>
<td>0.366</td>
</tr>
<tr>
<td>6</td>
<td>0.112</td>
<td>0.1010</td>
<td>0.110</td>
<td>0.146</td>
<td>0.135</td>
<td>0.142</td>
<td>0.415</td>
<td>0.330</td>
<td>0.364</td>
</tr>
<tr>
<td>7</td>
<td>0.111</td>
<td>0.0998</td>
<td>0.109</td>
<td>0.145</td>
<td>0.133</td>
<td>0.141</td>
<td>0.401</td>
<td>0.326</td>
<td>0.361</td>
</tr>
<tr>
<td>8</td>
<td>0.111</td>
<td>0.1000</td>
<td>0.109</td>
<td>0.145</td>
<td>0.134</td>
<td>0.141</td>
<td>0.388</td>
<td>0.326</td>
<td>0.360</td>
</tr>
<tr>
<td>9</td>
<td>0.111</td>
<td>0.1002</td>
<td>0.109</td>
<td>0.144</td>
<td>0.134</td>
<td>0.142</td>
<td>0.379</td>
<td>0.327</td>
<td>0.360</td>
</tr>
<tr>
<td>10</td>
<td>0.110</td>
<td>0.1001</td>
<td>0.109</td>
<td>0.144</td>
<td>0.134</td>
<td>0.141</td>
<td>0.373</td>
<td>0.327</td>
<td>0.358</td>
</tr>
</tbody>
</table>
Create Decision Trees with Gradient Boosting

The GRADBOOST procedure creates a series of decision trees that together form a single predictive model. The output table mycas.ap_scored_gradboost is used by PROC ASSESS in the section Assess the Models.

```
proc gradboost data=&dset. maxdepth=8 minleafsize=5 seed=9878 outmodel=mycas.model_gradboost;
  target &target. / level=nominal;
  input &nom_input. / level=nominal;
  input &int_input. / level=interval;
  partition rolevar=_partind_(train='0' validate='1');
  output out=mycas.ap_scored_gradboost copyvars=(_partind_ &target.);
  title "Gradient Boost";
run;
```
### The GRADBOOST Procedure

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
</tr>
<tr>
<td>Learning Rate</td>
</tr>
<tr>
<td>Subsampling Rate</td>
</tr>
<tr>
<td>Number of Variables Per Split</td>
</tr>
<tr>
<td>Number of Bins</td>
</tr>
<tr>
<td>Number of Input Variables</td>
</tr>
<tr>
<td>Maximum Number of Tree Nodes</td>
</tr>
<tr>
<td>Minimum Number of Tree Nodes</td>
</tr>
<tr>
<td>Maximum Number of Branches</td>
</tr>
<tr>
<td>Minimum Number of Branches</td>
</tr>
<tr>
<td>Maximum Depth</td>
</tr>
<tr>
<td>Minimum Depth</td>
</tr>
<tr>
<td>Maximum Number of Leaves</td>
</tr>
<tr>
<td>Minimum Number of Leaves</td>
</tr>
<tr>
<td>Maximum Leaf Size</td>
</tr>
<tr>
<td>Minimum Leaf Size</td>
</tr>
<tr>
<td>Seed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>63571</td>
<td>31785</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>63571</td>
<td>31785</td>
</tr>
</tbody>
</table>

### Variable Importance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbi_code</td>
<td>446.76</td>
<td>146.79</td>
</tr>
<tr>
<td>Beat</td>
<td>318.26</td>
<td>8.8548</td>
</tr>
<tr>
<td>Location_Description</td>
<td>174.32</td>
<td>15.6598</td>
</tr>
<tr>
<td>Ward</td>
<td>69.8432</td>
<td>2.0081</td>
</tr>
<tr>
<td>Community_Area</td>
<td>64.1946</td>
<td>2.2089</td>
</tr>
<tr>
<td>Domestic</td>
<td>4.2084</td>
<td>3.6915</td>
</tr>
<tr>
<td>District</td>
<td>2.4016</td>
<td>0.7272</td>
</tr>
<tr>
<td>percent_aged_under_18_or_over_64</td>
<td>0.2541</td>
<td>0.4971</td>
</tr>
<tr>
<td>hardship_index</td>
<td>0.1784</td>
<td>0.8767</td>
</tr>
<tr>
<td>percent_of_housing_crowded</td>
<td>0.1778</td>
<td>0.4194</td>
</tr>
<tr>
<td>percent_aged_16_unemployed</td>
<td>0.1059</td>
<td>0.3569</td>
</tr>
<tr>
<td>percent_households_below_poverty</td>
<td>0.1019</td>
<td>0.4947</td>
</tr>
<tr>
<td>percent_aged_25_without_high_sch</td>
<td>0.07159</td>
<td>0.4093</td>
</tr>
<tr>
<td>per_capita_income</td>
<td>0.04658</td>
<td>0.5935</td>
</tr>
</tbody>
</table>
Perform Logistic Regression with Variable Selection

The LOGSELECT procedure fits binary and binomial response models.

```sas
proc logselect data=&dset. noclprint;
   class &target. &nom_input.;
   model &target.(event='1') = &nom_input. &int_input.;
   selection method=stepwise (choose=validate) ;
   partition rolevar=_partind_(train='0' validate='1');
   code file="&outdir./logselect1.sas";
   title "Logistic Regression";
run;
```

/* The SAS log can include notes for operations on missing values. */
data mycas.ap_scored_logistic;
   set &dset.;
   %include "&outdir./logselect1.sas";
   p_&target.1=p_&target.;
   p_&target.0=1-p_&target.;
run;

- The CLASS statement lists the nominal variables.
- The MODEL statement defines the statistical model. The MODEL statement and the CLASS statement together specify that observations with the formatted value of 1 represent events in the data. The probability that the LOGSELECT procedure models is the probability that the variable arrest_code takes on the (formatted) value of 1.
- The SELECTION statement specifies that the model selection is stepwise regression using validation data.
- The CODE statement writes SAS DATA step code and saves it in the file logselect1.sas. An in-memory DATA step is used with the scoring code.
The LOGSELECT Procedure

Model Information

<table>
<thead>
<tr>
<th>Description</th>
<th>CWSAPT</th>
<th>arrest_code</th>
<th>Binary</th>
<th>Logit</th>
<th>Newton-Raphson with Ridging</th>
</tr>
</thead>
</table>

Number of Observations

<table>
<thead>
<tr>
<th>Description</th>
<th>Total</th>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations Read</td>
<td>105951</td>
<td>63571</td>
<td>37105</td>
</tr>
<tr>
<td>Number of Observations Used</td>
<td>86537</td>
<td>67076</td>
<td>28461</td>
</tr>
</tbody>
</table>

Response Profile

<table>
<thead>
<tr>
<th>Ordered Value</th>
<th>arrest_code</th>
<th>Total Frequency</th>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>60101</td>
<td>40120</td>
<td>19981</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>25436</td>
<td>16956</td>
<td>8480</td>
</tr>
</tbody>
</table>

Probability modeled is arrest_code = 1.

Selection Information

<table>
<thead>
<tr>
<th>Selection Method</th>
<th>Stepwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Criterion</td>
<td>SBC</td>
</tr>
<tr>
<td>Choose Criterion</td>
<td>Validation ASE</td>
</tr>
<tr>
<td>Stop Criterion</td>
<td>SBC</td>
</tr>
<tr>
<td>Effect Hierarchy Enforced</td>
<td>None</td>
</tr>
<tr>
<td>Stop Horizon</td>
<td>3</td>
</tr>
</tbody>
</table>

Selection Details

Convergence criterion (ABSGCONV=1E-7) satisfied.

Selection Summary

<table>
<thead>
<tr>
<th>Step</th>
<th>Effect Entered</th>
<th>Number Effects In</th>
<th>SBC</th>
<th>Validation ASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Intercept</td>
<td>1</td>
<td>69457.3498</td>
<td>0.2092</td>
</tr>
<tr>
<td>1</td>
<td>fbi_code</td>
<td>2</td>
<td>45793.2014</td>
<td>0.1251</td>
</tr>
<tr>
<td>2</td>
<td>Location_Description</td>
<td>3</td>
<td>41282.659*</td>
<td>0.1136*</td>
</tr>
</tbody>
</table>

* Optimal Value Of Criterion

Stepwise selection stopped because adding or removing an effect does not improve the SBC criterion.

The model at step 2 is selected where Validation ASE is 0.113634.

Selected Effects: Intercept fbi_code Location_Description
Build a Decision Tree

The TREESPLIT procedure is used to build a classification tree.

```
proc treesplit data=&dset. minleafsize=5 outmodel=mycas.model_treesplit;
  target &target. /level=nominal;
  input &nom_input. /level=nominal;
  input &int_input. /level=interval;
  partition rolevar=_partind_(train='0' validate='1');
  output out=mycas.ap_scored_treesplit copyvars=(_partind_ &target);
  title "Decision Tree";
run;
```

The TREESPLIT procedure displays a visualization for the first few splits. The visualization is followed by the fit statistics and variable importance.
Output 13  Treesplit Procedure Results: Subtree Visualization For The First Three Splits
The TREESPLIT Procedure

**Fit Statistics for Selected Tree**

<table>
<thead>
<tr>
<th>Number of Leaves</th>
<th>Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>18</td>
</tr>
<tr>
<td>Validation</td>
<td>18</td>
</tr>
</tbody>
</table>

**Variable Importance**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Std Dev Importance</th>
<th>Relative Importance</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrest_code</td>
<td>19976</td>
<td>2586.20</td>
<td>1.0000</td>
<td>8</td>
</tr>
<tr>
<td>Location_Description</td>
<td>1463.00</td>
<td>161.66</td>
<td>0.1333</td>
<td>6</td>
</tr>
<tr>
<td>Domestic</td>
<td>154.49</td>
<td>0</td>
<td>0.0141</td>
<td>1</td>
</tr>
<tr>
<td>percent_households_below_poverty</td>
<td>5.7782</td>
<td>0</td>
<td>0.0005</td>
<td>1</td>
</tr>
<tr>
<td>Ward</td>
<td>4.3838</td>
<td>0</td>
<td>0.0004</td>
<td>1</td>
</tr>
</tbody>
</table>

Output 15  TREESPLIT Procedure Results: Pruning Plot

Assess the Models

Assessment is also available inside the SAS Visual Data Mining and Machine Learning add-on to SAS Visual Analytics.
Create a Macro for the ASSESS Procedure

- Macros can reduce repetitive code. The macro uses the scored tables that were generated by the analytic procedures (and the in-memory DATA step, in the case of the LOGSELECT procedure).
- The ODS OUTPUT statement is used to create a ROC and lift data set for each model.

```sas
%macro assess_model(prefix=, var_evt=, var_nevt=);
proc assess data=mycas.ap_scored_&prefix.
bins=10;
   input &var_evt.;
   target &target. / level=nominal event='1';
   fitstat pvar=&var_nevt. / pevent='0';
   by _partind_;
ods output fitstat=work.&prefix._fitstat
      rocinfo=work.&prefix._rocinfo
      liftinfo=work.&prefix._liftinfo;
run;
%mend assess_model;
```

```sas
title "Assess Forest";
%assess_model(prefix=forest,
       var_evt=p_arrest_code1,
       var_nevt=p_arrest_code0);

title "Assess Gradient Boost";
%assess_model(prefix=gradboost,
       var_evt=p_arrest_code1,
       var_nevt=p_arrest_code0);

title "Assess Logistic Regression";
%assess_model(prefix=logistic,
       var_evt=p_arrest_code1,
       var_nevt=p_arrest_code0);

title "Assess Decision Tree";
%assess_model(prefix=treesplit,
       var_evt=p_arrest_code1,
       var_nevt=p_arrest_code0);
```

The ASSESS procedure produces result tables for the lift, ROC information, and fit statistics. Instead of showing the tabular results, the data is plotted and shown in the following sections.

Prepare ROC and Lift Data Sets for Plotting

Combine all the ROC information data sets into a single data set. Do the same for the lift information data sets.

```sas
data work.all_rocinfo;
   set work.logistic_rocinfo(keep=sensitivity fpr_partind_in=l)
     work.forest_rocinfo(keep=sensitivity fpr_partind_in=f)
     work.treesplit_rocinfo(keep=sensitivity fpr_partind_in=t)
     work.gradboost_rocinfo(keep=sensitivity fpr_partind_in=g);
   length model $ 16;
   select;
```
```
when (l) model='Logistic';
when (f) model='Forest';
when (g) model='GradientBoosting';
when (t) model='TreeSplit';
end;
run;

data work.all_liftinfo;
set work.logistic_liftinfo(keep=depth lift cumlift _partind_ in=l)
  work.forest_liftinfo(keep=depth lift cumlift _partind_ in=f)
  work.treesplit_liftinfo(keep=depth lift cumlift _partind_ in=t)
  work.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';
  when (t) model='TreeSplit';
end;
run;

Plot ROC Curves

/* _partind_=2 specifies the test partition */
proc sgplot data=work.all_rocinfo(where=( _partind_=2 )) aspect=1;
  title "ROC Curves";
  series x=fpr y=sensitivity / group=model;
  lineparm x=0 y=0 slope=1 / transparency=.7;
  yaxis values=(0 to 1 by 0.25) grid offsetmin=.05 offsetmax=.05;
  xaxis values=(0 to 1 by 0.25) grid offsetmin=.05 offsetmax=.05;
run;

■ The WHERE clause in the data set option limits the data to the validation data. The training (0) and test (2) partitions are not included in the plot.

■ Because the data set includes the ROC information for all the models, the GROUP= option is used to plot a ROC curve for each series independently.
```
Plot Lift

proc sgplot data=work.all_liftinfo(where=(_partind_=2));
  title "Lift Chart";
  xaxis label="Percentile" grid;
  series x=depth y=lift / group=model markers
    markerattrs=(symbol=circlefilled);
run;

Your lift chart will look similar to the following.
Create Fit Statistics

Similar to the macro that was used for PROC ASSESS, the following code uses a macro to reduce the repetitive code for printing the fit statistics tables.

```sas
%macro print_fitstats(prefix=);
proc print data=work.&prefix._fitstat;
run;
%mend print_fitstats;

title "Forest Fit Statistics";
%print_fitstats(prefix=forest);

title "Gradient Boosting Fit Statistics";
%print_fitstats(prefix=gradboost);

title "Logistic Fit Statistics";
%print_fitstats(prefix=logistic);

title "TreeSplit Fit Statistics";
%print_fitstats(prefix=treesplit);

Remember that the partitioning is as follows:

0  training data
1  validation
```
## Forest Fit Statistics

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>PartInd</em></th>
<th>NOBS</th>
<th>ASE</th>
<th>DIV</th>
<th>RASE</th>
<th>MCE</th>
<th>MCLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>63571</td>
<td>0.098848</td>
<td>63571</td>
<td>0.314401</td>
<td>0.133599</td>
<td>0.323704</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>31785</td>
<td>0.107763</td>
<td>31785</td>
<td>0.328273</td>
<td>0.141859</td>
<td>0.358711</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10595</td>
<td>0.107231</td>
<td>10595</td>
<td>0.327462</td>
<td>0.140066</td>
<td>0.350360</td>
</tr>
</tbody>
</table>

## Gradient Boosting Fit Statistics

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>PartInd</em></th>
<th>NOBS</th>
<th>ASE</th>
<th>DIV</th>
<th>RASE</th>
<th>MCE</th>
<th>MCLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>63571</td>
<td>0.061443</td>
<td>63571</td>
<td>0.247877</td>
<td>0.082538</td>
<td>0.209933</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>31785</td>
<td>0.115834</td>
<td>31785</td>
<td>0.340345</td>
<td>0.149945</td>
<td>0.384551</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10595</td>
<td>0.115403</td>
<td>10595</td>
<td>0.339710</td>
<td>0.148938</td>
<td>0.384754</td>
</tr>
</tbody>
</table>

## Logistic Fit Statistics

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>PartInd</em></th>
<th>NOBS</th>
<th>ASE</th>
<th>DIV</th>
<th>RASE</th>
<th>MCE</th>
<th>MCLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>63567</td>
<td>0.112086</td>
<td>63567</td>
<td>0.334792</td>
<td>0.149653</td>
<td>0.361991</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>31781</td>
<td>0.113105</td>
<td>31781</td>
<td>0.336311</td>
<td>0.152796</td>
<td>0.366637</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10594</td>
<td>0.112316</td>
<td>10594</td>
<td>0.335135</td>
<td>0.150085</td>
<td>0.364265</td>
</tr>
</tbody>
</table>

## TreeSplit Fit Statistics

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>PartInd</em></th>
<th>NOBS</th>
<th>ASE</th>
<th>DIV</th>
<th>RASE</th>
<th>MCE</th>
<th>MCLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>63571</td>
<td>0.109483</td>
<td>63571</td>
<td>0.330882</td>
<td>0.139089</td>
<td>0.358368</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>31785</td>
<td>0.110266</td>
<td>31785</td>
<td>0.332063</td>
<td>0.140664</td>
<td>0.359722</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10595</td>
<td>0.109220</td>
<td>10595</td>
<td>0.330484</td>
<td>0.139217</td>
<td>0.360978</td>
</tr>
</tbody>
</table>
Score New Data

About This Section

- This section of the document shows how to get and analyze the latest data from the City of Chicago crime data. As a result, your experience and analysis should be similar, but not identical.
- Much of the code in this section is similar to the code in preceding sections. The differences to notice are as follows:
  - The latest records are downloaded directly from the City of Chicago web service with the JSON engine. A DATA step is used to match the data types from the crimes.sas7bdat file that was used.
  - The FOREST, GRADBOOST, and TREESPLIT procedures were used earlier with an OUTMODEL= option to save their models as in-memory tables. For scoring, the procedures are used again, with an INMODEL= option that reads a model from an in-memory table.
  - The training and validation data used a _PartInd_ column to identify the partition. The latest data does not have this column and it is removed from all the code.

Download and Prepare the Latest Data

- The first three lines of code cause the SAS Workspace Server to download the records with a date after 15FEB2016.
- The first DATA step sets the data types and formats to match the existing data.
- The second DATA step joins the most recent records with the census data. The date that is specified in the WHERE clause ensures that the records have not been processed already.

```sas
/* This option enables Server Name Indication on UNIX */
options set=SSL_USE_SNI=1;

/* Retrieve the columns that are used in the model only. */
/* Retrieve data with a date after 15FEB16 */
filename chicago url 'https://data.cityofchicago.org/resource/6zsd-86xi.json?$query=
select%20id%2C%20case_number%2C%20date%2C%20community_area
%2C%20fbi_code%2C%20location_description%2C%20domestic%2C%20beat
%2C%20district%2C%20ward%2C%20arrest
%20where%20date%20%3E%20%272016-02-15%27';
libname chicago sasejson;

data mycas.arrest err;
  set chicago.root(rename=(
    date=tmpts arrest=arrest_code domestic=tmpds
    beat=tbeat community_area=tca district=tdis   id=tid
    ward=tward
  ));
  if arrest_code eq 0 then arrest = 'false';
  else arrest = 'true';
  if tmpds eq 0 then domestic = 'false';
  else domestic = 'true';
  beat = input(tbeat, best12.);
```
community_area = input(tca, best12.);
district = input(tdis, best12.);
id = input(tid, best12.);
ward = input(tward, best12.);

format fbi_code $fbi. location_description $47.
arrest domestic $5.
arrest_code 8. date mmddyy10. timestamp datetime. ;
pos = kindex(tmpts, 'T');
if -1 eq pos then output err;
date = input(substr(tmpts, 1, pos-1), yymmdd10.);
time = input(substr(tmpts, pos+1), time.);
timestamp = dhms(date, 0, 0, time);

drop tmpts pos time tmpds tbeat tca tdis tid tward;
output mycas.arrest;
run;

data mycas.latest_crimes;
merge
    mycas.arrest(in=in1)
    mycas.census(in=in2);
by community_area;
if in1;
run;

Note: If you receive an error in the SAS log that is related to missing a CA trust list, then locate the trustedcerts.pem file that is part of the SAS installation, submit code like the following, and then rerun:

options sslcalistloc="/path/to/trustedcerts.pem";

Score the Latest Data

data mycas.latest_logistic;
set mycas.latest_crimes;
%include "&outdir./logselect1.sas";
p_&target.1=p_&target.;
p_&target.0=1-p_&target.;
run;

proc treesplit data=mycas.latest_crimes inmodel=mycas.model_treesplit noprint;
target &target. /level=nominal;
input &nom_input. /level=nominal;
in &int_input. /level=interval;
output out=mycas.latest_treesplit copyvars=(&target);
run;

proc gradboost data=mycas.latest_crimes inmodel=mycas.model_gradboost noprint;
output out=mycas.latest_gradboost copyvars=(&target);
run;

proc forest data=mycas.latest_crimes inmodel=mycas.model_forest noprint;
output  out=mycas.latest_forest copyvars=(&target);
run;
An in-memory DATA step is used to score the data with the model from PROC LOGSELECT. The other procedures use the INMODEL= option to read the model from an in-memory table.

The result of this code is an additional four in-memory tables. You can use the ASSESS procedure again to see how the models operated on the latest data.

Assess the Models on the Latest Data

```
%macro assess_latest(prefix=, var_evt=, var_nevt=);
  proc assess data=mycas.latest_&prefix. nbins=10;
    input &var_evt.;
    target &target. / level=nominal event='1';
    fitstat pvar=&var_nevt. / pevent='0';
  ods output fitstat=work.&prefix._lfitstat
                 rocinfo=work.&prefix._lrocinfo
                 liftinfo=work.&prefix._lliftinfo;
  run;
%mend assess_latest;

title "Assess Decision Tree";
%assess_latest(prefix=treesplit, var_evt=p_arrest_code1, var_nevt=p_arrest_code0);
title "Assess Forest";
%assess_latest(prefix=forest, var_evt=p_arrest_code1, var_nevt=p_arrest_code0);
title "Assess Gradient Boost";
%assess_latest(prefix=gradboost, var_evt=p_arrest_code1, var_nevt=p_arrest_code0);
title "Assess Logistic Regression";
%assess_latest(prefix=logistic, var_evt=p_arrest_code1, var_nevt=p_arrest_code0);

Combine the ROC and lift data sets that were written to the Work libref on the SAS client.

```
data work.latest_rocinfo;
  set work.logistic_lrocinfo(keep=sensitivity fpr in=l)
        work.forest_lrocinfo(keep=sensitivity fpr in=f)
        work.treesplit_lrocinfo(keep=sensitivity fpr in=t)
        work.gradboost_lrocinfo(keep=sensitivity fpr in=g);
  length model $ 16;
  select;
    when (l) model='Logistic';
    when (f) model='Forest';
    when (g) model='GradientBoosting';
    when (t) model='TreeSplit';
  end;
run;
```

```
data work.latest_liftinfo;
  set work.logistic_lliftinfo(keep=depth lift cumlift in=l)
        work.forest_lliftinfo(keep=depth lift cumlift in=f)
        work.treesplit_lliftinfo(keep=depth lift cumlift in=t)
        work.gradboost_lliftinfo(keep=depth lift cumlift in=g);
  length model $ 16;
  select;
    when (l) model='Logistic';
    when (f) model='Forest';
    when (g) model='GradientBoosting';
```
when (t) model='TreeSplit';
end;
run;

**Plot ROC Curves and Lift Chart**

```
proc sgplot data=work.latest_rocinfo aspect=1;
  title "ROC Curves";
  title2 "Latest Data";
  series x=fpr y=sensitivity / group=model;
  lineparm x=0 y=0 slope=1 / transparency=.7;
  yaxis values=(0 to 1 by 0.25) grid offsetmin=.05 offsetmax=.05;
  xaxis values=(0 to 1 by 0.25) grid offsetmin=.05 offsetmax=.05;
run;
```

```
proc sgplot data=work.latest_liftinfo;
  title "Lift Chart";
  title2 "Latest Data";
  xaxis label="Percentile" grid;
  series x=depth y=lift / group=model markers markerattrs=(symbol=circlefilled);
run;
```
Additional Reading

Here is the recommended reading list for this title:

- SAS Cloud Analytic Services: User’s Guide
- SAS Cloud Analytic Services: Fundamentals
- SAS Visual Data Mining and Machine Learning: Statistical Procedures
- SAS Visual Data Mining and Machine Learning: Data Mining and Machine Learning Procedures