



Deploying and Managing Models in SAS[®] Viya[®]

Course Notes

Deploying and Managing Models in SAS® Viya® Course Notes was developed by Jordan Bakerman and Ari Zitin. Additional contributions were made by NULL. Instructional design, editing, and production support was provided by the Learning Design and Development team.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

Deploying and Managing Models in SAS® Viya® Course Notes

Copyright © 2019 SAS Institute Inc. Cary, NC, USA. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Book code E71534, course code UA19DMMM, prepared date 16Aug2019.

UA19DMMM_001

ISBN 978-1-64295-551-4

Table of Contents

Lesson 1	Managing SAS® and Open Source Models	1-1
1.1	Introduction	1-3
1.2	Building and Managing Models	1-5
	Demonstration: Building SAS and Open Source Models in a Pipeline	1-6
	Demonstration: Setting Project Properties	1-18
	Demonstration: Importing an Open Source Model into Model Manager	1-21
	Demonstration: Monitoring Champion Model Performance	1-30
	Demonstration: Retraining the Champion Model and Deploying to MAS	1-40
1.3	Deploying Models	1-43
	Demonstration: AI Object Detection	1-45

To learn more...



For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at <http://support.sas.com/training/> as well as in the Training Course Catalog.

For a list of SAS books (including e-books) that relate to the topics covered in this course notes, visit <https://www.sas.com/sas/books.html> or call 1-800-727-0025. US customers receive free shipping to US addresses.

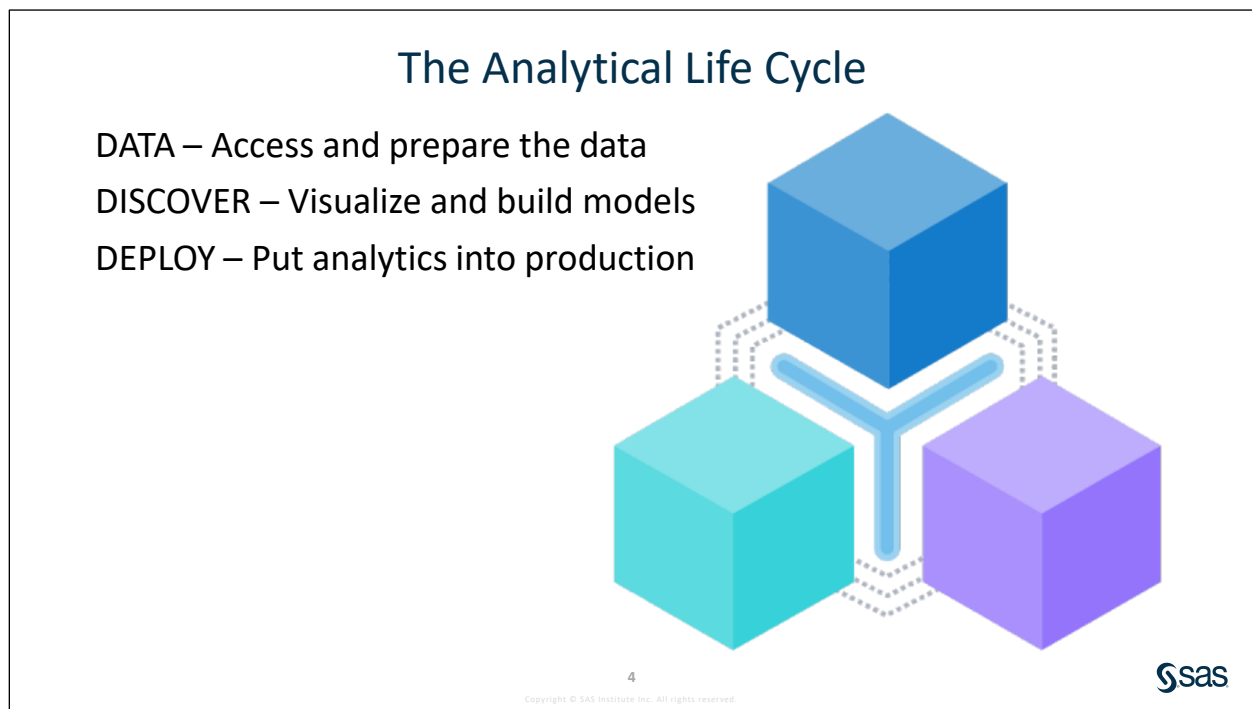
Lesson 1 Managing SAS[®] and Open Source Models

- 1.1 Introduction 1-3**

- 1.2 Building and Managing Models..... 1-5**
 - Demonstration: Building SAS and Open Source Models in a Pipeline 1-6
 - Demonstration: Setting Project Properties 1-18
 - Demonstration: Importing an Open Source Model into Model Manager 1-21
 - Demonstration: Monitoring Champion Model Performance 1-30
 - Demonstration: Retraining the Champion Model and Deploying to MAS 1-40

- 1.3 Deploying Models 1-43**
 - Demonstration: AI Object Detection..... 1-45

1.1 Introduction



DATA – The SAS Platform streamlines data access and preparation while ensuring that the proper controls are in place. Access data of any complexity, size, or speed with a robust suite of data management tools.

DISCOVERY – Visualize and build analytical models. Take advantage of programming flexibility by building SAS, Python, and R models in a unified pipeline approach.

DEPLOY – Putting analytics into production is where value is realized. The SAS Platform helps you put analytics into action quickly using a transparent and governed process so that you can reap value from your data faster. Move quickly from creating and selecting the champion model to deployment with robust model management capabilities, whether you are building a single model or thousands.

SAS Model Manager

Streamlines the steps of creating, managing, deploying, monitoring, and operationalizing analytical models.



Maintains a repository of SAS and open source models for analytical projects.



5

Copyright © SAS Institute Inc. All rights reserved.



SAS Model Manager streamlines the often error-prone steps used to bring analytical models to production. Model Manager maintains a model repository for each project to manage and deploy the organization's SAS and open source models. The repository enables you to easily compare registered models, and it provides tools to monitor the performance of production models. In the event of model degradation, models can be efficiently retrained and redeployed.

1.2 Building and Managing Models

A financial services company offers a home equity line of credit to its clients. The company extended several thousand lines of credit in the past, and many of these accepted applicants (approximately 20%) defaulted on their loans. By using geographic, demographic, and financial variables, the company wants to build a model to predict whether an applicant might default.

After analyzing the data, the company selected a subset of 12 predictor (or input) variables to model whether each applicant defaulted. The response (or target) variable (**BAD**) indicates whether an applicant defaulted on the home equity line of credit. These variables, with their model role, measurement level, and description, are shown in the following table from the **HMEQ** data set.

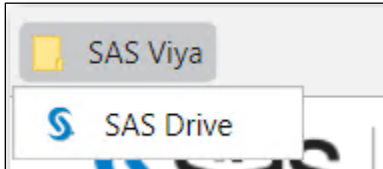
NAME	MODEL ROLE	MEASUREMENT LEVEL	DESCRIPTION
BAD	Target	Binary	1 = applicant defaulted on loan or delinquent, 0 = applicant paid loan
CLAGE	Input	Interval	Age of oldest credit line in months
CLNO	Input	Interval	Number of credit lines
DEBTINC	Input	Interval	Debt to income ratio
DELINQ	Input	Interval	Number of delinquent credit lines
DEROG	Input	Interval	Number of derogatory reports
JOB	Input	Nominal	Occupational categories
LOAN	Input	Interval	Amount of loan request
MORTDUE	Input	Interval	Amount due on existing mortgage
NINQ	Input	Interval	Number of recent credit inquiries
REASON	Input	Binary	<i>DebtCon</i> = debt consolidation, <i>HomeImp</i> = home improvement
VALUE	Input	Interval	Value of current property
YOJ	Input	Interval	Years at present job



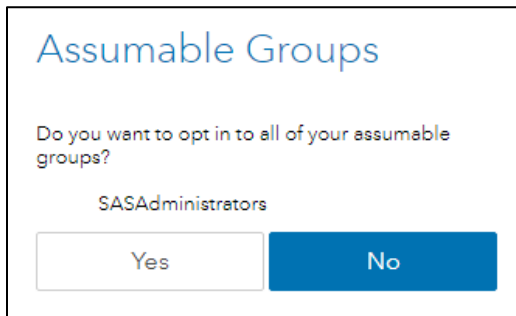
Building SAS and Open Source Models in a Pipeline

Open SAS Drive


1. Follow the instructions given by your instructor to access the virtual machine.
2. Click the **Google Chrome** shortcut on your virtual machine.
3. Open the **SAS Viya** shortcut folder and select **SAS Drive**.



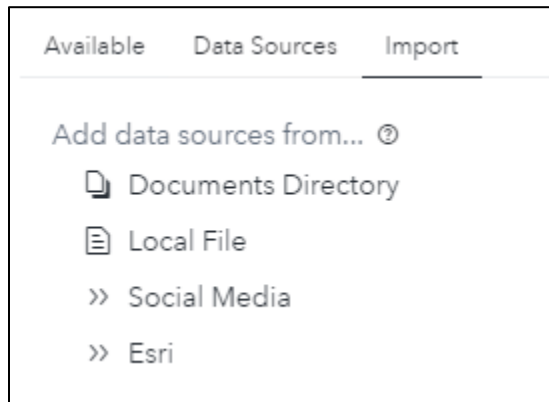
4. Sign in to SAS using the user name **student** and the password **Metadata0**.
5. Click **No** when asked if you want to opt in to your assumable groups.



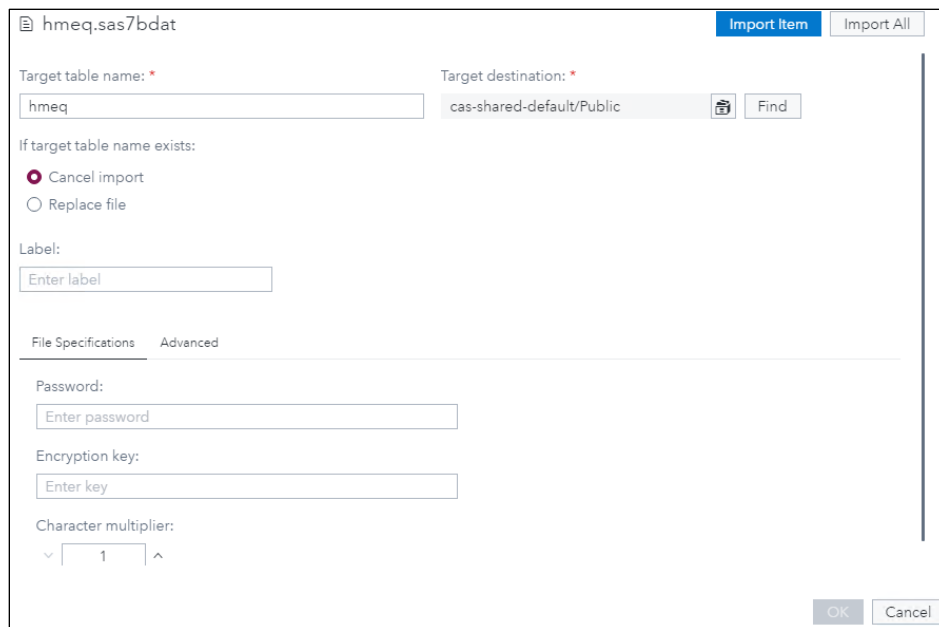
Load the Data and Create a Project

1. Open the **Show Applications** menu  and select **Build Models** to open the Model Studio visual interface.
2. In the Model Studio application, select **New Project**.
3. Name your project **Model Manager Workshop**, and leave **Type** and **Template** at their default values.
4. Under **Data**, select **Browse** to access the Choose Data window.

- From the Choose Data window, click the **Import** tab.



- Select **Local File** to import a file from the local client machine.
- Navigate to **D:\Workshop\Winsas\UA19DMMM**. Select the file **hmeq.sas7bdat** and click **Open**.
- Click **Import Item** to import the data to the SAS server. After the import is successful, click **OK** to use this data source for your project.



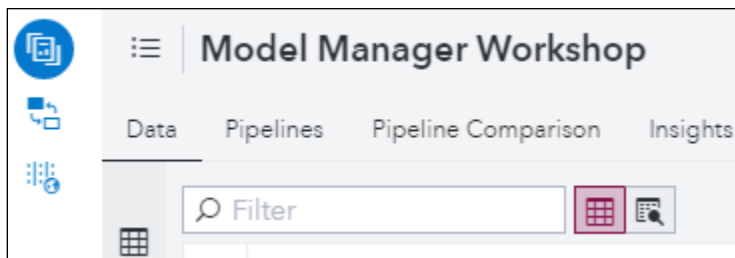
- Click **Save** in the New Project window to save the new project settings.


Build a Modeling Pipeline

- After the project is created, it opens and brings you to the Data tab. The software has already tried to make some decisions about how to treat the input variables, but some changes are needed. Select **BAD** and change the role from Input to **Target**.

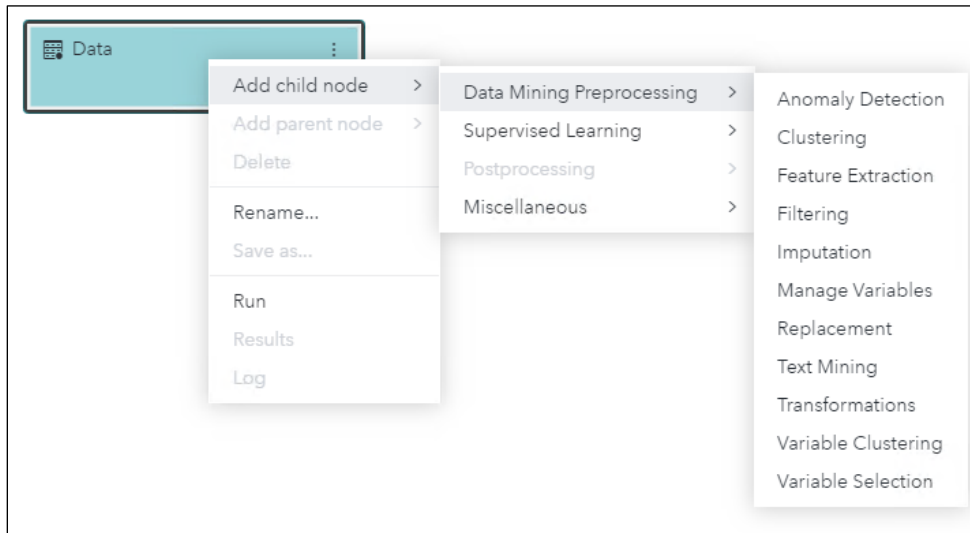
<input type="checkbox"/>	Variable Name	↑	Label	Type	Role	Level	Order #	
<input checked="" type="checkbox"/>	BAD			Numeric	Target	Binary	Default	Role: Target
<input type="checkbox"/>	CLAGE			Numeric	Input	Interval	Default	Level: Binary
<input type="checkbox"/>	CLNO			Numeric	Input	Interval	Default	Specify the Target Event Level
<input type="checkbox"/>	DEBTINC			Numeric	Input	Interval	Default	Order: Default
<input type="checkbox"/>	DELINQ			Numeric	Input	Nominal	Default	
<input type="checkbox"/>	DEROG			Numeric	Input	Nominal	Default	
<input type="checkbox"/>	JOB			Character	Input	Nominal	Default	

- Select **Pipelines** next to **Data** to move to the Pipelines tab to create a predictive modeling pipeline.

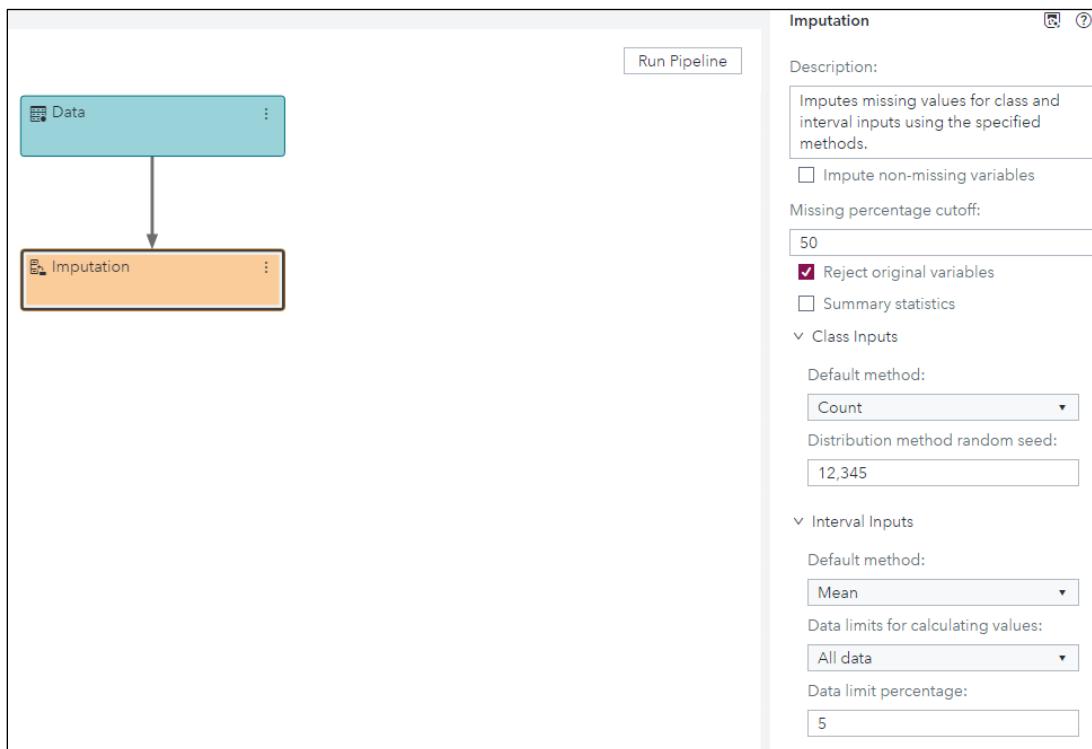


Note that the default predictive modeling partition is 60% for training, 30% for validation, and 10% for test. For this project, we keep the default settings. However, the partition can be changed by selecting project settings from the Settings icon .

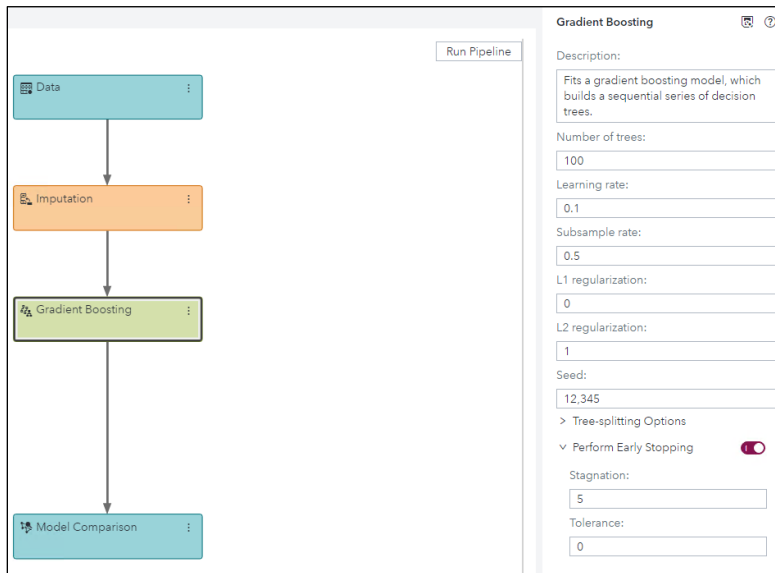
- The **HMEQ** data set has missing data for 11 of the 13 variables. Only the target **BAD** and the input **LOAN** do not contain missing values. To impute missing values, right-click the **Data** node and select **Add child node** ⇒ **Data Mining Preprocessing** ⇒ **Imputation**.



- Click the **Imputation** node to display the settings for imputing missing values. Notice that, by default, missing class inputs are replaced with the mode (**Count** refers to the one with the highest count), and missing interval inputs are replaced with the mean.

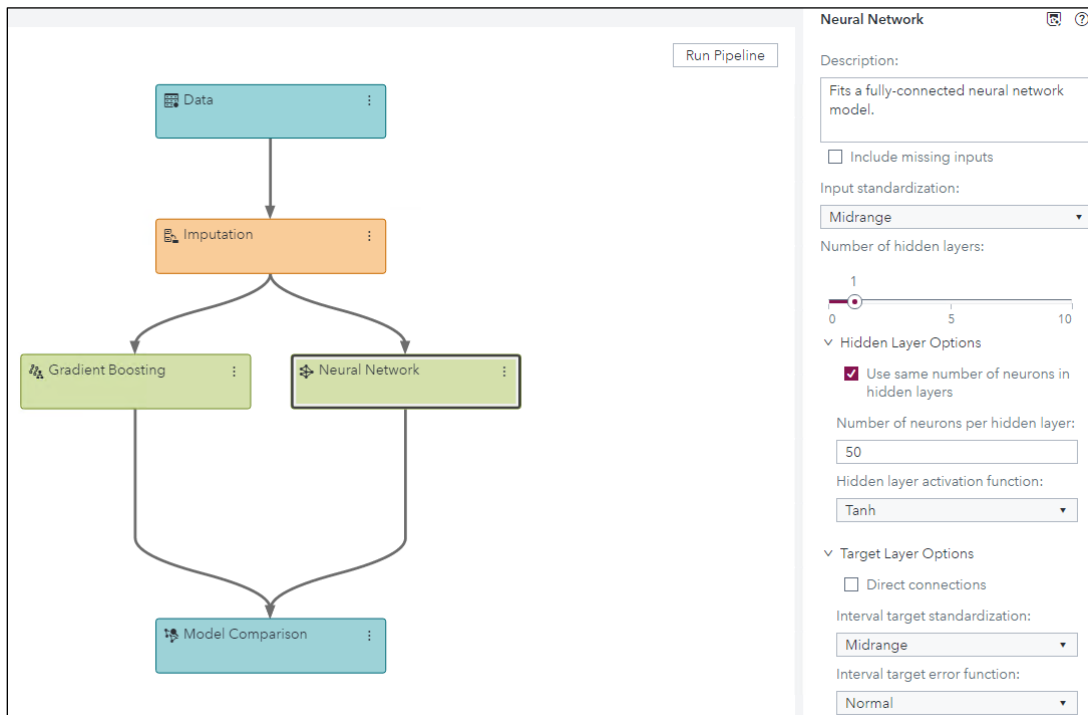


- To add a SAS model to the pipeline, right-click the **Imputation** node and select **Add child node** ⇒ **Supervised Learning** ⇒ **Gradient Boosting**.

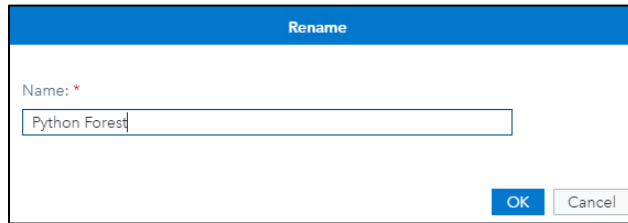



- Add another SAS model to the pipeline. Right-click the **Imputation** node and select **Add child node** ⇒ **Supervised Learning** ⇒ **Neural Network**.

Note: The neural network model requires imputation because it ignores any rows with missing values. Tree-based models can handle missing values by default. To have an even playing field for all our models within the pipeline, the models are connected to the Imputation node.



7. Right-click the **Imputation** node and select **Add child node** ⇒ **Miscellaneous** ⇒ **Open Source Code**.
8. Right-click the **Open Source Code** node and select **Rename**. Rename the node **Python Forest**.



9. In the Python Forest settings, select **Data Sample** to open more node options. Change the sampling method to **None** and exclude SAS formats. In order to run an open source model from Model Studio, the data are downloaded from the server to the client where the Python model is run. Because the **HMEQ** data set contains only 5960 observations, we can download the entire data, instead of just a sample, using the None sampling method without concern for flooding the local memory. Excluding SAS formats can prevent possible formatting errors when converting data from SAS to Python or R.
10. From the Python Forest settings, select **Open Code Editor**. On the operating system, navigate to **D:\WorkshopWinsas\UA19DMMM** and open the text file **Python_Forest_Code.txt**. Copy the full text of the file into the code editor and click **Save**  in the top right and then click **Close**.

```

1 #####
2 ### Python ###
3 #####
4
5 from sklearn import ensemble
6
7 # Get full data with inputs + partition indicator
8 dm_input.insert(0, dm_partitionvar)
9 fullX = dm_inputdf.loc[:, dm_input]
10
11 # Dummy encode categorical variables
12 fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)
13
14 # Training data
15 X_enc = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
16 X_enc = X_enc.drop(dm_partitionvar, 1)
17
18 # Labels
19 y = dm_traindf[dm_dec_target]
20
21 # Fit RandomForest model w/ training data
22 params = {'n_estimators': 100, 'max_depth': 5, 'min_samples_split': 100,
23          'min_samples_leaf': 100, 'max_features': 5, 'random_state': 12345}
24 dm_model = ensemble.RandomForestClassifier(**params)
25 dm_model.fit(X_enc, y)
26 print(dm_model)
27
28 # Save VariableImportance to CSV
29 varimp = pd.DataFrame(list(zip(X_enc, dm_model.feature_importances_)), columns=['Variable Name', 'Importance'])
30 varimp.to_csv(dm_nodedir + '/rpt_var_imp.csv', index=False)
31
32 # Score full data
33 fullX_enc = fullX_enc.drop(dm_partitionvar, 1)
34 dm_scoreddf = pd.DataFrame(dm_model.predict_proba(fullX_enc), columns=['P_BAD0', 'P_BAD1'])

```

Notice that to the left of the code editor are variables that were created for make writing the open source code easier. For example, **dm_class_input** will contain all the categorical variables in the pipeline's data. Other variables are required for the flow of data back and forth between the two languages. For example, you must save your predictions in **dm_scoreddf** to return the results back to the pipeline.

- a. Load the RandomForestClassifier model.

```
from sklearn import ensemble
```

- b. Dummy code the categorical inputs. Python accepts only numeric variables within machine learning algorithms. Therefore, categorical variables must be dummy coded before running the algorithm.

```
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input,
                           drop_first=True)
```

- c. Get the training data and target.

```
X_enc = fullX_enc[fullX_enc[dm_partitionvar] ==
                  dm_partition_train_val]
X_enc = X_enc.drop(dm_partitionvar, 1)
y = dm_traindf[dm_dec_target]
```

- d. Fit the Python random forest and print the model object.

```
params = {'n_estimators': 100, 'max_depth': 5,
          'min_samples_split': 100, 'min_samples_leaf': 100,
          'max_features': 5, 'random_state': 12345}
dm_model = ensemble.RandomForestClassifier(**params)
dm_model.fit(X_enc, y)
print(dm_model)
```

Note: The PRINT function returns command-line output from Python to the Model Studio results page.

Selected RandomForestClassifier arguments:

n_estimators	Number of trees in the forest.
max_depth	Maximum depth allowed for each tree.
min_samples_split	Minimum number of samples required to split an internal node.
min_samples_leaf	Minimum number of samples required to be a leaf node. A split point at any depth is considered only if it leaves this many training samples in each of the left and right branches.
max_features	Number of features to consider when looking for the best split.
random_state	Seed used by the random number generator to reproduce results.

- e. Save variable importance to be printed in Model Studio. To save and return the variable importance measures, save the data frame as a CSV file in the predetermined node directory using the variable handle **dm_nodedir**.

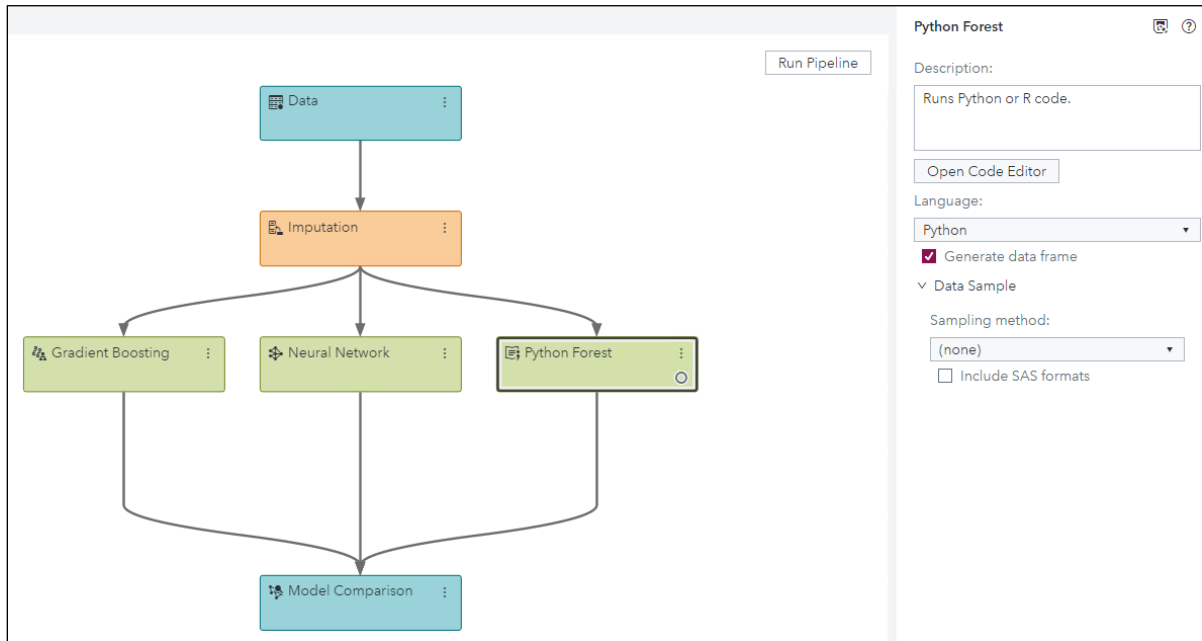
```
varimp = pd.DataFrame(list(zip(X_enc,
                              dm_model.feature_importances_)),
                      columns=['Variable Name', 'Importance'])
varimp.to_csv(dm_nodedir + '/rpt_var_imp.csv', index=False)
```

- f. Score the **HMEQ** data and save predictions in **dm_scoredf**.

```
fullX_enc = fullX_enc.drop(dm_partitionvar, 1)
dm_scoreddf = pd.DataFrame(dm_model.predict_proba(fullX_enc),
                           columns=['P_BAD0', 'P_BAD1'])
```


To return the prediction to Model Studio, you must save results in the **dm_scoreddf** variable handle. Furthermore, you must also use the SAS naming conventions for the predictions to match the pipeline standards. The naming convention is **P_** followed by the target name **BAD** and then the level of prediction, 1 or 0. That is, the column names are **P_BAD0** and **P_BAD1**.

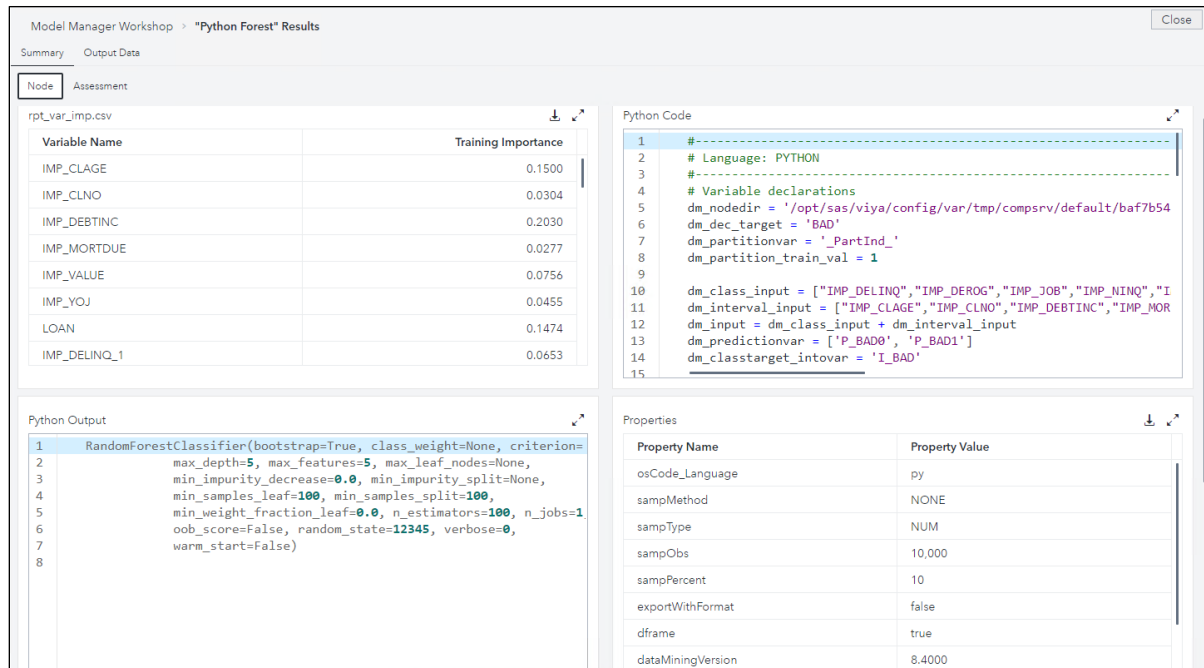
- Right-click the **Python Forest** node and select **Move** ⇒ **Supervised Learning**. This tells the software that this node contains a predictive model and will output scored data. It turns the node green and automatically connects it to the Model Comparison node.



- Click **Run Pipeline**.

Compare Models and Register the Champion

1. Right-click the Python Forest node and select Results.



Model Manager Workshop > "Python Forest" Results

Summary Output Data

Node Assessment

rpt_var_imp.csv

Variable Name	Training Importance
IMP_CLAGE	0.1500
IMP_CLNO	0.0304
IMP_DEBTINC	0.2030
IMP_MORTDUE	0.0277
IMP_VALUE	0.0756
IMP_YOJ	0.0455
LOAN	0.1474
IMP_DELIQ_1	0.0653

Python Code

```

1 #-----
2 # Language: PYTHON
3 #-----
4 # Variable declarations
5 dm_nodedir = '/opt/sas/viya/config/var/tmp/compsrv/default/baf7b54
6 dm_dec_target = 'BAD'
7 dm_partitionvar = '_PartInd_'
8 dm_partition_train_val = 1
9
10 dm_class_input = ["IMP_DELIQ", "IMP_DEROG", "IMP_JOB", "IMP_NINQ", "I
11 dm_interval_input = ["IMP_CLAGE", "IMP_CLNO", "IMP_DEBTINC", "IMP_MOR
12 dm_input = dm_class_input + dm_interval_input
13 dm_predictionvar = ['P_BAD0', 'P_BAD1']
14 dm_classtarget_intovar = 'I_BAD'
15

```

Python Output

```

1 RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
2 max_depth=5, max_features=5, max_leaf_nodes=None,
3 min_impurity_decrease=0.0, min_impurity_split=None,
4 min_samples_leaf=100, min_samples_split=100,
5 min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1
6 oob_score=False, random_state=12345, verbose=0,
7 warm_start=False)
8

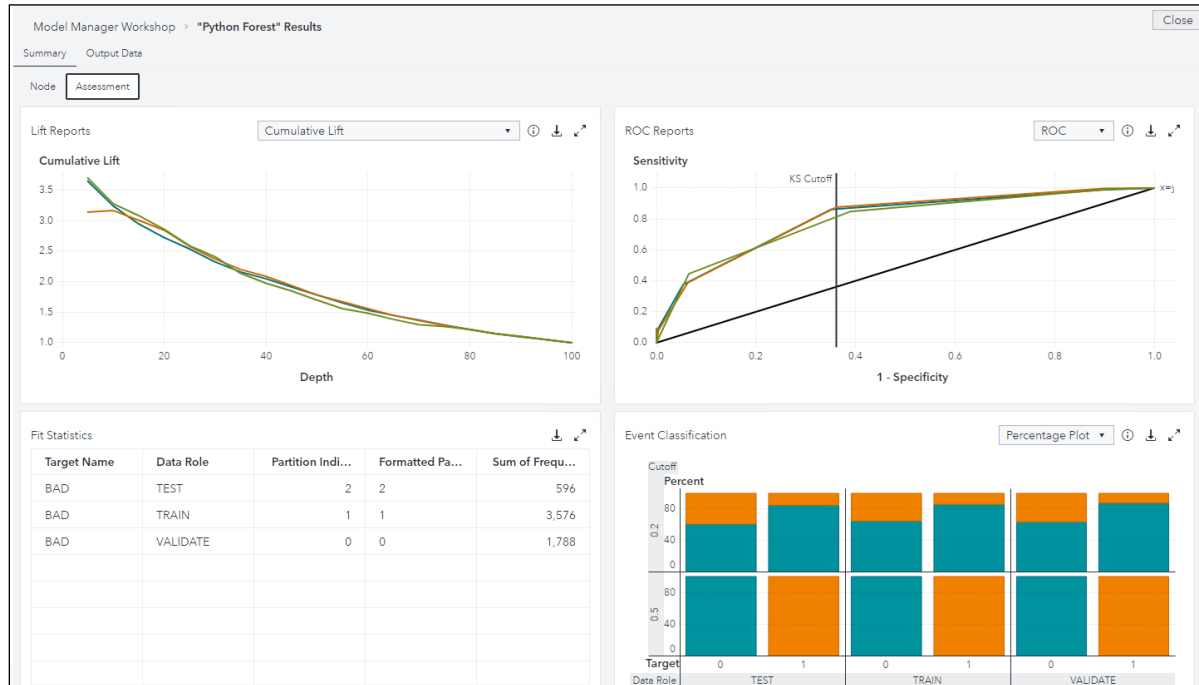
```

Properties

Property Name	Property Value
osCode_Language	py
sampMethod	NONE
sampType	NUM
sampObs	10,000
sampPercent	10
exportWithFormat	false
dframe	true
dataMiningVersion	8.4000

The first window displays the saved CSV file of variable importance measures, and the Python output window displays the model output requested with the PRINT statement. The Python code window shows not only the code that we copied and pasted into the code editor but also automatic variable handles created for the user.

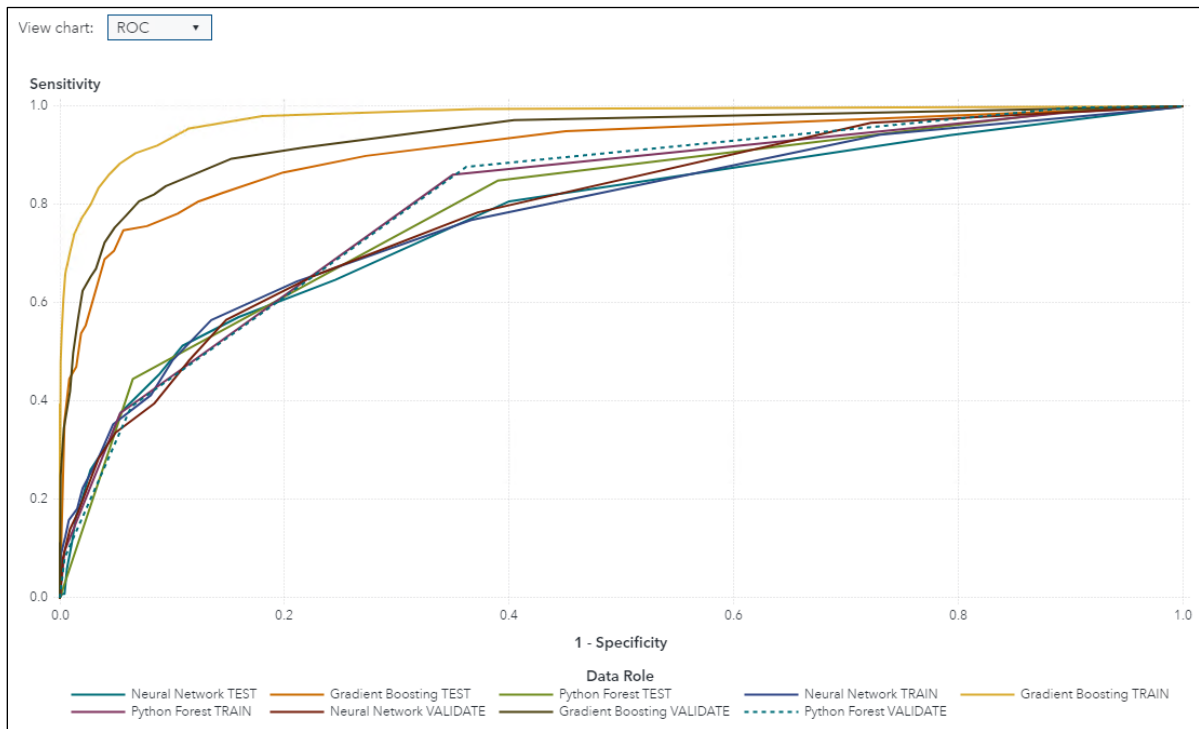
- Navigate to the Assessment tab to see lift plots, ROC plots, and fit statistics for the open source model. Notice that the Open Source Code node still produces the same assessment plots as all the other modeling nodes provided by SAS.



For Python or R models, SAS automatically creates lift plots, ROC plots, and other model comparison tools when scored data are returned to the pipeline from the Open Source Code node.

- Close the Python Forest node.
- Right-click the **Model Comparison** node and select **Results**.
- Click the **Assessment** tab to view numeric and graphical comparisons of the models.

6. Change the accuracy graphic to **ROC** and expand the window.



The ROC comparison for the three models shows that the gradient boosting model performs the best, with the largest C statistic for all three data partitions, followed by the SAS neural network and the Python random forest.

7. Close the ROC window and expand the Fit Statistics window.

Statistics Label	Train: Python ...	Validate: Pyth...	Test: Python F...	Train: Gradien...	Validate: Gra...	Test: Gradien...	Train: Neural ...	Validate: Neu...	Test: Neural ...
Area Under ROC	0.8029	0.8070	0.7962	0.9769	0.9380	0.9123	0.7793	0.7866	0.7727
Average Squared Error	0.1453	0.1453	0.1455	0.0460	0.0665	0.0763	0.1258	0.1278	0.1284
Divisor for ASE	3,576	1,788	596	3,576	1,788	596	3,576	1,788	596
Formatted Partition	1	0	2	1	0	2	1	0	2
Gamma	0.7853	0.7965	0.7679	0.9648	0.9067	0.8644	0.6289	0.6433	0.6100
Gini Coefficient	0.6058	0.6141	0.5924	0.9539	0.8761	0.8247	0.5586	0.5732	0.5455
KS (Youden)	0.5108	0.5147	0.4588	0.8398	0.7431	0.6912	0.4327	0.4290	0.4120
KS Cutoff	0.2000	0.2000	0.2000	0.1500	0.2500	0.4000	0.2000	0.2000	0.2500
Misclassification at Cutoff	0.1993	0.1996	0.1996	0.0609	0.0872	0.0939	0.1711	0.1750	0.1694
Misclassification Rate	0.1993	0.1996	0.1996	0.0609	0.0872	0.0939	0.1711	0.1750	0.1694
Misclassification Rate (Event)	0.1993	0.1996	0.1996	0.0609	0.0872	0.0939	0.1711	0.1750	0.1694
Multi-Class Log Loss	0.4568	0.4568	0.4578	0.1668	0.2282	0.2627	0.4069	0.4097	0.4156
Partition Indicator	1	0	2	1	0	2	1	0	2

Again, the numeric summaries show that the gradient boosting model performed the best overall. For this reason, we declare the SAS gradient boosting model the champion and register it to Model Manager for follow up analyses. Note that the forthcoming analyses can also be done with open source models.

- Close the Fit Statistics window and then navigate back to the Node tab. Notice that the champion model for this pipeline is designated as the gradient boosting model, with the largest KS statistic and smallest misclassification rate.

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate
<input checked="" type="checkbox"/>	Gradient Boosting	Gradient Boosting	0.6913	0.0940
	Python Forest	Open Source Code	0.4588	0.1997
	Neural Network	Neural Network	0.4121	0.1695

- Close the Model Comparison node and then click the **Pipeline Comparison** tab.

Data Pipelines Pipeline Comparison Insights

Filter Data: Test Compare

Champion	Name	Algorithm Name	Pipeline Name	KS (Youden)	Sum of Frequency
<input checked="" type="checkbox"/>	Gradient Boosting	Gradient Boosting	Pipeline 1	0.691	596

Error Plot

Average Squared Error

Number of Trees

Variable Importance

Variable Label	Variable Name	Training Imp...	Importance ...	Relative Imp...
Imputed DEBTINC	IMP_DEBTINC	13.5956	13.0522	1
Imputed DELINQ	IMP_DELINQ	6.4156	16.0096	0.4719
Imputed CLAGE	IMP_CLAGE	3.8663	2.7957	0.2844
Imputed VALUE	IMP_VALUE	3.0484	1.8316	0.2242
Imputed NINQ	IMP_NINQ	2.7530	1.7287	0.2025
Imputed	IMP_MORTDU	2.2022	1.1480	0.1404

- Right-click the champion model name and then click **Register models**.

Register Models

Models

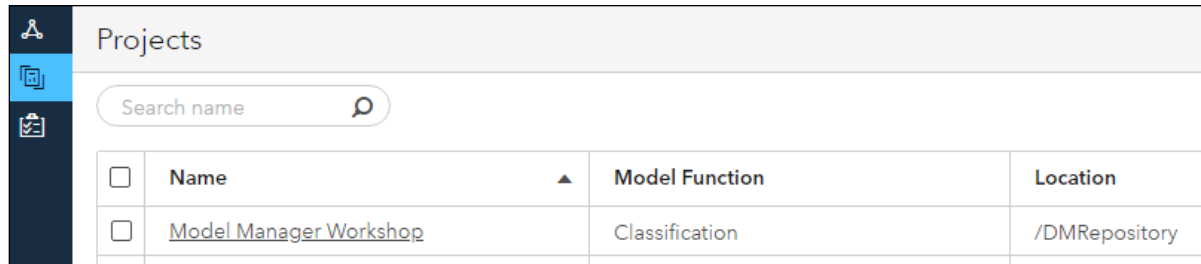
Name:	Status:
Gradient Boosting	✔ Registered successfully

End of Demonstration



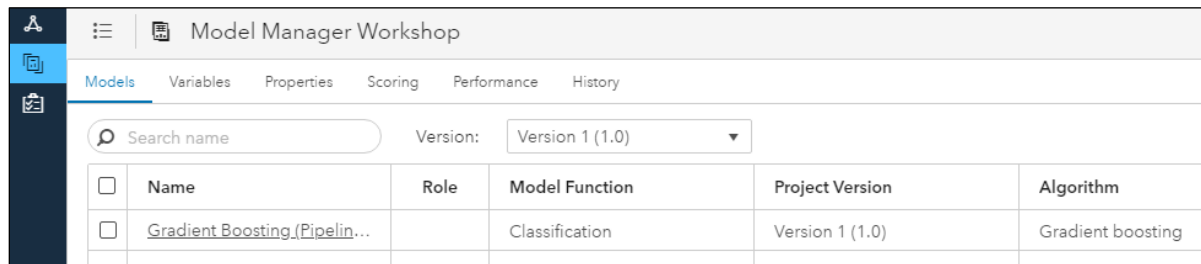
Setting Project Properties

1. Open the **Show Applications** menu  and select **Manage Models** to open the Model Manager visual interface.




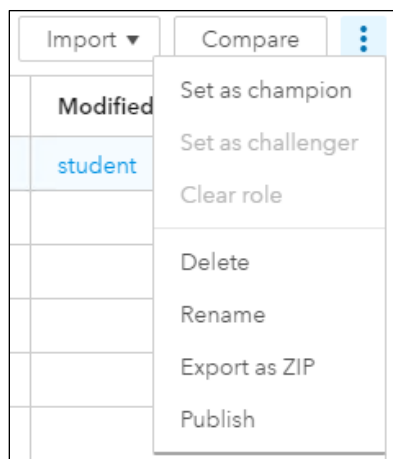
<input type="checkbox"/>	Name	Model Function	Location
<input type="checkbox"/>	Model Manager Workshop	Classification	/DMRepository


2. Select the **Model Manager Workshop** project.



<input type="checkbox"/>	Name	Role	Model Function	Project Version	Algorithm
<input type="checkbox"/>	Gradient Boosting (Pipel...		Classification	Version 1 (1.0)	Gradient boosting

3. Select the box next to the Gradient Boosting model and then click the **Actions** icon  in the top right. Select **Set as champion** from the **Actions** list. Setting the champion is optional, but doing so enables Model Manager to set default project settings needed for future analyses. For example, setting the champion model imports the appropriate variables into the project from the Model Studio pipeline.



Import ▾	Compare	
Modified	Set as champion	
student	Set as challenger	
	Clear role	
	Delete	
	Rename	
	Export as ZIP	
	Publish	

- Select **EM_EVENTPROBABILITY** as the model output variable to use as the project output variable. Click **Save**. This simply species the variable name to be used as output for scoring within the Model Manager project.


Select Project Output Variables

Select the model output variables to use as project level output variables. The project output variables are used to generate output results when running performance monitoring reports.

<input type="checkbox"/>	Model Output Variables	Project Output Variables
<input type="checkbox"/>	EM_CLASSIFICATION	EM_CLASSIFICATION
<input checked="" type="checkbox"/>	EM_EVENTPROBABILITY	EM_EVENTPROBABILITY
<input type="checkbox"/>	EM_PROBABILITY	EM_PROBABILITY
<input type="checkbox"/>	IMP_CLAGE	IMP_CLAGE
<input type="checkbox"/>	IMP_CLNO	IMP_CLNO
<input type="checkbox"/>	IMP_DEBTINC	IMP_DEBTINC
<input type="checkbox"/>	IMP_DEBNO	IMP_DEBNO

Save Cancel

- Click **Yes** to add all the model variables to the project. This adds all the Model Studio pipeline variables to the project in lieu of manually importing the variables.

 SAS Model Manager


Some of the champion model input variables for this version are not project input variables.


Do you want to add these input variables to the project?

Yes No

6. From the Projects window, click the **Variables** tab to view the imported project variables.

Models Variables Properties Scoring Performance History							
<input type="checkbox"/>	Name ^	Data Type	Input ▲	Output	Length	Measurement	
<input type="checkbox"/>	CLAGE	Decimal	✓		8	Interval	
<input type="checkbox"/>	CLNO	Decimal	✓		8	Interval	
<input type="checkbox"/>	DEBTINC	Decimal	✓		8	Interval	
<input type="checkbox"/>	DELINQ	Decimal	✓		8	Nominal	
<input type="checkbox"/>	DEROG	Decimal	✓		8	Nominal	
<input type="checkbox"/>	JOB	Character	✓		6	Nominal	
<input type="checkbox"/>	LOAN	Decimal	✓		8	Interval	
<input type="checkbox"/>	MORTDUE	Decimal	✓		8	Interval	
<input type="checkbox"/>	NINQ	Decimal	✓		8	Nominal	
<input type="checkbox"/>	REASON	Character	✓		7	Binary	
<input type="checkbox"/>	VALUE	Decimal	✓		8	Interval	
<input type="checkbox"/>	YOJ	Decimal	✓		8	Interval	
<input type="checkbox"/>	EM_EVENTPROBABILITY	Decimal		✓	8		

7. From the Projects window, select **Properties**. Set **Default train table** to **HMEQ** (if the table is not available, you might need to refresh), set **Target variable** to **BAD**, set **Target event value** to **1**, set **Target values** to **1** and **0**, set **Target level** to **Binary**, and set **Output event probability variable** to **EM_EVENTPROBABILITY**. Finally, click the **Save** icon  in the top right. These steps ensure that the user can score new data with any model within the project.

Champion version:	Version 1
Champion model name:	Gradient Boosting (Pipeline 1)
Default train table:	cas-shared-default/Public/HMEQ  <input type="button" value="Clear"/>
Target variable:	BAD
Target event value:	1
Target values:	1 0
Target level:	Binary ▼
Output event probability variable:	EM_EVENTPROBABILITY ▼

End of Demonstration



Importing an Open Source Model into Model Manager

View the Open Source Code

1. Open the `R_Logistic_Regression_Code.txt` file located in the course folder at `D:\Workshop\Winsas\UA19DMMM`.
2. Create a logistic regression model in R.
 - a. Load the `HMEQ.csv` file and count the number of missing values for each variable.

```
# Load
df <- read.csv(file="D:/Workshop/Winsas/UA19DMMM/hmeq.csv",
               header=TRUE, sep=",")

head(df)
dim(df)

# Missing
sum(is.na(df) | df=="") # 5271

# Interval
sum(is.na(df$MORTDUE))
sum(is.na(df$VALUE))
sum(is.na(df$YOJ))
sum(is.na(df$DEROG))
sum(is.na(df$DELINQ))
sum(is.na(df$CLAGE))
sum(is.na(df$NINQ))
sum(is.na(df$CLNO))
sum(is.na(df$DEBTINC))

# Nominal
sum(df$REASON=="")
table(df$REASON)
sum(df$JOB=="")
table(df$JOB)
```

- b. Impute missing values with the mean for interval variables and the most frequent level for categorical variables.

```
# Impute Interval
df$MORTDUE[is.na(df$MORTDUE)] = mean(df$MORTDUE, na.rm=T)
df$VALUE[is.na(df$VALUE)] = mean(df$VALUE, na.rm=T)
df$YOJ[is.na(df$YOJ)] = mean(df$YOJ, na.rm=T)
df$DEROG[is.na(df$DEROG)] = mean(df$DEROG, na.rm=T)
df$DELINQ[is.na(df$DELINQ)] = mean(df$DELINQ, na.rm=T)
df$CLAGE[is.na(df$CLAGE)] = mean(df$CLAGE, na.rm=T)
df$NINQ[is.na(df$NINQ)] = mean(df$NINQ, na.rm=T)
df$CLNO[is.na(df$CLNO)] = mean(df$CLNO, na.rm=T)
df$DEBTINC[is.na(df$DEBTINC)] = mean(df$DEBTINC, na.rm=T)
```

```
# Impute Nominal
df$REASON[df$REASON==""] =
names(which(table(df$REASON)==max(table(df$REASON))))
df$JOB[df$JOB==""] =
names(which(table(df$JOB)==max(table(df$JOB))))

# Missing?
sum(is.na(df) | df=="") # 0
```

c. Partition the data into training (60%), validation (30%), and test (10%) sets.

```
# Partition Data
nr = nrow(df)
set.seed(802)
rows = sample(c(rep(1,nr*.6), rep(0,nr*.3), rep(2, nr*.1)))
train_rows = which(rows==1)
valid_rows = which(rows==0)
test_rows = which(rows==2)

df_train = df[train_rows,]
df_valid = df[valid_rows,]
df_test = df[test_rows,]
```

d. Create an open source logistic regression model.

```
# Logistic Regression Model
mymod = glm(BAD ~ LOAN + MORTDUE + VALUE + REASON + JOB + YOJ +
            DEROG + DELINQ + CLAGE + NINQ + CLNO + DEBTINC,
            data=df_train, family = "binomial")
```

e. Convert the model to PMML and save as an XML file.

```
# Markup
library(pmml)
saveXML(pmml(mymod), "D:/Workshop/Winsas/UA19DMMM
/R_Logistic_Regression_Model.xml")
```

Note: The **pmml** XML file is saved in 4.3. SAS Model Manager requires 4.2 to import the model. The version can be manually changed by opening the XML file and editing the heading to the appropriate version.

3. Create a JSON file containing summary statistics for the model.

a. Load the gradient boosting model **dmcas_fitstat** JSON file as a template.

```
library(rjson)
json = fromJSON(file = "D:/Workshop/Winsas/UA19DMMM
/dmcas_fitstat_sas.json")
```

b. Calculate the average squared error for the R model.

```
# Calculate ASE statistics
train_probs = predict(mymod, df_train, type = "response")
valid_probs = predict(mymod, df_valid, type = "response")
test_probs = predict(mymod, df_test, type = "response")
```

```
train_ASE = mean((df_train$BAD - train_probs)^2)
valid_ASE = mean((df_valid$BAD - valid_probs)^2)
test_ASE  = mean((df_test$BAD - test_probs)^2)
```

c. Calculate the C-statistic for the R model.

```
# Calculate C-statistics
library(pROC)

train_C = as.numeric(gsub("Area under the curve: ", "",
                          auc(df_train$BAD, train_probs)))
valid_C = as.numeric(gsub("Area under the curve: ", "",
                          auc(df_valid$BAD, valid_probs)))
test_C  = as.numeric(gsub("Area under the curve: ", "",
                          auc(df_test$BAD, test_probs)))
```

d. Add the calculated summary statistics to the JSON template file.

```
# Test
json$data[[1]]$dataMap$_RASE_ = paste0(round(sqrt(test_ASE),4))
json$data[[1]]$dataMap$_ASE_  = paste0(round(test_ASE,4))
json$data[[1]]$dataMap$_C_    = paste0(round(test_C,4))

#Train
json$data[[2]]$dataMap$_RASE_ =
paste0(round(sqrt(train_ASE),4))
json$data[[2]]$dataMap$_ASE_  = paste0(round(train_ASE,4))
json$data[[2]]$dataMap$_C_    = paste0(round(train_C,4))

# Valid
json$data[[3]]$dataMap$_RASE_ =
paste0(round(sqrt(valid_ASE),4))
json$data[[3]]$dataMap$_ASE_  = paste0(round(valid_ASE,4))
json$data[[3]]$dataMap$_C_    = paste0(round(valid_C,4))
```

e. Save the JSON file as **dmcas_fitstat.json**.

```
# Export to json file
export_json = rjson::toJSON(json, indent=1)
write(export_json, "D:/Workshop/Winsas/UA19DMMM
                  /dmcas_fitstat.json")
```

4. Create a JSON file that contains ROC information for the model.

a. Load the gradient boosting model **dmcas_roc** JSON file as a template.

```
library(rjson)
json = fromJSON(file = "D:/Workshop/Winsas/UA19DMMM
                      /dmcas_roc_sas.json")
```

b. Remove the validation and test ROC data from the JSON file.

```
# Remove the Validation and test ROC data
for(i in rev(23:length(json$data))){
  json$data[[i]]=NULL
}
```

```
}

```

c. Create confusion matrices for the model.

```
# Get Predictions
preds = predict(mymod,df_train,type = "response")
target = df_train$BAD

# Create Confusion Matrices to get ROC curve information
myseq = seq(.05,.95,.05)
sens = NULL
spec = NULL

for(i in 1:length(myseq)){
  temp = (preds>myseq[i])*1
  tab = table(target,temp)
  sens = c(sens, tab[2,2]/(tab[2,2]+tab[2,1]))
  spec = c(spec, tab[1,1]/(tab[1,1]+tab[1,2]))
}

myseq = c(0,myseq,1)
sens = c(1,sens,0)
spec = c(0,spec,1)
plot(1-spec,sens)

```

d. Add ROC information to the JSON template.

```
for(i in 1:21){
  json$data[[i]]$dataMap$`_OneMinusSpecificity_` =
    paste0(1-spec[i])
  json$data[[i]]$dataMap$`_Sensitivity_` = paste0(sens[i])
  json$data[[i]]$dataMap$`_Specificity_` = paste0(spec[i])
}

```

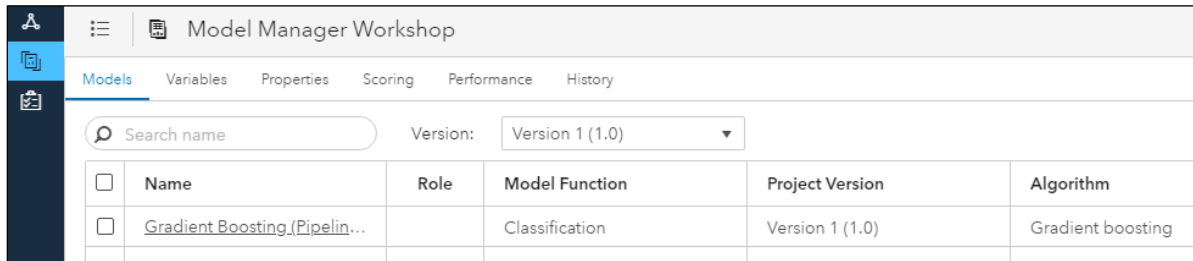
e. Save the JSON file as **dmcas_roc.json**.

```
# Export to json file
export_json = rjson::toJSON(json, indent=1)
write(export_json, "D:/Workshop/Winsas/UA19DMMM/dmcas_roc.json")

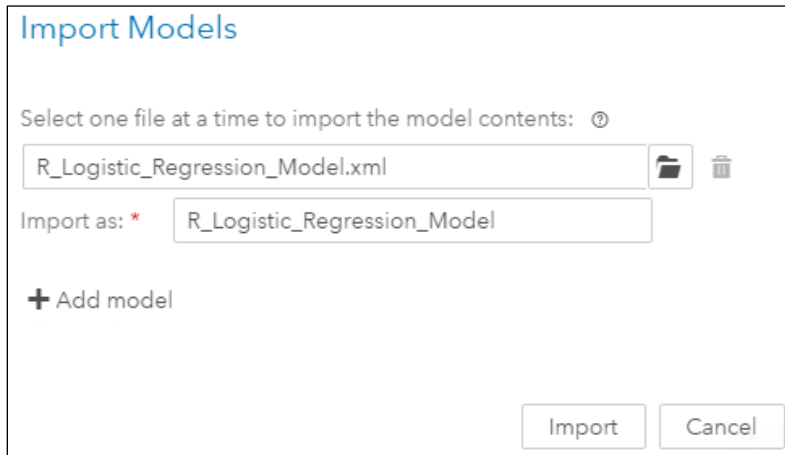
```

Import the Open Source Model

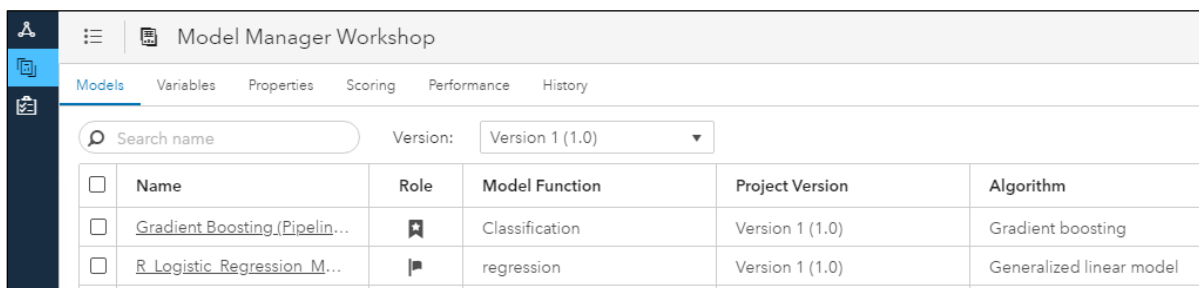
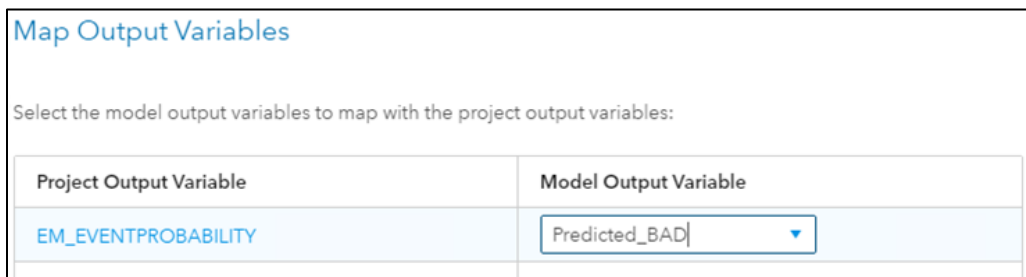
1. From the Model Manager Workshop project Models tab, click the **Import** tab and then click **Import**.



2. Open the **R_Logistic_Regression_Model.xml** file in the course folder and then click **Import**.



3. Select the box next to the open source model in the project window. Then click the **Actions** icon and click **Set as challenger**. In the Map Output Variables window, map the **EM_EVENTPROBABILITY** project output variable to the model output variable **Predicted_BAD** and click **OK**.



- To ensure that the open source model has been imported correctly, go to the Scoring tab of the project window and click **New Test**. In the New Test window, change the test name to **Open Source Model Import Test**, and in the **Model** field, select **R_Logistic_Regression_Model**. Next, open the **Data source** field and select the **HMEQ** data set. Click the **Sample Data** tab to test only a sample of 100 observations. Click **OK** ⇨ **Run**.

The top screenshot shows the 'Choose a Data Source' dialog with the 'HMEQ' data source selected. The 'Sample rows' field is set to 100. Below the dialog, a table displays the first row of data from the HMEQ dataset:

BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELI
1	1100	25860	39025	Homelmp	Other	10.5	0	

The bottom screenshot shows the 'Model Manager Workshop' interface with the 'Scoring' tab selected. A table lists the test results:

Name	Results	Status	Model Name	Project Version	Input Table
Open Source M...		✓	R_Logistic_Regres sion_Model (1.0)	Version 1 (1.0)	HMEQ

Import the Open Source Model Summary Statistics and Compare Models

- Go to the Models tab of the project window and select the open source model.


The screenshot shows the 'R_Logistic_Regression_Model (1.0)' model details. The 'Files' tab is active, showing a list of files: score.sas, R_Logistic_Regression_Model.xml, input.xml, and output.xml. The 'score.sas' script is displayed in the editor:

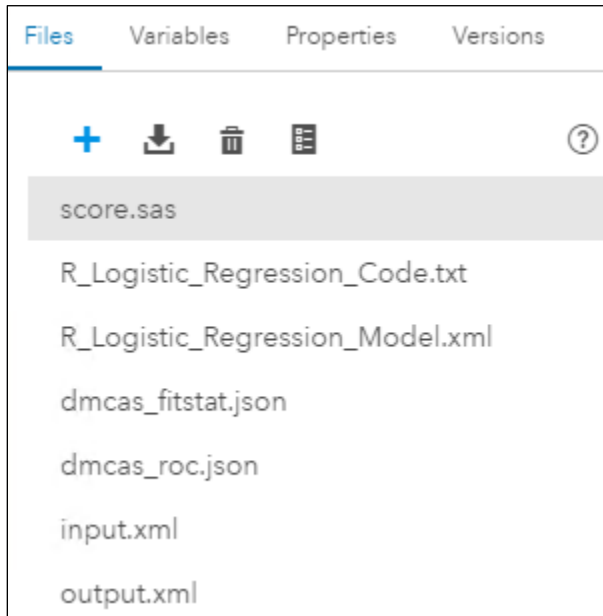
```

1  /*****/;
2  * PSCORE TIMESTAMP: 2019-6-8 15:48:10.28 ;
3  * SAS VERSION: V.03.04M0P050819 ;
4  * SAS HOSTNAME: intviya01 ;
5  * SAS ENCODING: utf-8 ;
6  * SAS USER: student ;
7  * SAS LOCALE: EN_US ;
8  * PMML Path: R_Logistic_Regression_Model.xml ;
9  * PMML SOURCE: SoftwareAG PMML Generator;
10 * PMML SOURCE VERSION: 1.4;
11 * PMML TIMESTAMP: 2019-08-06 15:36:19 ;
12 * MODEL TYPE: GeneralRegressionModel ;
13 * MODEL FUNCTION NAME: Regression ;
14 /*****/;
15
16
17 PSCR_WARN = 0;
18
19 if missing("LOAN"n) then do;
20     PSCR_WARN = 1;
21 end;

```

Notice that the XML file was imported and the PMML code was converted to SAS DATA step score code using the PSCORE procedure and saved as the **score.sas** file. This file can now be used to score new data here in Model Manager or after deployment. The gradient boosting model contains Astore files for scoring new data as well as JSON files that contain model summary information.

- Click the **Add** button  on the Files tab and import the **R_Logistic_Regression_Code.txt**, **dmcas_fitstat.json**, and **dmcas_roc.json** files. Click **Add**. This imports summary information created in R in order to compare with other models within Model Manager.

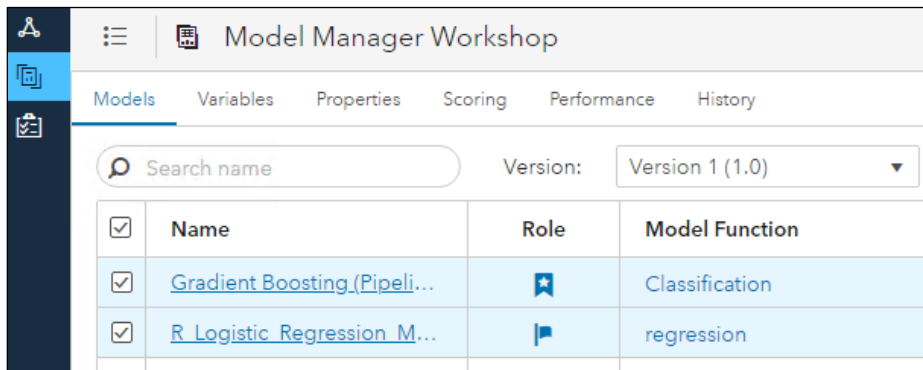


- Scroll through the **dmcas_fitstat.json** file to view its contents. Notice that the ASE, RASE, and C fields were specified using the R model.

```
dmcas_fitstat.json (Read-Only)
[Copy] [Print] [Refresh]
186     "data": [
187     {
188         "dataMap": {
189             "_RASE_": "0.3478",
190             "_MiscCutoff_": "0.0939597315436",
191             "_NObs_": "596",
```

- Click the project window to return to the Model Manager Workshop project window.

- Select the boxes next to both models and then click **Compare** in the top right.



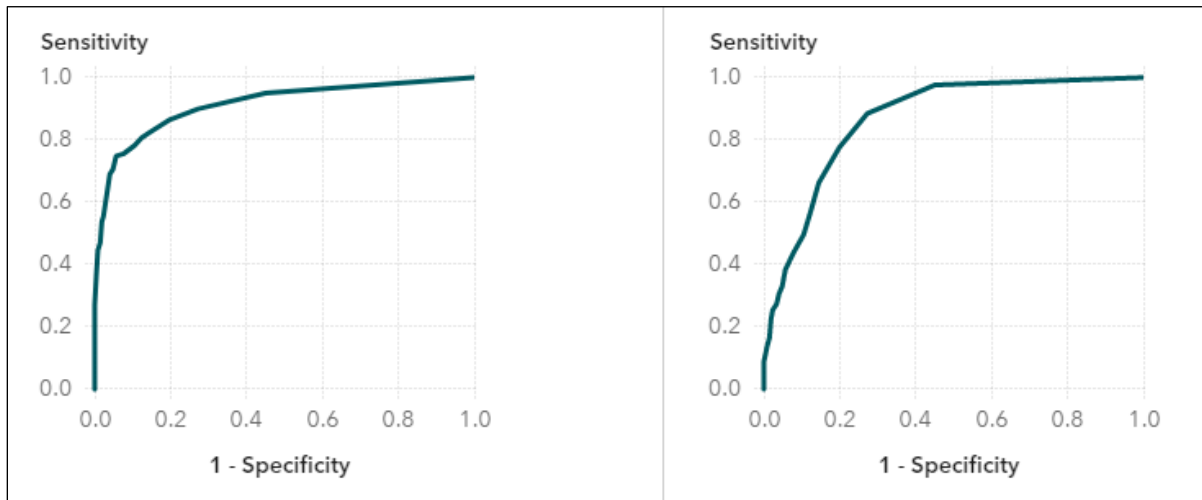
- Scroll through the Compare window to view numeric and graphical summaries.

Model Manager Workshop > Compare

Show All Show Differences

Models	Gradient Boosting (Pipeline 1)	R_Logistic_Regression_Model
TEST		
RASE	0.2762411067	0.3478
NObs	596	596
GINI	0.8247626094	0.8247626094
GAMMA	0.86444656178	0.86444656178
MCE	0.0939597315	0.0939597315
ASE	0.076309149	0.121
MCLL	0.2627776	0.2627776
KS	0.6912953861	0.6912953861
KSPostCutoff	0.6492433451	0.6492433451
DIV	596	596
TAU	0.2640347414	0.2640347414
KSCut	0.4	0.4
C	0.9123813047	0.8049
PartInd	2	2

The comparison of the ASE, RASE, and C-statistic for the Train, Validate, and Test partitions all show that the gradient boosting model outperforms the R model for this project.



In conjunction with the C-statistic, the ROC curve for the gradient boosting model (left) seems to show more area under the curve than the R model (right).

Note: The ROC plot compares the models for only the training data in this Model Manager comparison tool.

End of Demonstration



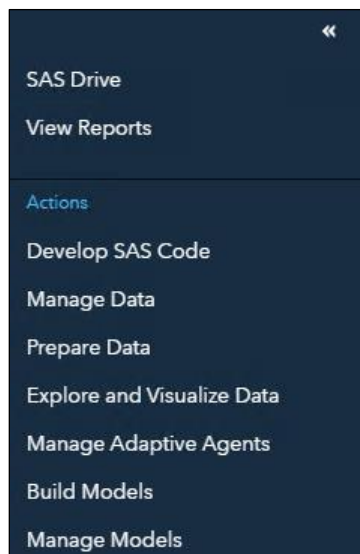
Monitoring Champion Model Performance

Thus far, the models have been built on the **HMEQ** baseline data set. Five additional data sets were collected over subsequent quarters to monitor the champion model performance and look for possible degradation. Note that to monitor performance on multiple data sets, they must be named in one of the following ways:

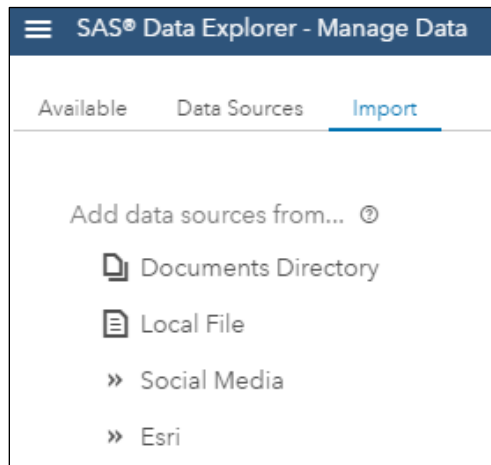
- 1: prefix_sequenceNumber_timeLabel
- 2: prefix_sequenceNumber_timeLabel_modelIIDUID
- 3: prefix_sequenceNumber_timeLabel_modelIIDUID_modelRole

HMEQ	Baseline Home Equity Data Set for Model Studio
HMEQ_1_Q1	Home Equity Data Collected in Q1 of 2018
HMEQ_2_Q2	Home Equity Data Collected in Q2 of 2018
HMEQ_3_Q3	Home Equity Data Collected in Q3 of 2018
HMEQ_4_Q4	Home Equity Data Collected in Q4 of 2018
HMEQ_5_Q5	Home Equity Data Collected in Q1 of 2019

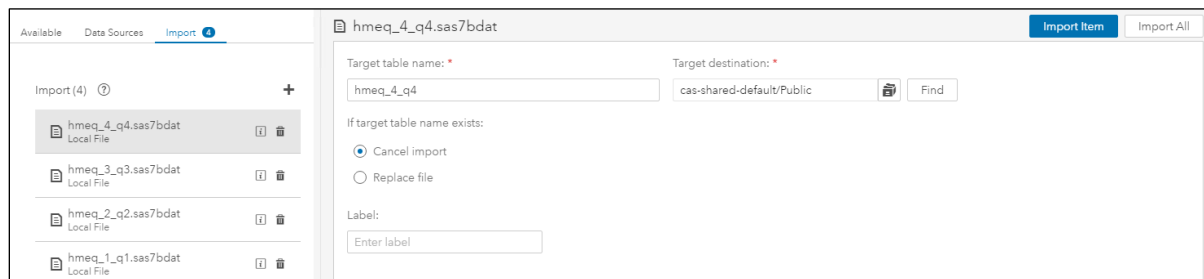
1. Open the **Show applications** menu  in the top left and select **Manage Data**.



- Click the **Import** tab and add a data source from a local file.



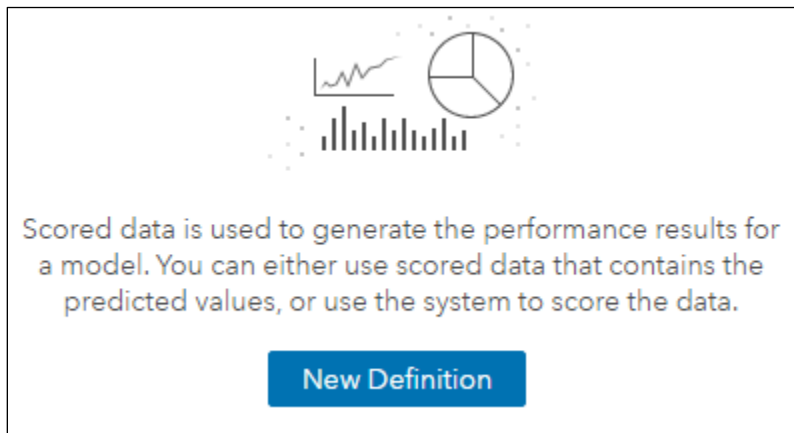
- Import **hmeq_1_q1.sas7bdat**, **hmeq_2_q2.sas7bdat**, **hmeq_3_q3.sas7bdat**, and **hmeq_4_q4.sas7bdat**. Click **Import All**.



- Open the **Show applications** menu  in the top left and select **Manage Models**.
- In the Model Manager Workshop project window, click **Performance**.

Name	Role	Model Function	Project Version	Algorithm
Gradient Boosting (Pipel...		Classification	Version 1 (1.0)	Gradient boosting
R Logistic Regression M...		regression	Version 1 (1.0)	Generalized linear model

6. Click **New Definition**.



7. From the New Performance Definition page, select **Use a Library that contains tables with a specified prefix**. Change the caslib (**CAS library** field) to **Public** and set the prefix to **hmeq**. Click **OK**.

The image shows a dialog box titled "New Performance Definition" in blue text. The "Data scoring method:" dropdown menu is set to "System scores data". Under "Input data method: @", the radio button for "Use a library that contains tables with a specified prefix" is selected. The "CAS library:" field contains "cas-shared-default/Public" and the "Prefix: *" field contains "hmeq". There are "OK" and "Cancel" buttons at the bottom right.

8. From the Definition page, set the name to **Model Performance Test**. In the **Models** section, select **Use Referenced Models** and then select the current champion, which is the gradient boosting model. The rest of the fields have been populated with project settings information. Click the **Save** icon in the top right.

Model Manager Workshop > Definition

▼ Details - System Scores Data

Name: *

Model Performance Test

Description:

Note: Each time the definition is run the results are appended to the output data tables. After you save the definition and are returned to the Performance tab, you can click Clear All to start over.

▼ Tables*

The tables in the selected library whose names contain the specified prefix, are used when generating the performance results.

CAS library:

cas-shared-default/Public

Prefix:

hmeq

▼ Models*

Select models from this project

Choose Models

Use referenced models ⓘ

Current champion: Gradient Boosting (Pipeline 1) (1.1)

All challengers: R_Logistic_Regression_Model (1.0)

▼ Report Settings*

Input variables for the deviation report:

CLAGE,CLNO,DEBTINC,DELINQ,DEROG,JOB,LOAN,MORTDUE,NINQ,REASON,VALUE,YOJ Choose Variables

Output variables for the stability report:

EM_EVENTPROBABILITY Choose Variables

▼ Project Properties

Target variable:

Target event value:

Target level:

Output event probability variable:

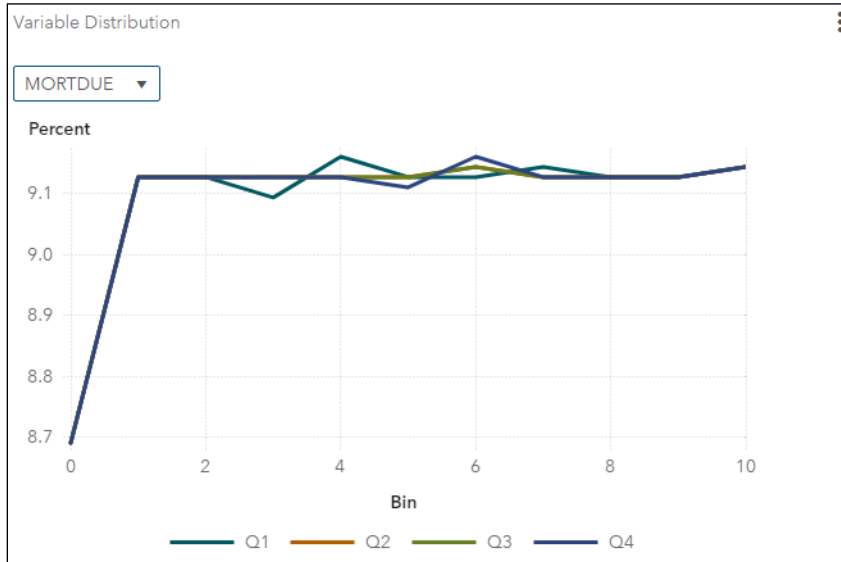
▼ Output Settings*

CAS library for output tables:

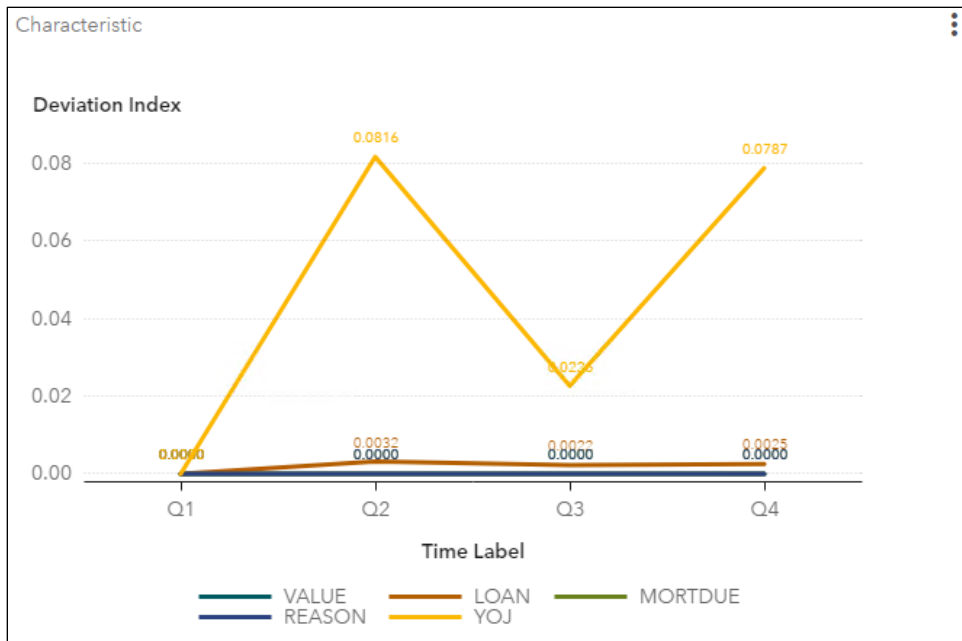
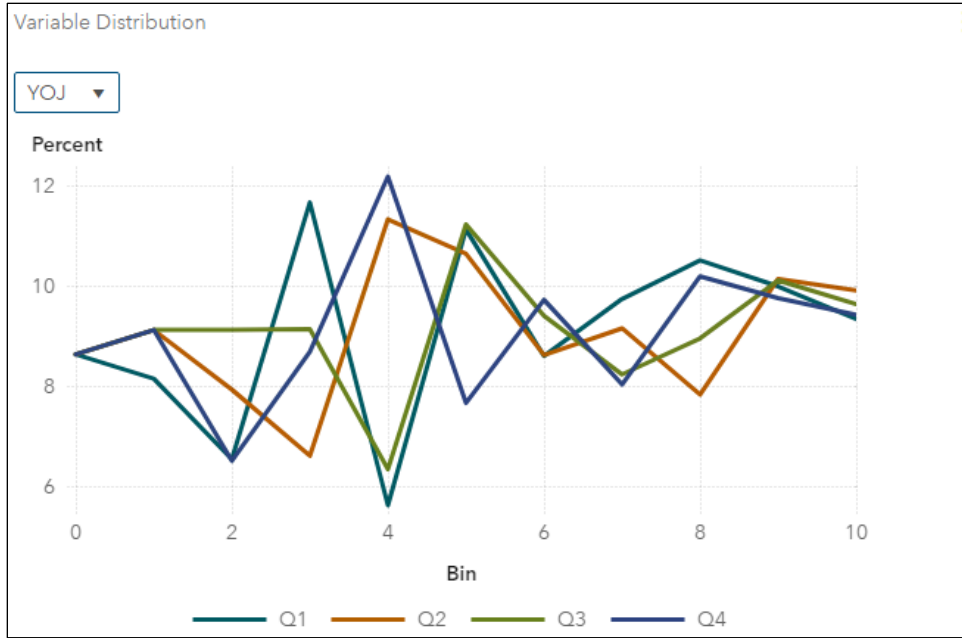
9. Click **Run** on the Performance tab to run the new definition.



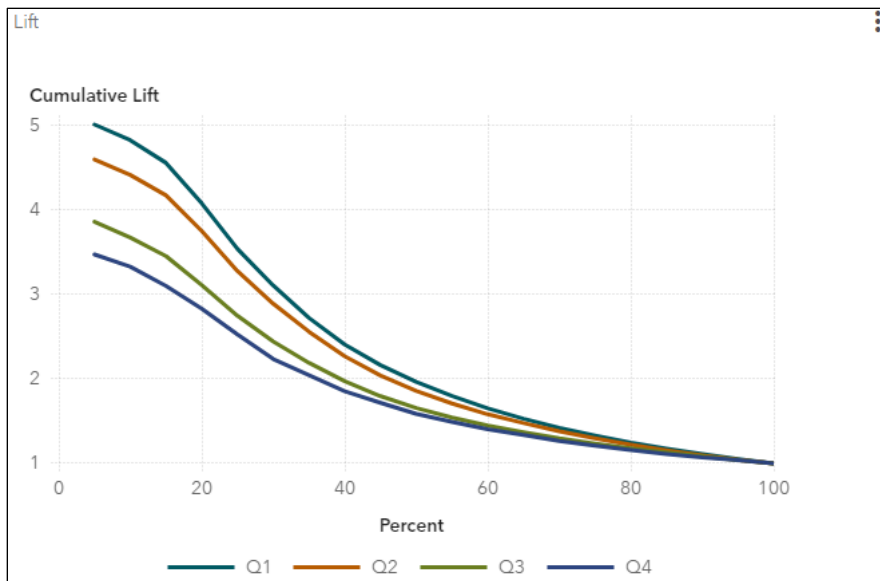
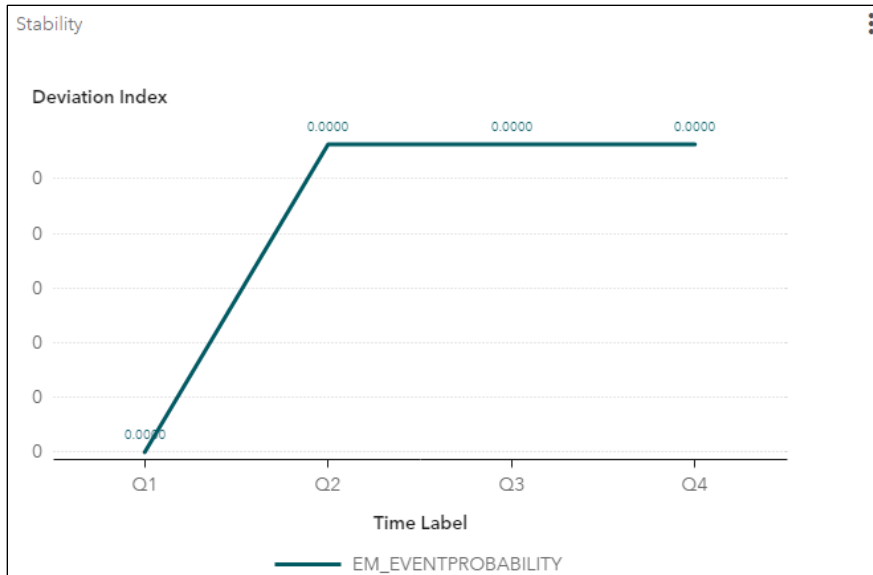
10. In the Variable Distribution window, select **MORTDUE**. After viewing the graphic, select **YOJ**. View the rest of the plots.



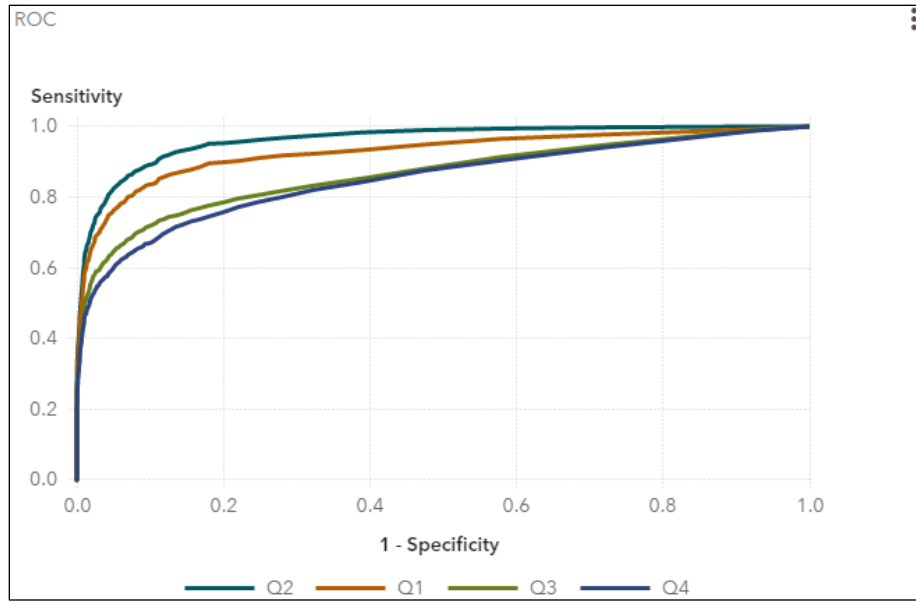
The Variable Distribution chart is a graphical representation of distributions over a period of time for the selected variable. Each line represents the data for a specified period of time. The Y axis is the percentage of observations in a bin that is proportional to the total count. As you can see, the **MORTDUE** variable distribution is relatively stable over the one-year period and unlikely contributing to any model degradation. The **YOJ** variable distribution on the other hand is much more erratic and therefore contributes more greatly to model degradation.



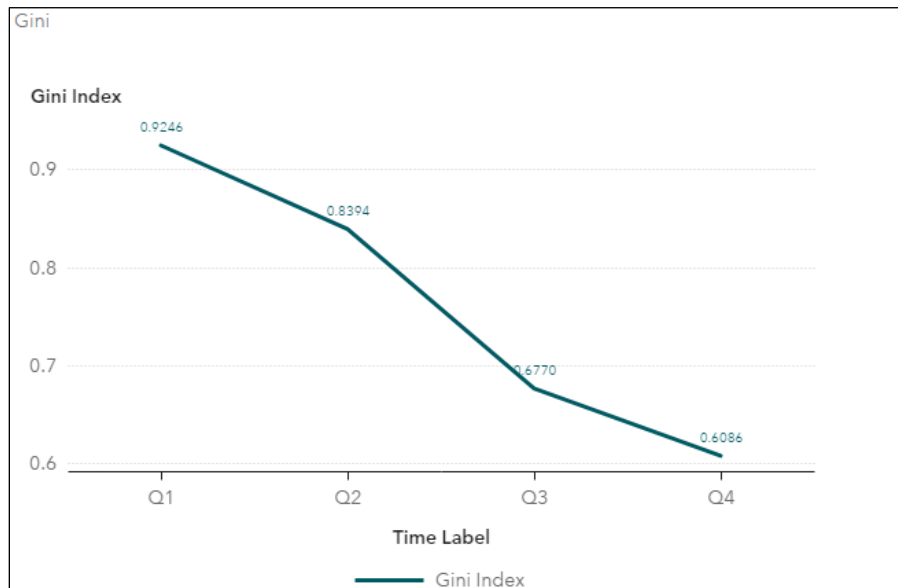
The Stability report evaluates changes in the distribution of scored output variable values as models score data over time, and it detects and quantifies shifts in the distribution of output variable values in the data that are produced by the models. If an output variable from the training data set and the output variable from the current data set have identical distributions, then that output variable's deviation index is equal to 0. An output variable with a deviation index value that is greater than 0.10 and less than 0.25 is classified as having a *mild deviation*. A variable that has a deviation index value that is greater than 0.30 is classified as having a *significant deviation*. Too much deviation in predictive variable output can indicate that model tuning, retraining, or replacement might be necessary.



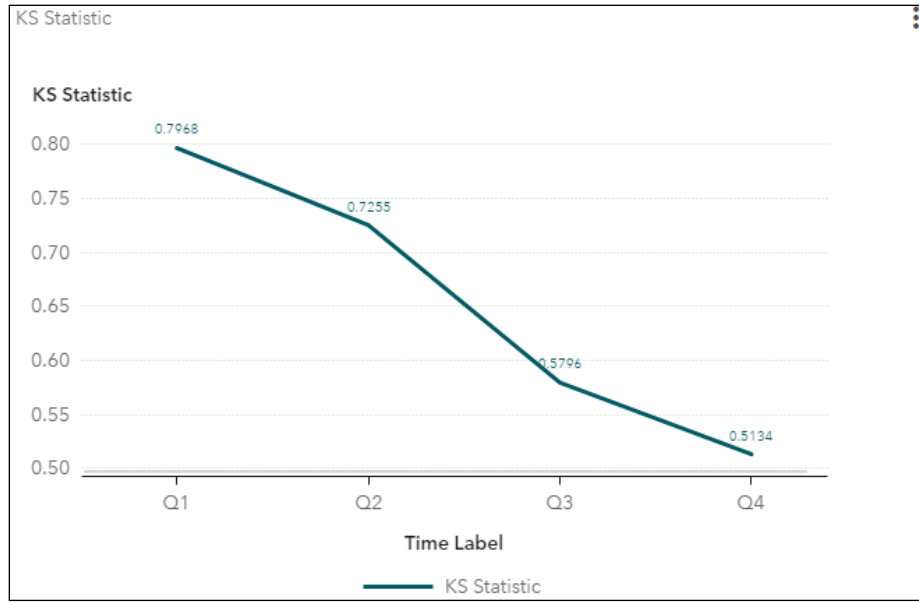
The Lift report provides a visual summary of the usefulness of the information that is provided by a model for predicting a binary outcome variable. Specifically, the report summarizes the utility that you can expect by using the champion model as compared to using baseline information only. Baseline information is the prediction accuracy performance of the initial performance monitoring definition or batch program using operational data. In this case, we can see clear degradation of the gradient boosting model over the one-year period as each lift curve over time becomes shallower.



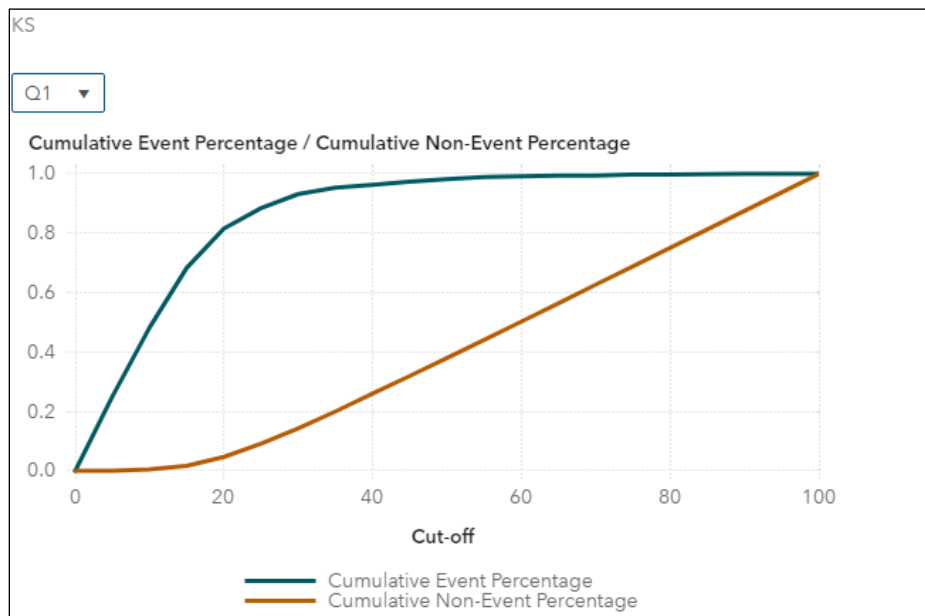
The ROC plot shows similar degradation behavior of the model over time as the area under the curve is greatest in the first half of the year and lowest in the second half of the year.

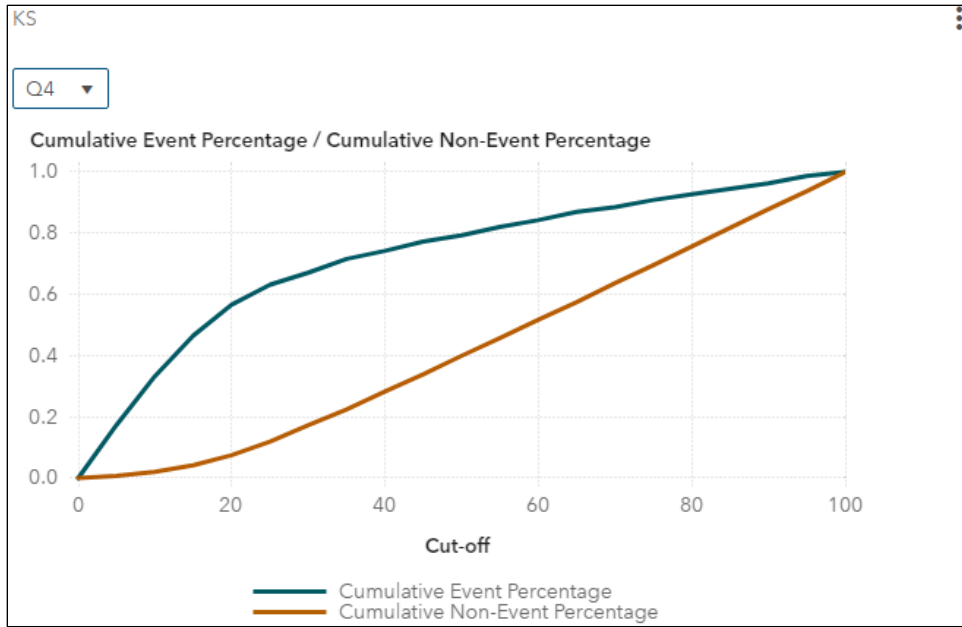


The Gini index is calculated for each ROC curve. The Gini coefficient is a benchmark statistic that can be used to summarize the predictive accuracy of a model and is directly related to the area under the ROC curve ($2 \cdot \text{AUC} - 1$). Again, the Gini index is another tool that demonstrates model degradation.



The KS report contains the Kolmogorov-Smirnov (KS) test plots for models with a binary target. The KS statistic measures the maximum vertical separation, or deviation between the cumulative distributions of events and non-events. This trend report uses a summary data set that plots the KS statistic and the KS probability cutoff values over time.





End of Demonstration



Retraining the Champion Model and Deploying to MAS

Given that the model has degraded across four data sets equating to one year of time, we can retrain the model and deploy the new and improved model.

1. From the Models tab of the Model Manager Workshop project, select the box next to the champion model and then click the **Actions** icon in the top right. From the Actions list, click **Retrain**.

Name	Role	Model Function	Project Version	Algorithm	Date Modified	Modified By
<input checked="" type="checkbox"/> Gradient Boosting (Pipeline...)	📌	Classification	Version 1 (1.0)	Gradient boosting	Aug 6, 2019 01:34 PM	student

2. From the Retrain Project window, select **Retrain now with a new data source**. From the Choose a Data Source window, select **hmeq_4_q4.sas7bdat** and click **OK**. Then click **Send** in the Retrain Project window to retrain the gradient boosting model in Model Studio.


Retrain Project

Select an option and send a request to the Model Studio application for retraining.

Set the project retrain state to needed
 Retrain now with a new data source

cas-shared-default/Public/HMEQ_4_Q4

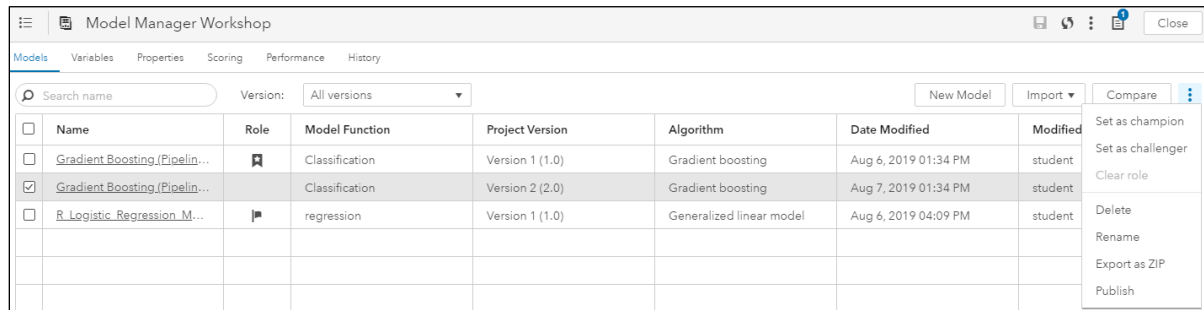
Send **Cancel**

3. From the Model Manager Workshop project window, change the version view to **All Versions**, and then refresh the page . Note that the process of retraining the model takes a moment. Continue refreshing until it appears.

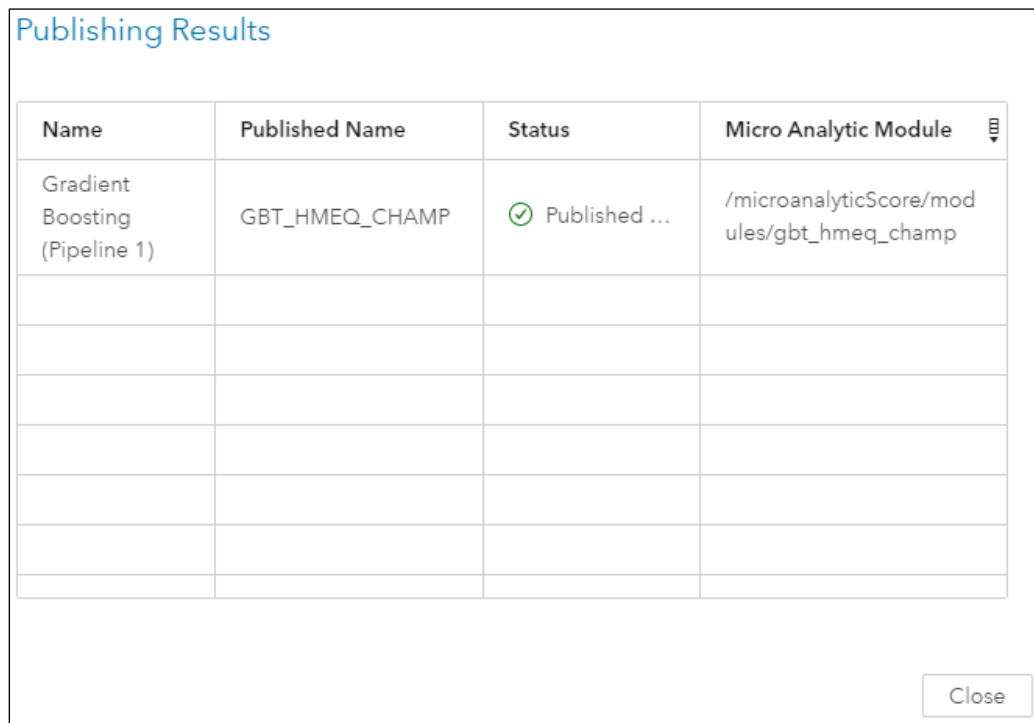
Name	Role	Model Function	Project Version	Algorithm
<input type="checkbox"/> Gradient Boosting (Pipeline...)	📌	Classification	Version 1 (1.0)	Gradient boosting
<input type="checkbox"/> Gradient Boosting (Pipeline...)		Classification	Version 2 (2.0)	Gradient boosting
<input type="checkbox"/> R Logistic Regression M...	📌	regression	Version 1 (1.0)	Generalized linear model

The champion model has been retrained in Model Studio using the Q4 data and registered to Model Manager in a new project version.

- From the Models tab of the project window, select the box next to the version 2 gradient boosting model. Then click **Publish** from the Actions list.

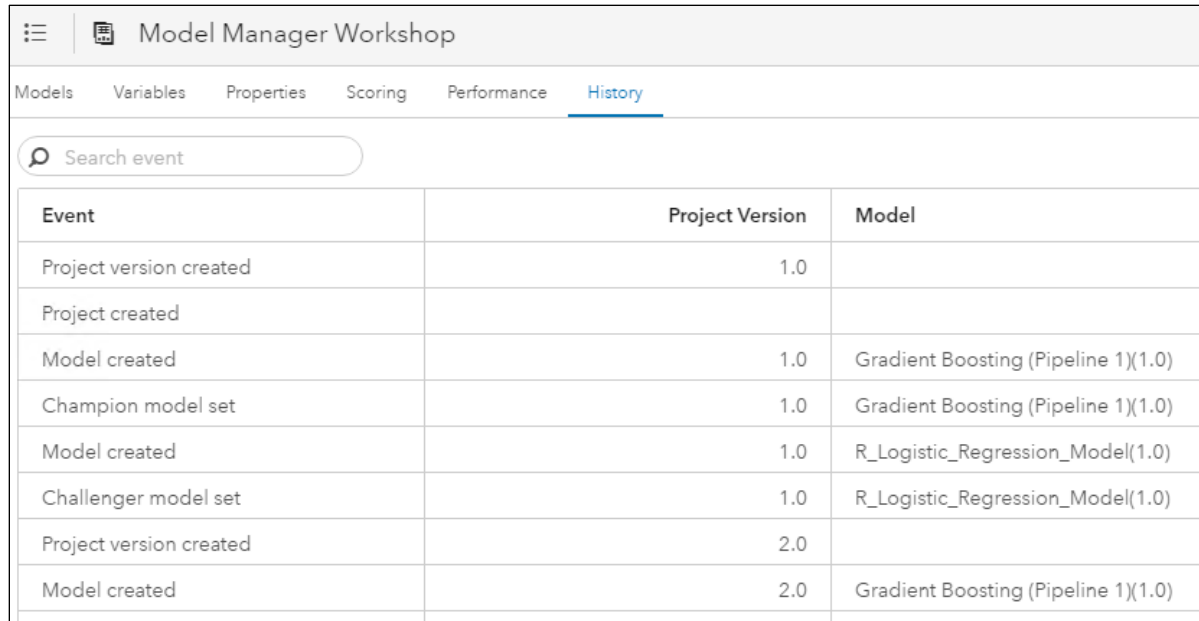


- In the Publish Models window, change the published name to **GBT_HMEQ_CHAMP**. Then click **Publish**.



Models can be published to CAS, Hadoop, Teradata, and SAS Micro Analytic Services depending on the score code type. For the SAS gradient boosting model, by default, the score code type is DS2 multi-type, which uses DS2 to create and push an Astore file to the required location. If we publish the open source model instead, the SAS DATA step score code saved in **score.sas** would be published for scoring in the desired location.

- Finally, click the **History** tab to view details of the project sequence.

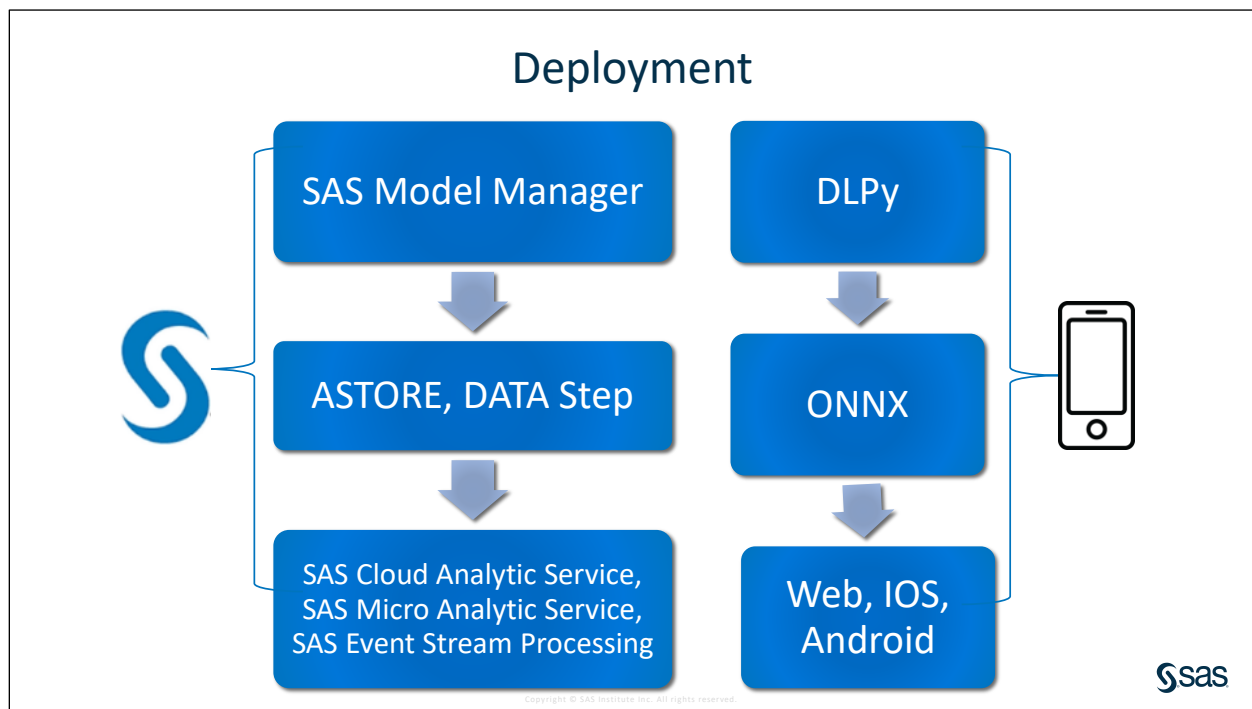


The screenshot shows the 'Model Manager Workshop' interface with the 'History' tab selected. A search bar labeled 'Search event' is present above a table. The table has three columns: 'Event', 'Project Version', and 'Model'. The events listed are: 'Project version created' (1.0), 'Project created', 'Model created' (1.0) for 'Gradient Boosting (Pipeline 1)(1.0)', 'Champion model set' (1.0) for 'Gradient Boosting (Pipeline 1)(1.0)', 'Model created' (1.0) for 'R_Logistic_Regression_Model(1.0)', 'Challenger model set' (1.0) for 'R_Logistic_Regression_Model(1.0)', 'Project version created' (2.0), and 'Model created' (2.0) for 'Gradient Boosting (Pipeline 1)(1.0)'.

Event	Project Version	Model
Project version created	1.0	
Project created		
Model created	1.0	Gradient Boosting (Pipeline 1)(1.0)
Champion model set	1.0	Gradient Boosting (Pipeline 1)(1.0)
Model created	1.0	R_Logistic_Regression_Model(1.0)
Challenger model set	1.0	R_Logistic_Regression_Model(1.0)
Project version created	2.0	
Model created	2.0	Gradient Boosting (Pipeline 1)(1.0)

End of Demonstration

1.3 Deploying Models



SAS streamlines the model building, management, and deployment steps required to put an organization’s analytics into production. Depending on the model, SAS publishes the deployed model as an Analytical store or SAS DATA step score code to SAS Cloud Analytic Services (CAS), SAS Micro Analytic Service, or SAS Event Stream Processing.

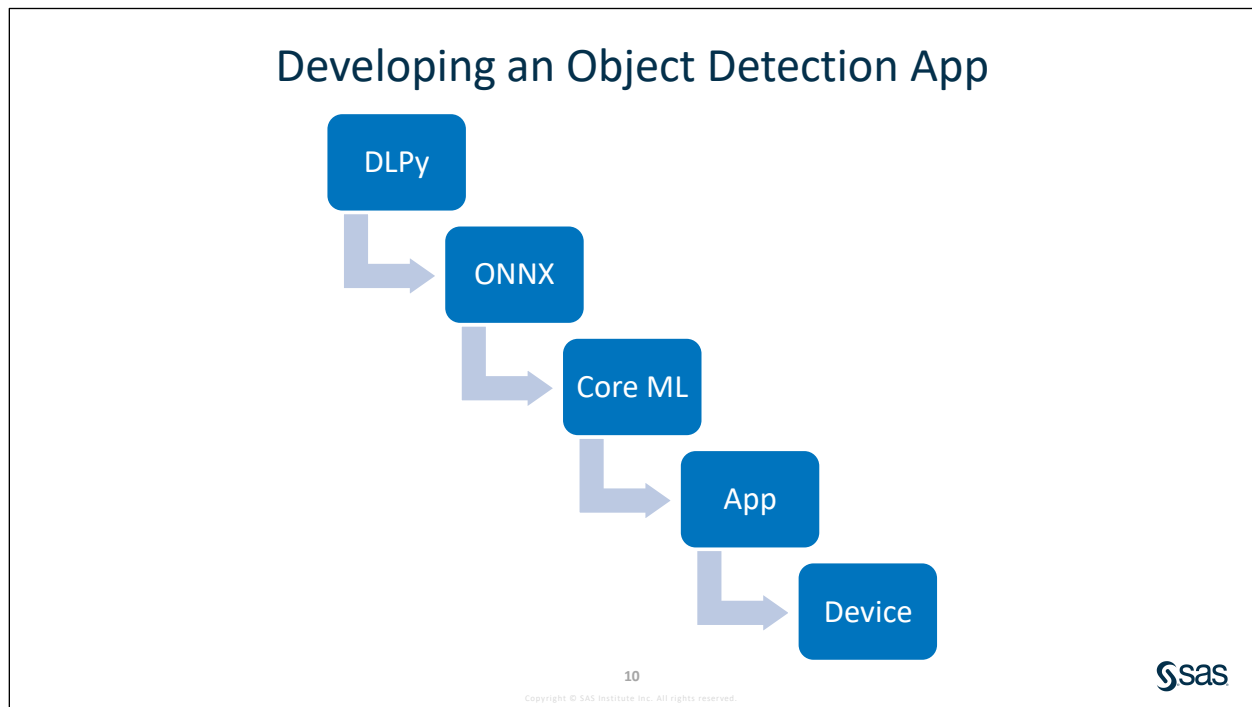
SAS Micro Analytic Service is a memory-resident, high-performance program execution service. As a SAS Platform service, it is not available for individual license, but it is included in selected SAS solutions. SAS Micro Analytic Service provides hosting for DS2 and Python programs and supports a “compile-once, execute-many-times” usage pattern. SAS Micro Analytic Service is multi-threaded and can be clustered for high availability. It can host multiple programs simultaneously, as well as multiple user or business contexts that are isolated from one another.

SAS Event Stream Processing is a form of complex event processing technology that is often used in data and decision applications. It analyzes and processes large volumes of streaming data quickly, helping you analyze events in motion—even as they are generated. Instead of storing data and running queries against it, the SAS Event Stream Processing Engine stores the queries and streams data through them. In this way, continuous analysis of data occurs as the data are received, updating the intelligence as new events occur. Incoming data are read through adapters that are part of a publish-and-subscribe architecture used to read data feeds. Objects written in XML, DS2, Python, and C++ are used to model business logic within the engine. Data that pass through the engine is examined for patterns and can be filtered to more permanent storage. Prebuilt high-performance adapters publish filtered data to other downstream applications. The SAS Event Stream Processing Engine uses a flexible threaded processing model for submillisecond response and high-volume throughput.

Models built in SAS Viya can also be deployed outside the SAS Platform to score data using websites, apps, or other devices. For example, to deploy SAS deep learning models to an IOS or Android app, models can be developed in DLPy and exported to an ONNX file, which can then be converted to readable score code by the app.

DLPy is a high-level Python library for the SAS Deep Learning features available in SAS Viya. DLPy is designed to provide an efficient way to apply deep learning methods to image, text, and audio data. DLPy APIs are created following the [Keras](#) APIs with a touch of [PyTorch](#) flavor. That is, DLPy has the look and feel of Python programming but runs SAS Viya distributed algorithms.

“Open Neural Network Exchange (ONNX) is an open format to represent deep learning models” (<https://onnx.ai/>). Models exported to ONNX are readable by a large community of AI developers, including Android and IOS technologies.



SAS used the PASCAL data set to train a YOLO model to detect 20 different objects. The model was developed in DLPy and exported as an ONNX file. This was then converted to Core ML (Apple machine learning technology) to follow IOS format standards for model deployment. The app is then downloaded on the iPad where the model is used to score new data.

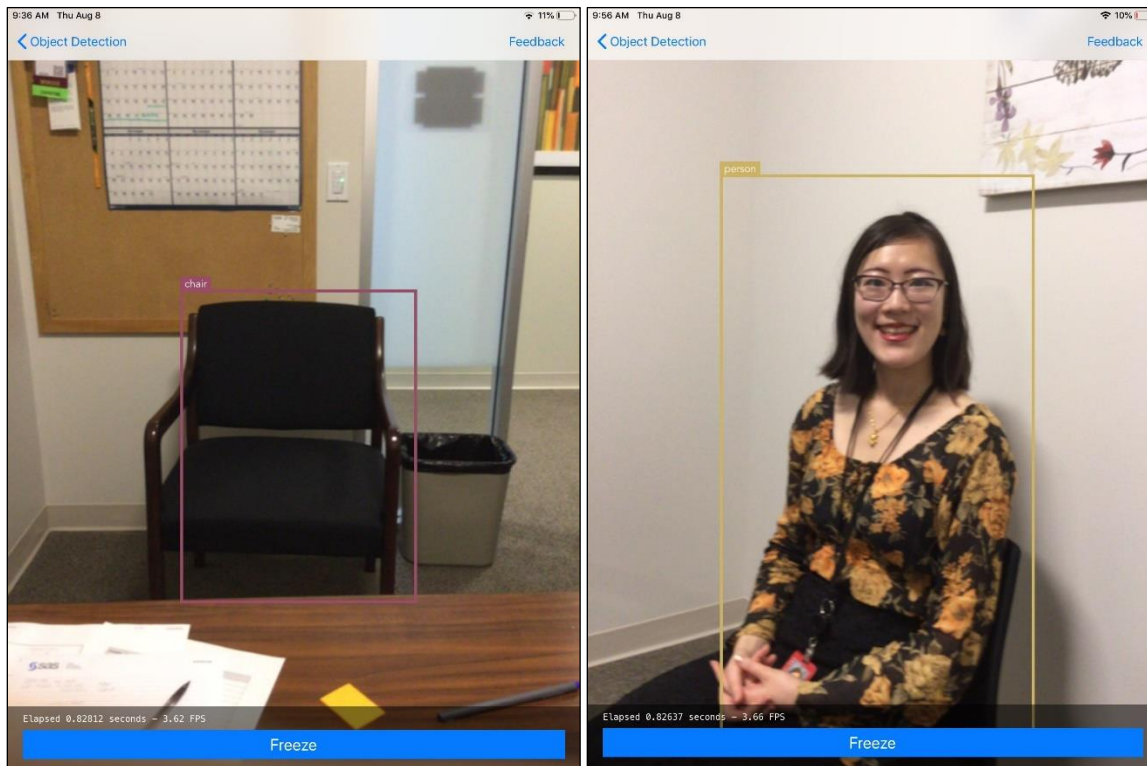


AI Object Detection

In the last few years, the field of machine learning has made tremendous progress in the area of computer vision. Object detection answers the question of **what** is in the image and **where** it is located in the image (object detection = identification + coordinates). YOLO (you only look once) is a common method used in object detection, and it leverages deep learning.

In this demonstration, we are using a YOLO model trained in SAS to detect any of 20 objects (PASCAL visual object classes data set) including person, bird, cat, cow, dog, horse, sheep, airplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, and TV/monitor. A bounding box is drawn around each identified object in the image.

Below are two examples of the AI Object Detection demonstration from the iPad. As you can see, the picture on the left has a bounding box around the chair in my office and the picture on the right has a bounding box around my colleague Aurora.



End of Demonstration

