



# I Spy with My Little Eye: Open-Source Code in my SAS Model Studio Project!

## SAS Guided Demo | SAS Academic Programs

### Purpose

This SAS Guided Demo strives to get you excited about incorporating your favorite Python and R machine learning models into your SAS Model Studio pipelines. We show you how easy it is to incorporate open-source code – and how SAS + open-source integration leads to more robust modeling predictions!

### Scenario

For this analysis, we pretend that you're a newly hired Retention Data Analyst at iLink Telecom, LLC. Congrats on the new job!

Your goal is to use historical customer cellphone data to identify customers most likely to leave iLink Telecom for another provider. We call this churn... and churn can be bad. Why? Because it's often very expensive to find new customers – and much better to keep our existing base.

So, for this SAS Guided Demo, you are provided a database with the outcome variable of interest – churn – and a bunch of predictive variables. Your goal is to find the best predictive model of churn – using SAS, Python, and R – so that we can eventually use that model to predict new cases.

### Learning Objectives

- Expose you to predictive modeling and machine learning in SAS Model Studio.
- Include R + Python machine learning tools into your SAS Model Studio pipelines.

### Software

This SAS Guided Demo was written utilizing SAS Viya for Learners 4, version 2024.09LTS.

### Prerequisites

Experience with both SAS Model Studio and Python and R code is helpful, but not required.

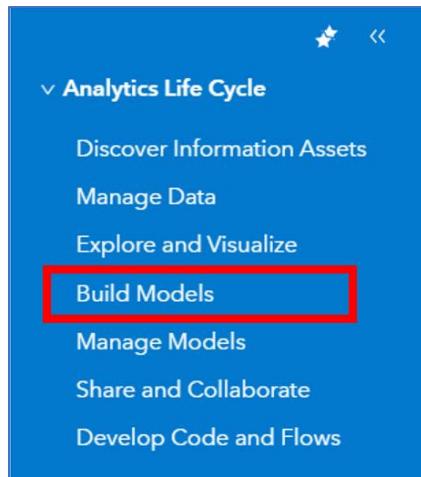


## Table of Contents

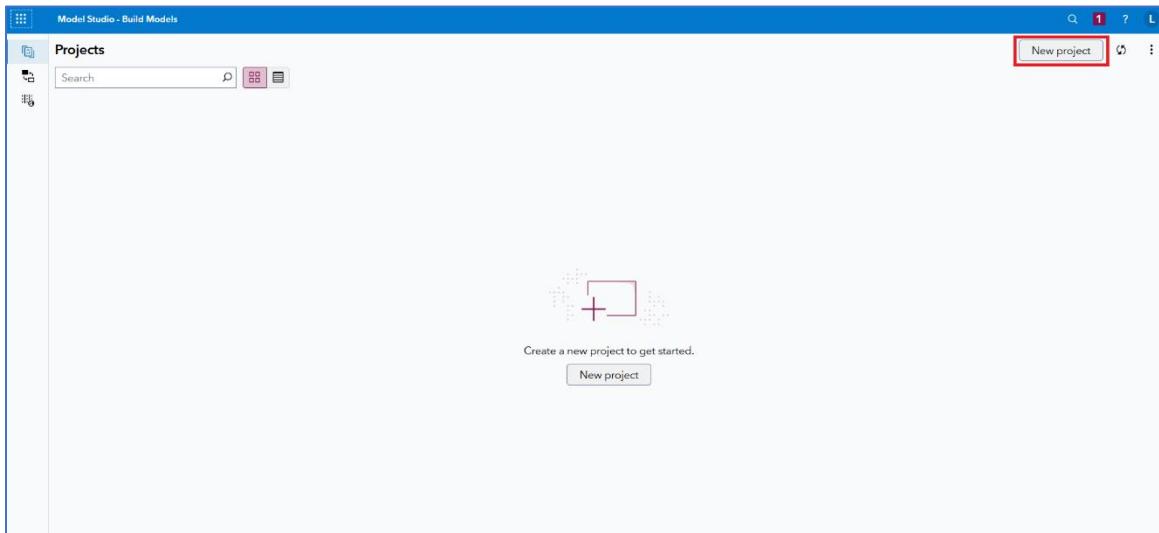
I Spy with My Little Eye: .....	1
Open-Source Code in my SAS Model Studio Project! .....	1
Purpose .....	1
Scenario.....	1
Learning Objectives.....	1
Software.....	1
Prerequisites .....	1
Part 1: Setting up a SAS Model Studio Project .....	3
Part 2: Open-Source Model Integration.....	10
Open Source Code: General Setup.....	10
Model 1: R – Decision Tree .....	15
Model 2: R – Forest .....	23
Model 3: Python – Forest Model .....	26
Model 4: Python – Gradient Boosting.....	30
Model 5: Python – Neural Network .....	33
Part 3: Run a SAS Model Studio Pipeline (for fun) .....	37
Appendix: Pulling Back the Curtain: How I Created (part of) This .....	43
R + Python Code for SAS Model Studio.....	43

## Part 1: Setting up a SAS Model Studio Project

- From the **Applications** menu, select **Build Models**



- Welcome to **SAS Model Studio**! Your environment may already have some shared projects out there. For now, ignore them, and find and click that **New Project** button:



- The **New Project** window will step you through setting up a new project. To start, give it a useful Name, such as *Customer Churn*. Keep the **Type** of project at the default, which is *Data Mining and Machine Learning*. Next, find the **Browse** button, under the Template section. Those first three steps:

**New Project**

Name:*	1
Customer Churn	
Type:*	2
Data Mining and Machine Learning	
Template:	
Blank template	3
Browse	
Data:	
Browse	
Description:	
Advanced	
<b>Save</b>	<b>Cancel</b>

- The **Browse** button opens the prebuilt pipelines included by SAS. And spoiler alert: prebuilt pipelines are awesome! Start simple and scroll down to “Basic template for class target”. Click on then option and then select **OK**:

**Browse Templates**

Template Name	Description	Owner	Last Modified
Basic template for class target 1	Data mining pipeline that contains a Data, Imputation, Logistic Regression, and Model Comparison node connected in a linear flow.	SAS Pipeline	March 27, 2025 at 09:52:24 AM
Basic template for interval target	Data mining pipeline that contains a Data, Imputation, Linear Regression, and Model Comparison node connected in a linear flow.	SAS Pipeline	March 27, 2025 at 09:52:24 AM
Intermediate template for class target	Data mining pipeline that extends the basic template to include more advanced modeling techniques.	SAS Pipeline	March 27, 2025 at 09:52:25 AM

2 **OK** **Cancel**

- Now we need a data set. So, click **Browse** under the **Data** section:

**New Project**

Name: \*  
Customer Churn

Type: \*  
Data Mining and Machine Learning

Template:  
Basic template for class target

Data:  
 Browse

Description:

- There are – hopefully – a lot of data sets already loaded to memory in your environment. Type **commsdata** in the **Search for all data** bar. Then, select commsdata. And since it's likely your first time seeing this cell phone plan data, check out the **Details** and **Columns** tab to learn a little more about the data before simply selecting OK. Those clicks:

**Choose Data**

commsdata 1

< Back Results: 1-1 of 1

Name	★	Library	Date Modified	Modified By
COMMSDATA <span style="border: 2px solid red; padding: 2px;">2</span>	☆	Public	Mar 21, 2025 1:40 PM	SAS System Au...

1 of 1

**COMMSDATA**  
Public

- Primary Key (Customer\_ID)  
Index: 0, Length: 12 formatted, 8 raw
- Xsell Upsell Flag (upsell\_xsell)  
Index: 1, Format: BEST, Length: 2 formatted,...
- Churn Flag (churn)  
Index: 2, Format: BEST, Length: 2 formatted,...
- Lifetime Value (lifetime\_value)  
Index: 3, Format: DOLLAR, Length: 8 format

- We'll explore the data more in a bit. For now, simply know that **Churn** is the outcome variable that we really care about and then click **OK** at the bottom right:



- You've now got all the elements of a new project. Click **Save**:

**New Project**

Name: \*  
Customer Churn

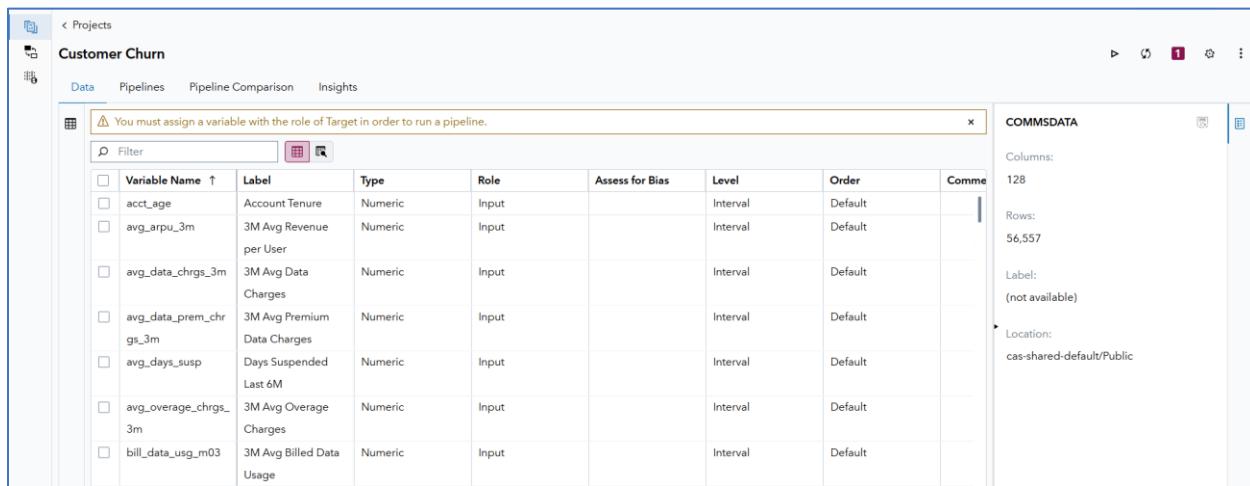
Type: \*  
Data Mining and Machine Learning

Template:  
Basic template for class target

Data:  
Public.COMMSDATA

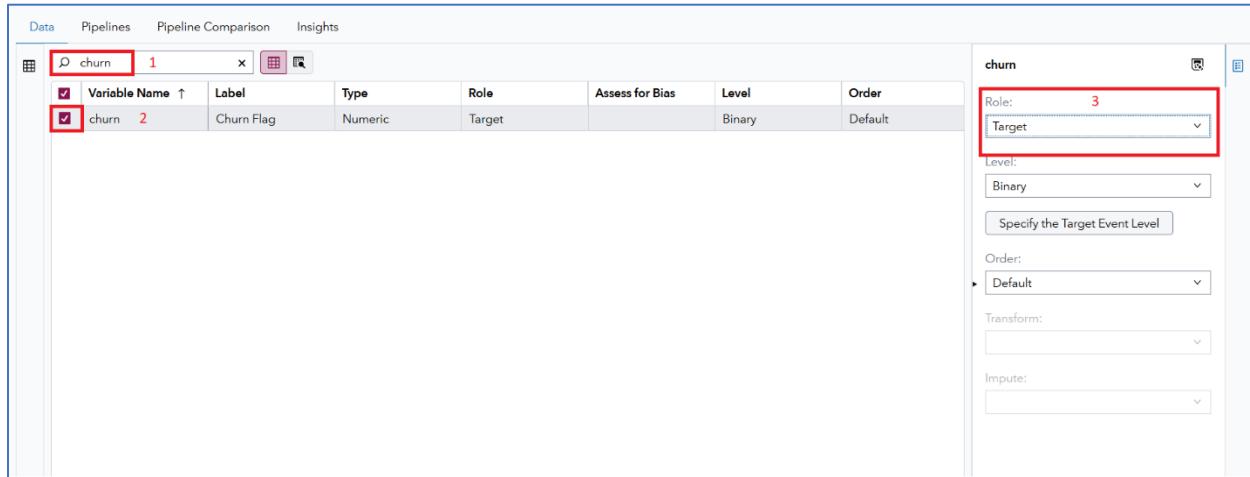
Description:

- The **Data** tab awaits you in SAS Model Studio! Like any good project, you should start by knowing thy data. So, spend a little time here examining the variables, their roles, types, and other attributes. You'll quickly note that there are 128 columns in the data and 56,557 rows. Additionally, you'll see this highlighted note:



Variable Name ↑	Label	Type	Role	Assess for Bias	Level	Order	Comment
acct_age	Account Tenure	Numeric	Input		Interval	Default	
avg_arpu_3m	3M Avg Revenue per User	Numeric	Input		Interval	Default	
avg_data_chrgs_3m	3M Avg Data Charges	Numeric	Input		Interval	Default	
avg_data_prem_chr_gs_3m	3M Avg Premium Data Charges	Numeric	Input		Interval	Default	
avg_days_susp	Days Suspended Last 6M	Numeric	Input		Interval	Default	
avg_overage_chrgs_3m	3M Avg Overage Charges	Numeric	Input		Interval	Default	
bill_data_usg_m03	3M Avg Billed Data Usage	Numeric	Input		Interval	Default	

- Gotta have an outcome variable – or target – if we’re going to do some predicting. So, type “churn” into the filter and press Enter. Then click the check mark next to churn to change the attributes. In the **Properties**, you can then change **Role** to **Target**. Those cumulative steps:

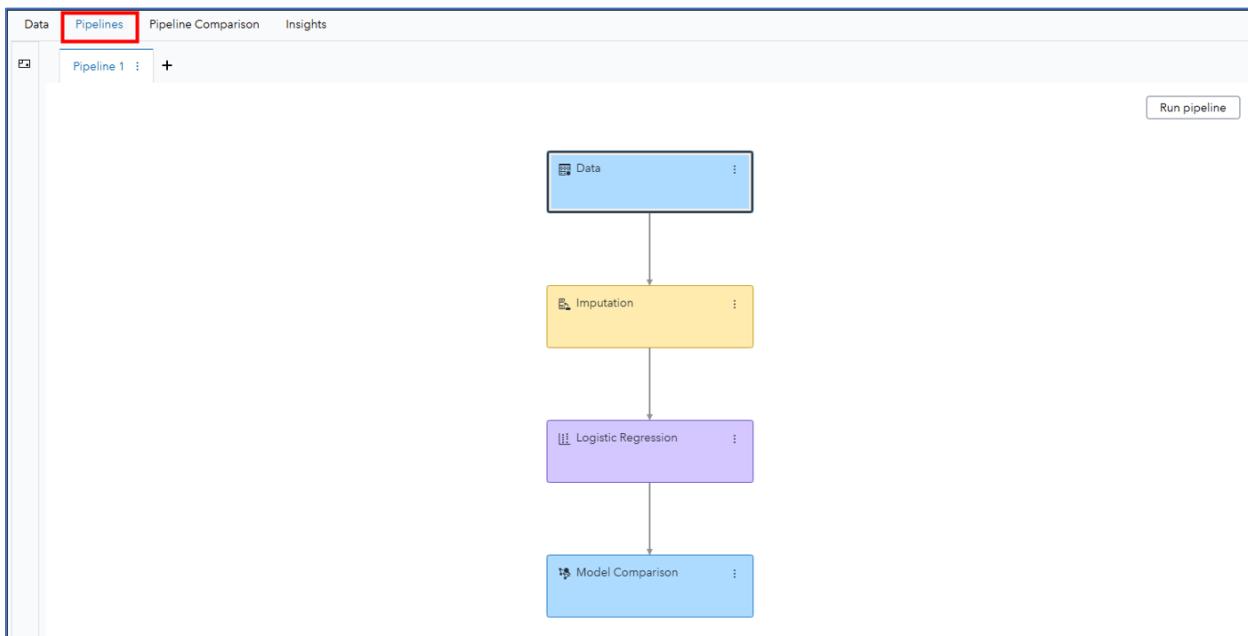


Variable Name ↑	Label	Type	Role	Assess for Bias	Level	Order
churn	Churn Flag	Numeric	Input		Binary	Default
churn	Churn Flag	Numeric	Target		Binary	Default

- Notice that the warning goes away once a target is assigned. Yay! You can then click the **Clear filter text** button and examine the data a bit more:



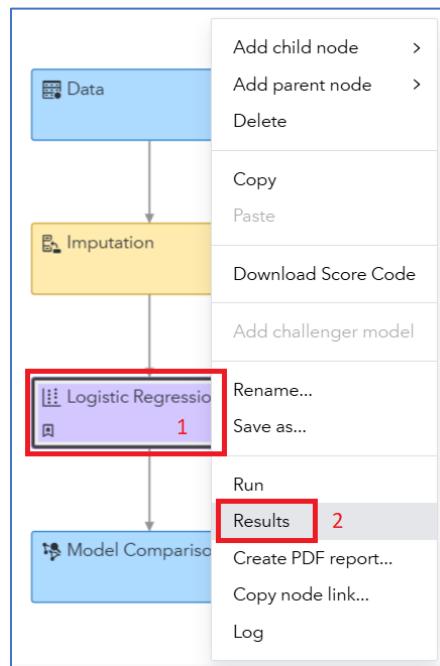
- We normally spend a LOT of time ensuring that the data and metadata are setup properly. But, since this is a shorter workshop on open-source integration, let’s simply ignore all that and accept the system defaults. The next step in our adventure is to click the **Pipelines** tab, where you will see our very basic pipeline:



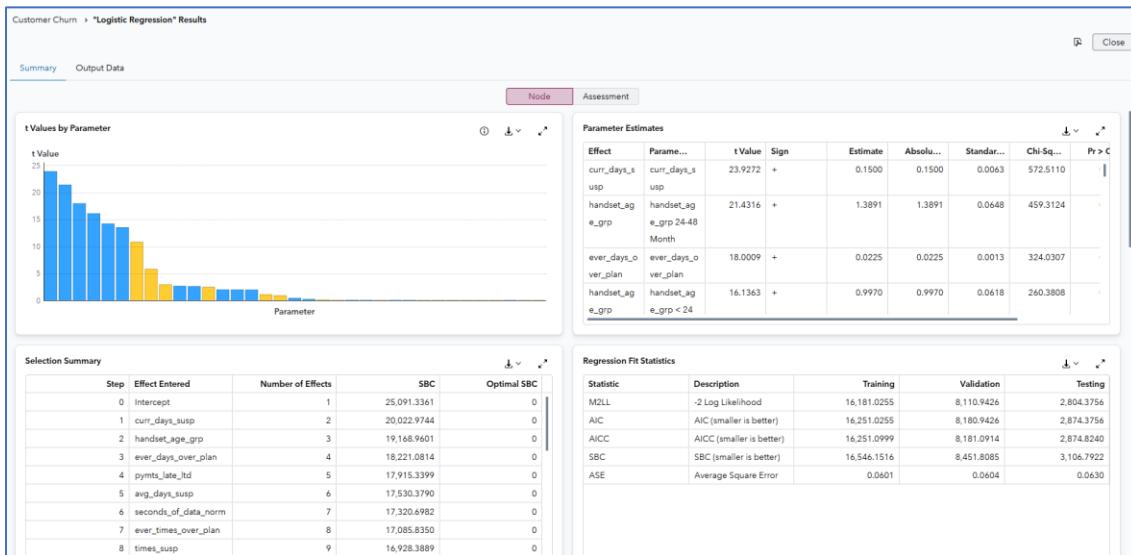
- Welcome to your first pipeline! To better understand how it works, click that **Run Pipeline** button, so we can learn by exploring output:

Run pipeline

- As you can see, SAS Model Studio is a node-centric environment that provides items in a logical, data-lifecycle process flow. When relevant, results for each node can be investigated by clicking the node and selecting Results. For example, find **Logistic Regression** in the Supervised Learning section of the analysis and select **Results**:



- Take some time to explore the logistic regression results. Which variables matter most? And what do the assessment plots look like. All those answers found in here:



- Once finished, give yourself a high-five and click the **Close** button. You just set up your first SAS Model Studio Project – and we can move on to *Part 2: Open-Source Model Integration*.

## Part 2: Open-Source Model Integration

As a SAS Guided Demo meant to fit it roughly 90 minutes, we'll get to the good stuff as soon as possible. Many of our new hires come in with significant open-source experience and ask the question "can I incorporate R + Python into my predictive modeling pipelines?"

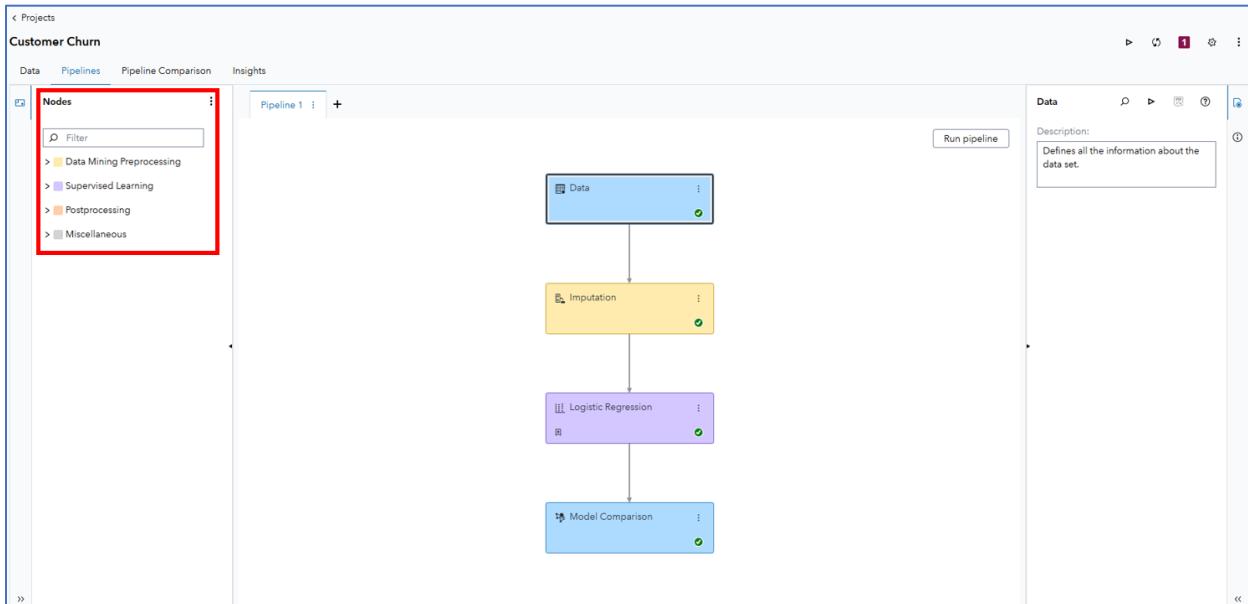
The answer is yes. A resounding yes.

So, let's get to it. In this section, we'll show you how to add increasingly complex machine learning models to our basic template pipeline – in both R and Python. Our first two models will be in R – as we'll start with a Decision Tree and then add a Forest model (because what's often better than a single tree? A: a Forest!)

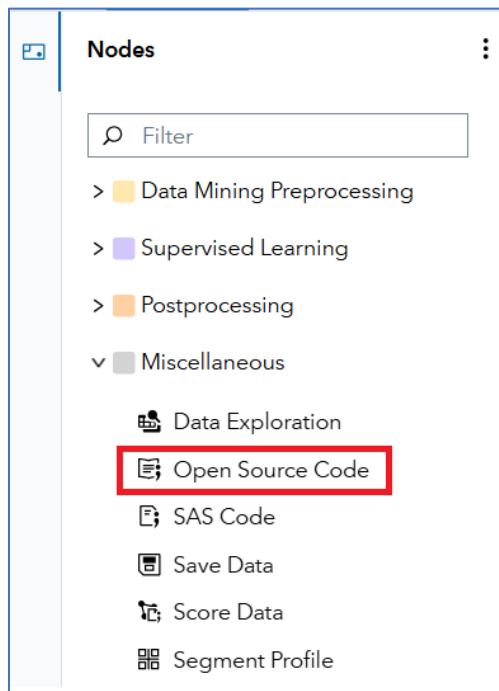
After the R programs, we'll then switch over to Python and add a gradient boosting model and, finally, a neural network. That's a whole lot of predictive modeling, so let's get on it.

### Open Source Code: General Setup

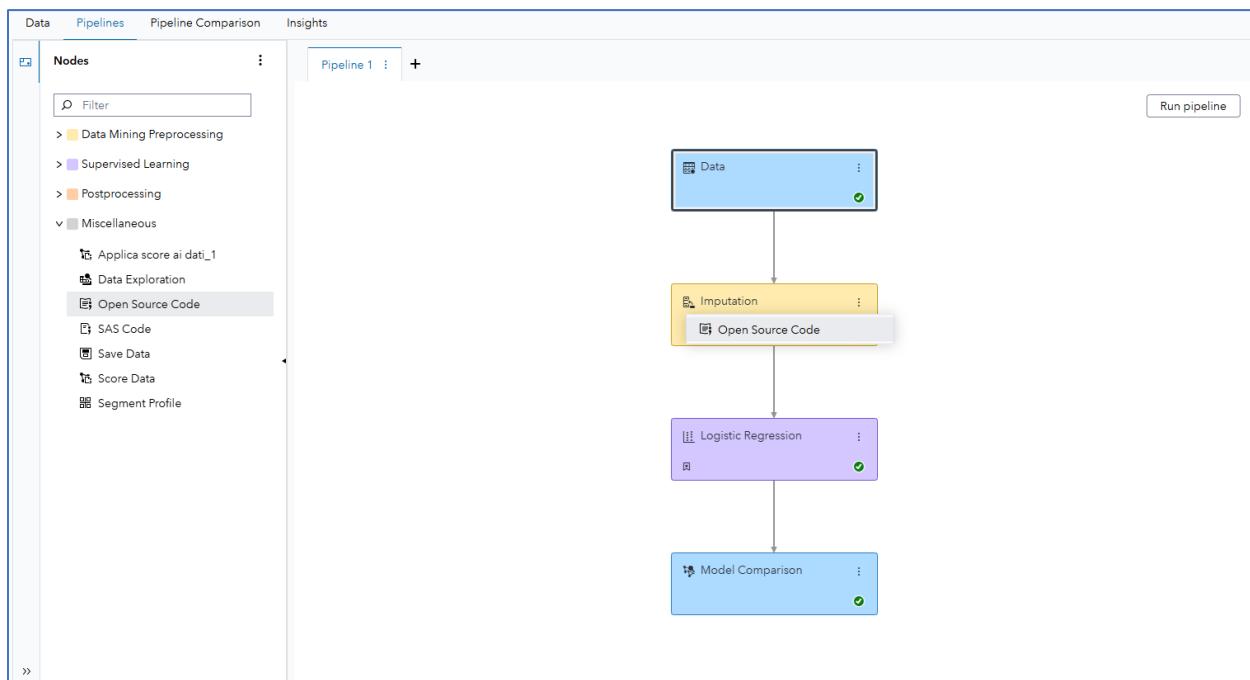
- In case you missed it on the tour, SAS Model Studio is node driven, with all the nodes contained here:



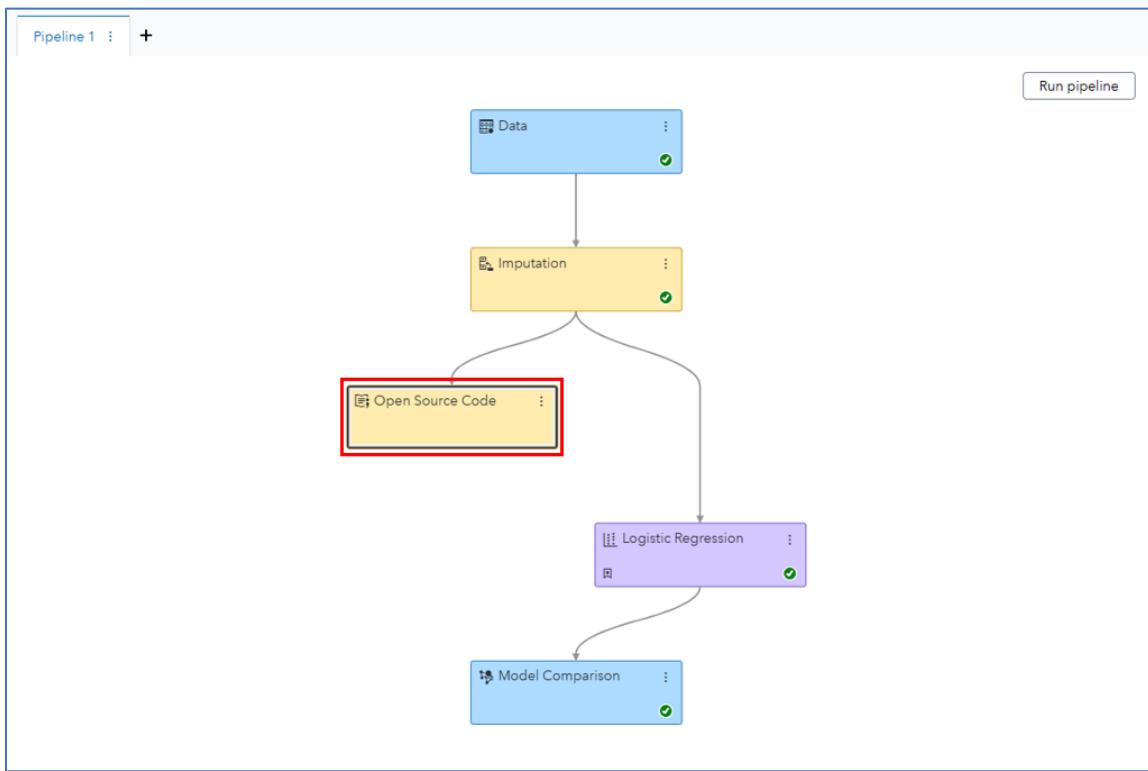
- Open the individual subsection and explore at your leisure. Then finish by opening the **Miscellaneous nodes**, and locating the **Open Source Code** node found here:



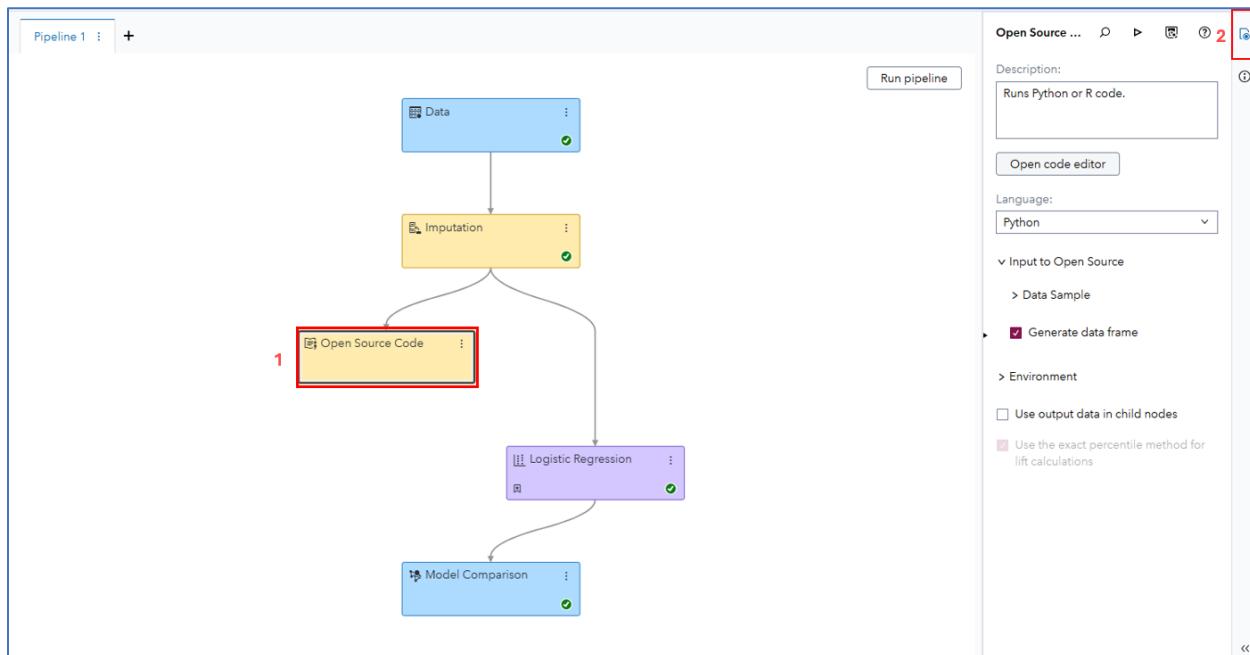
- Drag-and-drop the **Open Source Code** node on top of the **Imputation** node, located in the main pipeline. Here's how the dragging-and-droppin' looks in action:



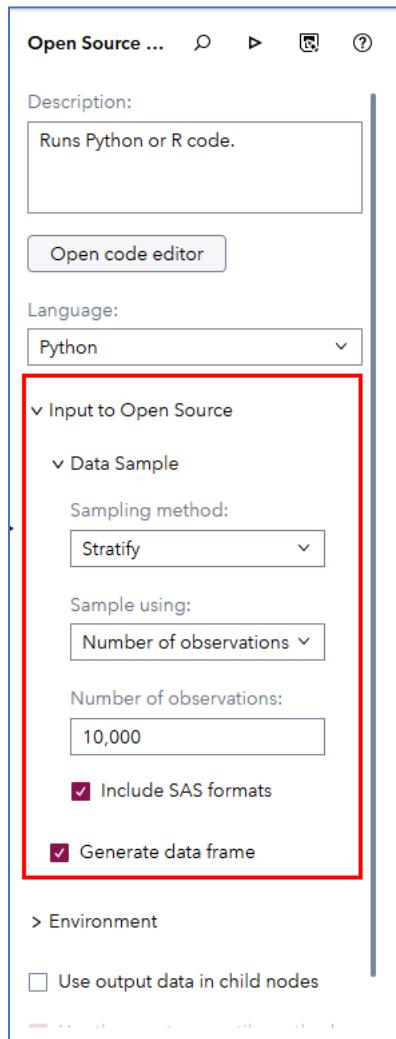
- You've now got an Open Source Code node in your pipeline:



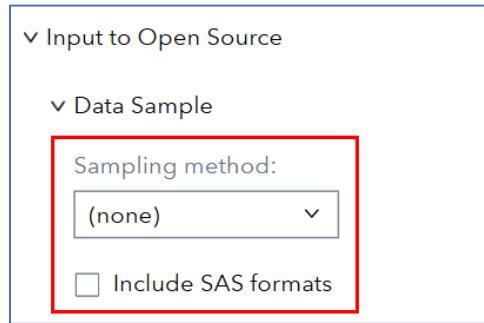
- Before hopping right into the coding, let's make a few global changes that we'll want to make to all our Open Source code nodes. Start by clicking on the **Open Source Code** node and opening the **Node options** pane:



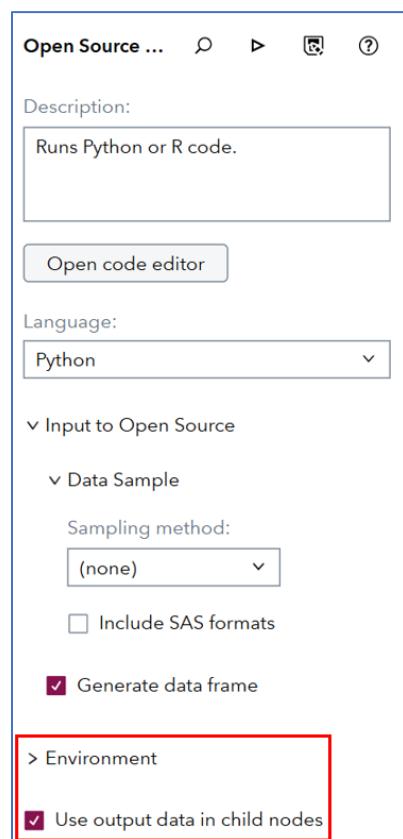
- You've got many, many options to explore! And while I won't cover the "why" behind a lot of the settings we'll adjust, know that we need to change some setting to ensure that the data are being processed the same as the other Supervised Learning models in the pipeline (here just the Logistic Regression). To start, expand the **Input to Open Source** section here:



- We've got some work to do. We don't want to sample, as this was done in SAS Model Studio in a previous step. So, change **Sampling method** to **(none)**. Moreover, we don't want to use **SAS formats**, because they can affect how the open source code is processed. So, ensure that option isn't selected. Those two changes:



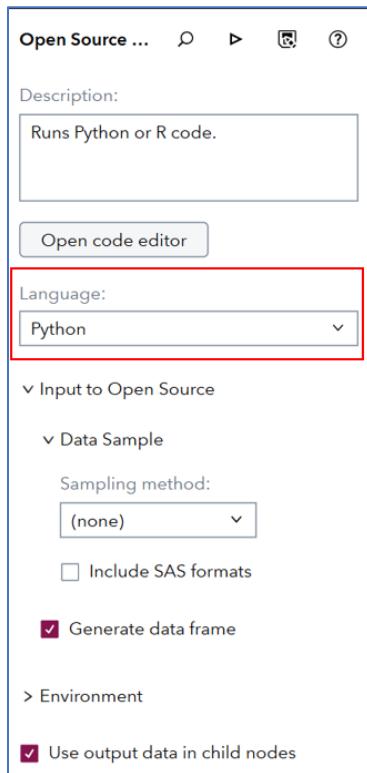
- Finally, we'd like to **Use the output data in child nodes**. It's kind of why we're running the analysis. So, ensure that this box is checked:



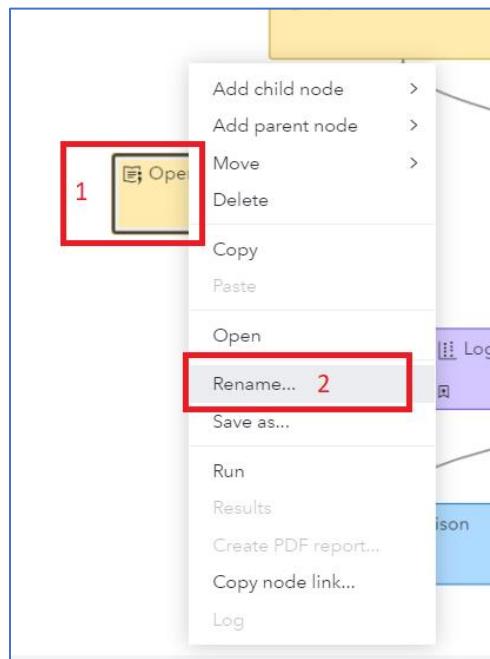
- You've got the global setup set up (see what I did there?). Yay! You'll need to do this setup 5 times – one for each of the open source models we'll incorporate.

## Model 1: R – Decision Tree

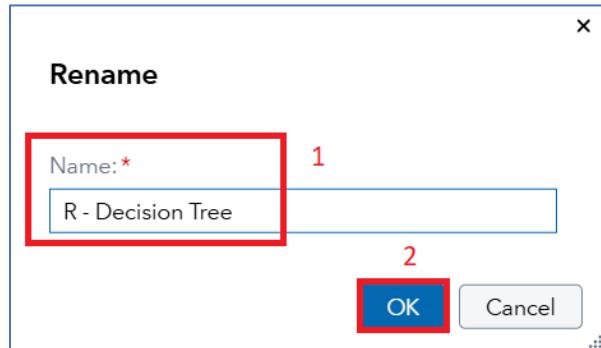
- Now let's get the actual modeling started. You may have noticed the **Language** option in the **Open Source Code** node:



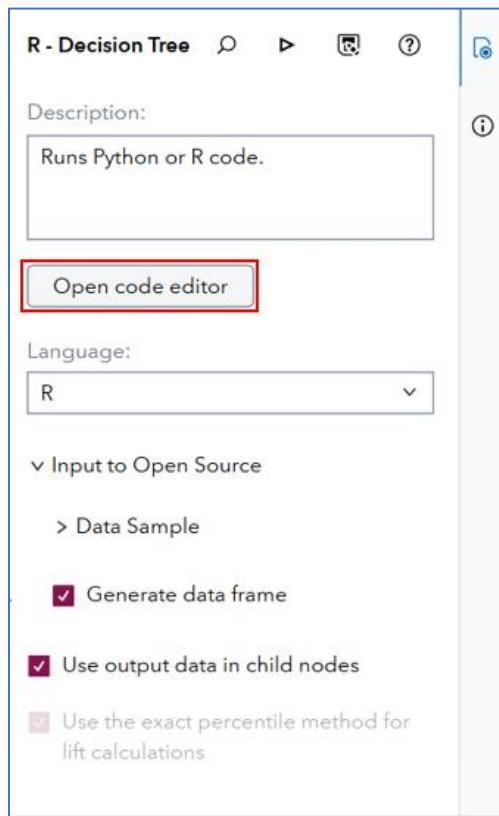
- The default language is Python. But, since we'd like to start with some R code, we can change that setting to R. Moreover, the name of the node is pretty generic at “Open Source Code”. Right click on the node and select **Rename**:



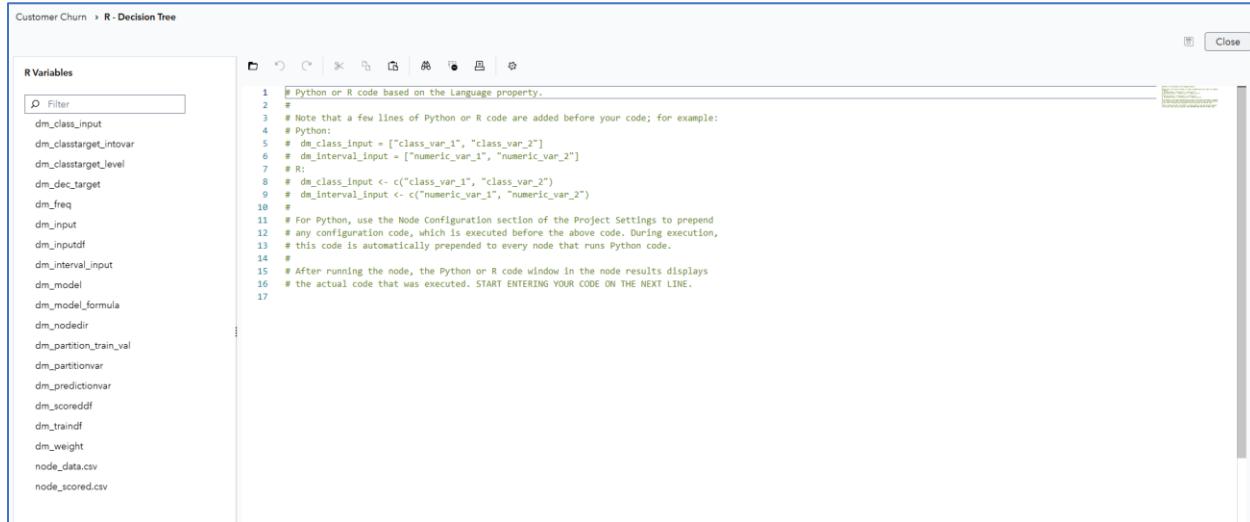
- Change the name to something a bit more helpful like **R – Decision Tree** and then click **OK**.



- Alright, progress! Now we can get to the code. From Node options, find the **Open code editor** button and push it:



- Check it out! Welcome to the Open Source Code coding interface!



The screenshot shows the code editor window for the R - Decision Tree node. On the left is a list of variables: dm\_class\_input, dm\_classtarget\_intovar, dm\_classtarget\_level, dm\_dee\_target, dm\_freq, dm\_input, dm\_inputdf, dm\_interval\_input, dm\_model, dm\_model\_formula, dm\_nodedir, dm\_partition\_train\_val, dm\_partitionvar, dm\_predictionvar, dm\_scoredf, dm\_trandf, dm\_weight, node\_data.csv, and node\_scored.csv. The main area contains Python code starting at line 18:

```

1 # Python or R code based on the Language property.
2 #
3 # Note that a few lines of Python or R code are added before your code; for example:
4 # Python:
5 # dm_class_input = ["class_var_1", "class_var_2"]
6 # dm_interval_input = ["numeric_var_1", "numeric_var_2"]
7 # R:
8 # dm_class_input <- c("class_var_1", "class_var_2")
9 # dm_interval_input <- c("numeric_var_1", "numeric_var_2")
10 #
11 # For Python, use the Node Configuration section of the Project Settings to prepend
12 # any configuration code, which is executed before the above code. During execution,
13 # this code is automatically prepended to every node that runs Python code.
14 #
15 # After running the node, the Python or R code window in the node results displays
16 # the actual code that was executed. START ENTERING YOUR CODE ON THE NEXT LINE.
17

```

- And while the true R hackers out there can start hacking away, I'm keeping an eye on time. So, instead of building code from scratch, I simply want you to grab the code below and past in into the coding editor, starting at line 18:

```

library(rpart)

# Decision Tree Model
dm_model <- rpart(dm_model_formula, data=dm_traindf, method="class")

# Prediction
pred <- predict(dm_model, dm_inputdf, type="prob")
dm_scoreddf <- data.frame(pred)
colnames(dm_scoreddf) <- c("P_CHURNO", "P_CHURN1")

# Print/plot model output
plot(dm_model, main='Decision Tree Plot')
text(dm_model, use.n=TRUE, cex=0.6) # Adds node labels with sample sizes to the plot

# Variable importance is not as straightforward in rpart as in randomForest, but we can
retrieve overall variable usage
importance <- data.frame(
  Variable = names(dm_model$variable.importance),
  Importance = as.vector(dm_model$variable.importance)
)
write.csv(importance, file = "rpt_treeIMP.csv", row.names = FALSE)
  
```

- The pasted code will appear as follows:

Customer Churn > R - Decision Tree

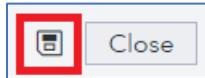
**R Variables**

Filter

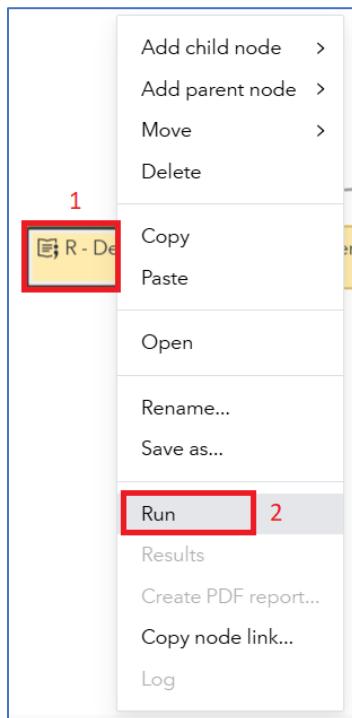
```

1  # Python or R code based on the Language property.
2
3  # Note that a few lines of Python or R code are added before your code; for example:
4  # Python:
5  #   dm_class_input = ["class_var_1", "class_var_2"]
6  #   dm_interval_input = ["numeric_var_1", "numeric_var_2"]
7  #   R:
8  #   dm_class_input <- c("class_var_1", "class_var_2")
9  #   dm_interval_input <- c("numeric_var_1", "numeric_var_2")
10 #
11 # For Python, use the Node Configuration section of the Project Settings to prepend
12 # any configuration code, which is executed before the above code. During execution,
13 # this code is automatically prepended to every node that runs Python code.
14 #
15 # After running the node, the Python or R code window in the node results displays
16 # the actual code that was executed. START ENTERING YOUR CODE ON THE NEXT LINE.
17
18 library(rpart)
19
20 # Decision Tree Model
21 dm_model <- rpart(dm_model_formula, data=dm_traindf, method="class")
22
23 # Prediction
24 pred <- predict(dm_model, dm_inputdf, type="prob")
25 dm_scoreddf <- data.frame(pred)
26 colnames(dm_scoreddf) <- c("P_CHURNO", "P_CHURN1")
27
28 # Print/plot model output
29 plot(dm_model, main='Decision Tree Plot')
30 text(dm_model, use.n=TRUE, cex=0.6) # Adds node labels with sample sizes to the plot
31
32 # Variable importance is not as straightforward in rpart as in randomForest, but we can retrieve overall variable usage
33 importance <- data.frame(
34   Variable = names(dm_model$variable.importance),
35   Importance = as.vector(dm_model$variable.importance)
36 )
37 write.csv(importance, file = "rpt_treeIMP.csv", row.names = FALSE)
  
```

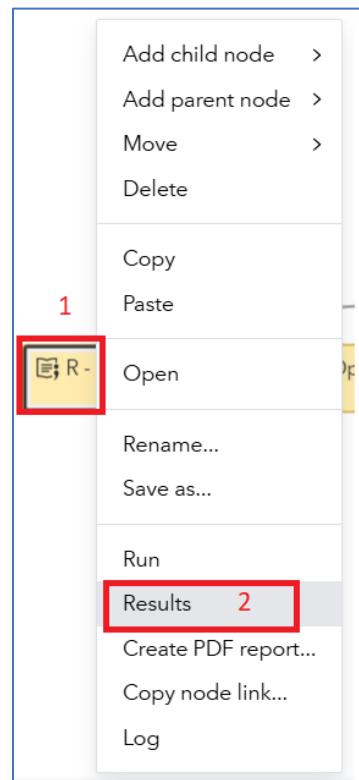
- Click that save button in the upper-right hand corner to lock-in those changes:



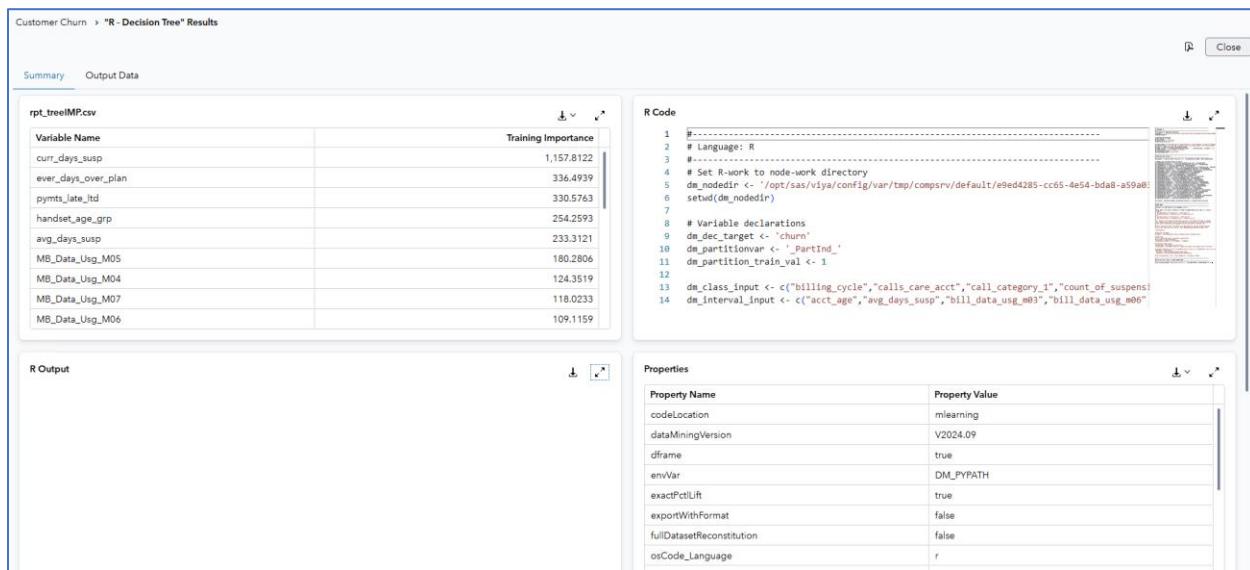
- Remember floppy disks? Wait, what? You're too young to remember? No worries, the cloud is way better, so we don't need to linger there. Click that **Close** button next.
- Now we can run that code. Right-click on the **R – Decision Tree** node and select **Run**:



- A completed pipeline will have a green check mark. Right-click on the decision-tree node and then select **Results**:



- Have some fun exploring those results! And then click **Close** when you're finished.



**R Code**

```

1 #-----
2 # Language: R
3 #-
4 # Set R-work to node-work directory
5 dm_nodedir <- '/opt/sas/viya/config/var/tmp/compsrv/default/e9ed4285-cc65-4e54-bda8-a59a0f'
6 setwd(dm_nodedir)
7
8 # Variable declarations
9 dm_dec_target <- 'churn'
10 dm_partitionvar <- 'PartInd_'
11 dm_partition_train_val <- 1
12
13 dm_class_input <- c("billing_cycle","calls_care_acct","call_category_1","count_of_suspensi
14 dm_interval_input <- c("acct_age","avg_days_susp","bill_data_usg_m03","bill_data_usg_m06")

```

- Your next logical question may be “how did the decision tree do relative to the logistic regression model?” And to you I say, “solid question”. Let’s check it out. Right-click on the **Model Comparison** node and select **Results**:

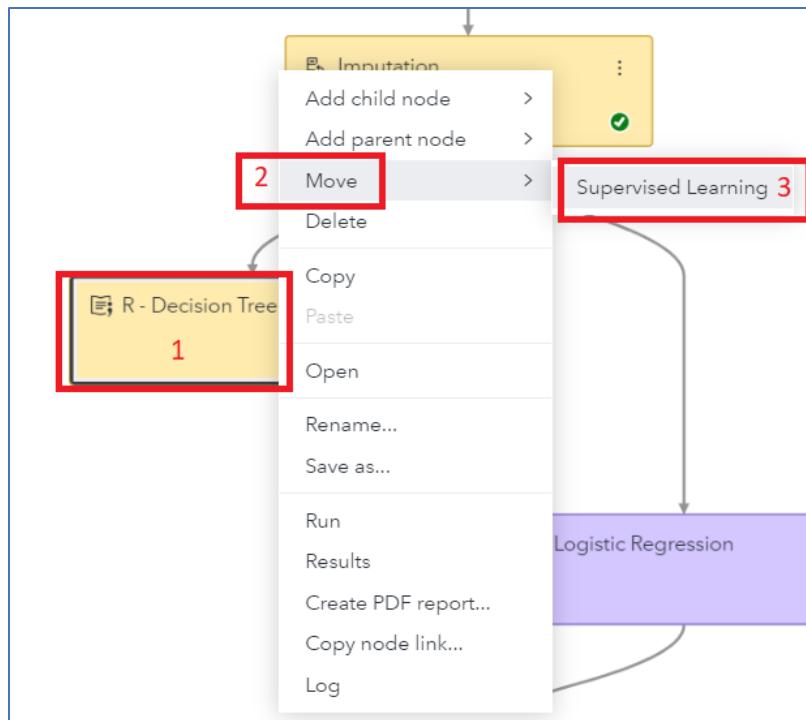
Customer Churn > "Model Comparison" Results

Ch...	Name	Algorit...	KS (You...	Accuracy	Average...	Area U...	Cumula...	Cumula...	Cutoff	Data Role	Depth	F1 Score	False D...	False P...	Gain	Gini Co...	ROC S...
*	Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029	5.6560	56.5598	0.5000	TEST	10	0.6183	0.1050	0.0076	4.6560	0.6058	0.464

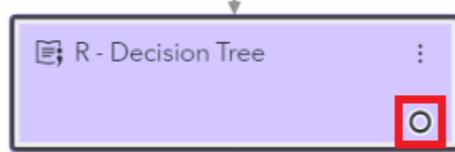
**Properties**

Property Name	Property Value
selectionCriteriaClass	Kolmogorov-Smirnov statistic (KS)
selectionCriteriaInterval	Average squared error
selectionTable	Test
selectionDepth	10
cutoff	0.50

- What in the what? Only the Logistic Regression results are there. We don't want that. So, click **Close** – as it looks like we've got an extra step.
- And I think I see the issue. See how the Logistic Regression node is in purple, while the **R - Decision Tree node** is still yellow? Purple denotes Supervised Learning models, while yellow is for Data Mining Preprocessing. So, we haven't actually registered that open source code node as being a supervised learning model. Don't fret – we're a click away from that. Right-click on the R-Decision tree node and then select **Move > Supervised Learning**. It's that simple, just three easy clicks:



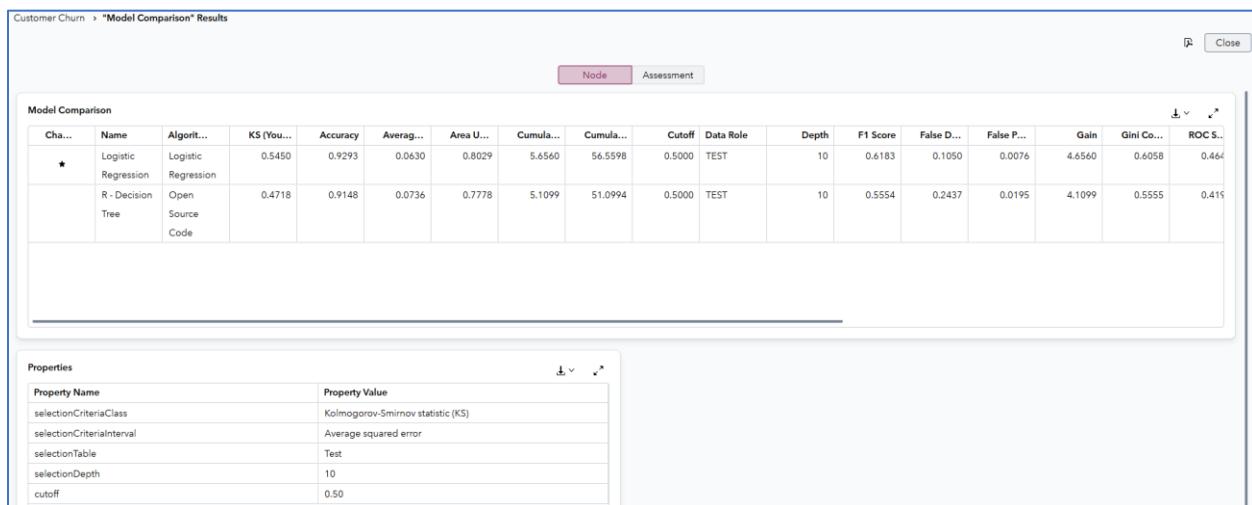
- You'll see that the green check mark has disappeared in the node, which means we need to run it again:



- This time, click the **Run pipeline**, so that we can update everything, including the Model Comparison:

Run pipeline

- When the modelling is completed, select **Results** from the **Model Comparison** node to see how the decision tree compares to the default logistic regression model. Do you see what I see?



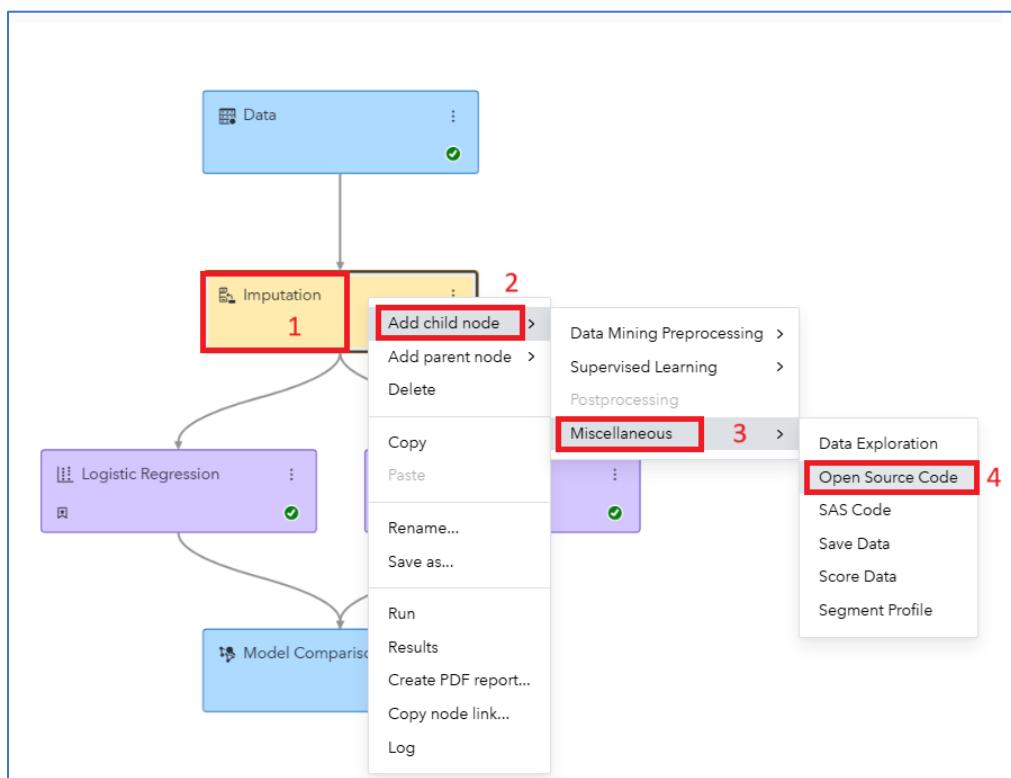
Model Comparison																	
Char...	Name	Algorit...	KS (You... r)	Accuracy	Averag...	Area U... nder	Cumula...	Cumula...	Cutoff	Data Role	Depth	F1 Score	False D...	False P...	Gain	Gini Co...	ROC S...
★	Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029	5.6560	56.5598	0.5000	TEST	10	0.6183	0.1050	0.0076	4.6560	0.6058	0.464
	R - Decision Tree	Open Source Code	0.4718	0.9148	0.0736	0.7778	5.1099	51.0994	0.5000	TEST	10	0.5554	0.2437	0.0195	4.1099	0.5555	0.419

- Looks like that Logistic Regression model is still our champ. The good news? We can still (potentially) beat it with other open source machine learning models.
- Click **Close** to exit the Model Comparison Results.

## Model 2: R – Forest

Are you feeling the modeling groove now? Let's crank out another machine learning model in our pipeline. And, in the interest of time, I won't cover ALL individual steps outline above. Instead, I'll just give you the general prompts and you can refer to the previous text to guide you along the pathway.

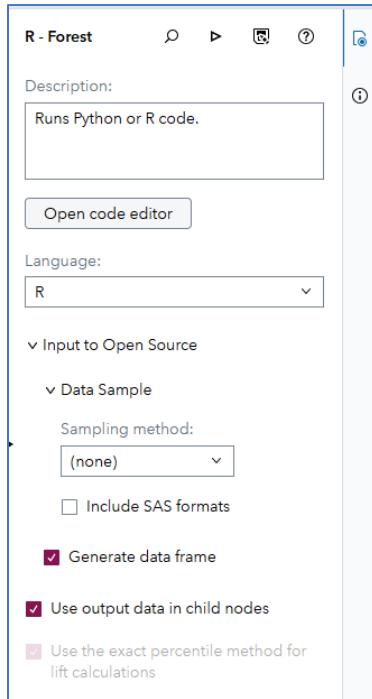
- In your SAS Model Studio Pipeline and let's add a second Open Source code. This time, for fun, right-click on the **Imputation** node, select **Add child node > Miscellaneous > Open Source Code**.



- Right-click on the node and rename the node to *R – Forest*:



- Next, under **Node options**, change the language from Python to **R**, change the **Sampling Method** to **none**, clear the checkmark for **Use SAS formats**, and **Use the output data in child nodes**. Those cumulative setting changes:



- Now, select **Open code editor**. It's time to code! But rather than making you work through the random forest code today, paste the following in the editor:

```
library(randomForest)

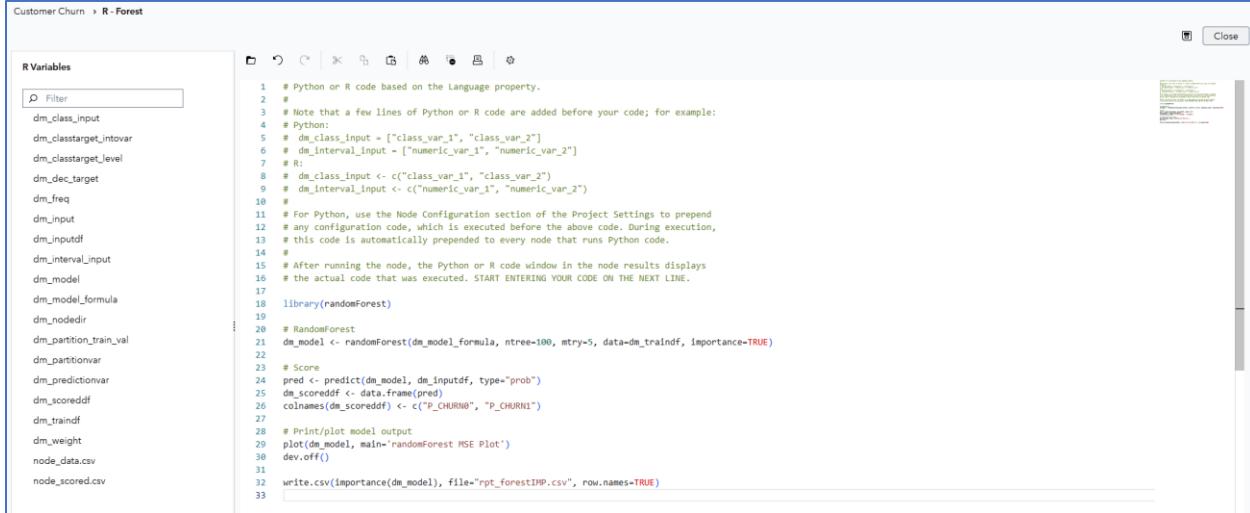
# RandomForest
dm_model <- randomForest(dm_model_formula, ntree=100, mtry=5, data=dm_traindf,
importance=TRUE)

# Score
pred <- predict(dm_model, dm_inputdf, type="prob")
dm_scoreddf <- data.frame(pred)
colnames(dm_scoreddf) <- c("P_CHURNO", "P_CHURN1")

# Print/plot model output
plot(dm_model, main='randomForest MSE Plot')
dev.off()

write.csv(importance(dm_model), file="rpt_forestIMP.csv", row.names=TRUE)
```

- Save that code – and the formatted code will appear in the editor as:



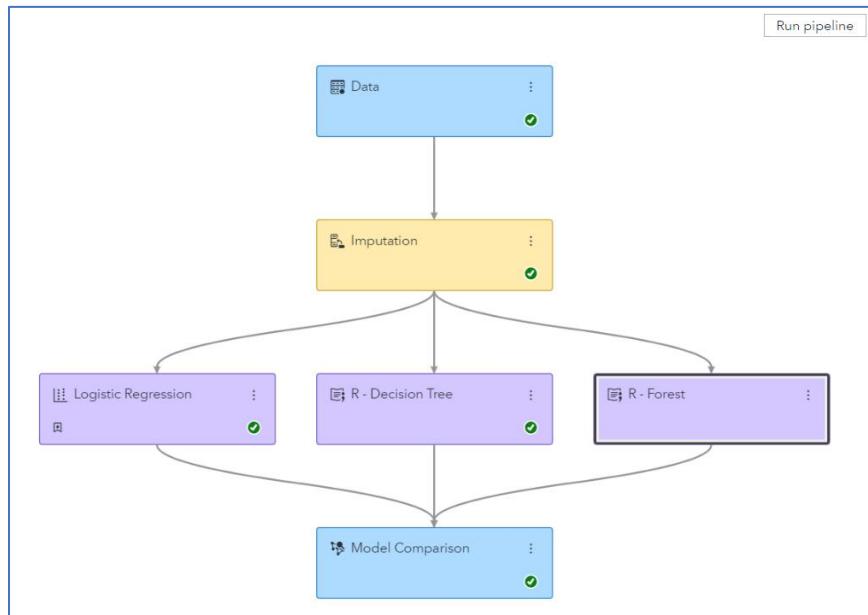
The screenshot shows the SAS Studio interface with the title "Customer Churn > R - Forest". On the left, there's a sidebar titled "R Variables" listing various objects like dm\_class\_input, dm\_classtarget\_intovar, etc. The main area contains an R script editor with the following code:

```

1  # Python or R code based on the Language property.
2  #
3  # Note that a few lines of Python or R code are added before your code; for example:
4  # Python:
5  # dm_class_input = ["class_var_1", "class_var_2"]
6  # dm_interval_input = ["numeric_var_1", "numeric_var_2"]
7  # R:
8  # dm_class_input <- c("class_var_1", "class_var_2")
9  # dm_interval_input <- c("numeric_var_1", "numeric_var_2")
10 #
11 # For Python, use the Node Configuration section of the Project Settings to prepend
12 # any configuration code, which is executed before the above code. During execution,
13 # this code is automatically prepended to every node that runs Python code.
14 #
15 # After running the node, the Python or R code window in the node results displays
16 # the actual code that was executed. START ENTERING YOUR CODE ON THE NEXT LINE.
17
18 library(randomForest)
19
20 # RandomForest
21 dm_model <- randomForest(dm_model_formula, ntree=100, mtry=5, data=dm_traindf, importance=TRUE)
22
23 # Score
24 pred <- predict(dm_model, dm_inputdf, type="prob")
25 dm_scoreddf <- data.frame(pred)
26 colnames(dm_scoreddf) <- c("P_CHURN0", "P_CHURN1")
27
28 # Print/plot model output
29 plot(dm_model, main="randomForest MSE Plot")
30 dev.off()
31
32 write.csv(importance(dm_model), file="rpt_forestIMP.csv", row.names=TRUE)
33

```

- Click **Close**. The last step before hitting that run pipeline button is to move the node down into the **Supervised Learning** pathway. Right-click, select **Move >> Supervised Learning**, and your pipeline should appear as:



- Let 'er rip, by clicking the **Run pipeline** button:



Run pipeline

- After it runs, let's check out the results in the **Model Comparison** window. How did we do?

Model Comparison						
Cha...	Name	Algorithm Name	↓ KS (Youden)	Accuracy	Averag...	Area U...
★	R - Forest	Open Source Code	0.5794	0.9362	0.0613	0.8099
	Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029
	R - Decision Tree	Open Source Code	0.4718	0.9148	0.0736	0.7778

- Oh yeah – that model beat the previous two. Woot! Forest for the win!

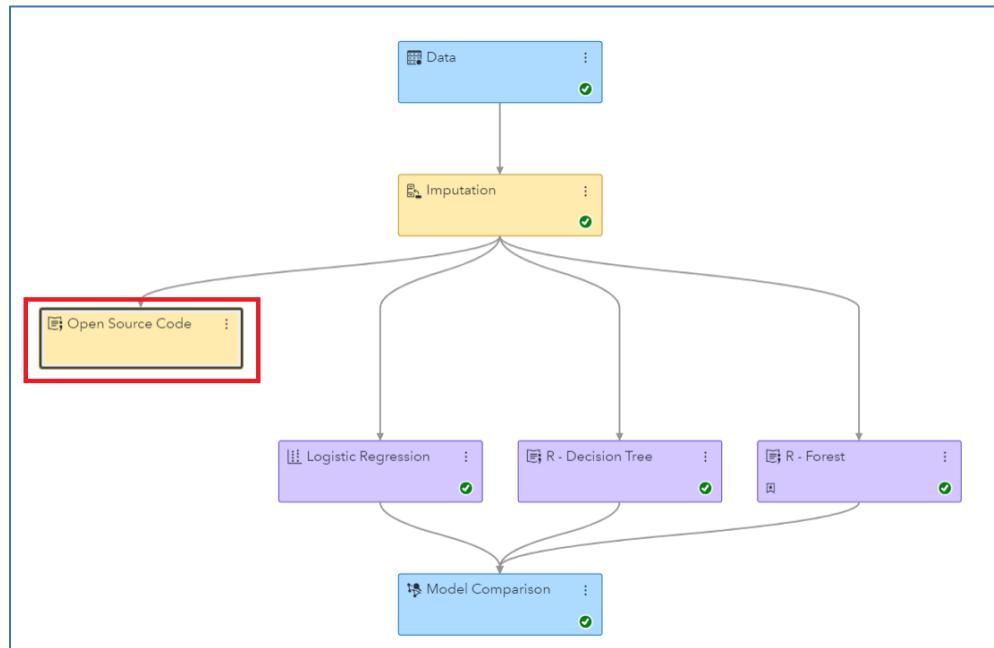
### Model 3: Python – Forest

Let's switch gears in our predictive modeling quest and focus on Python. And since we saw that Forest code was successful in the last set of modeling, let's compare like-for-like by running an equivalent forest model in Python.

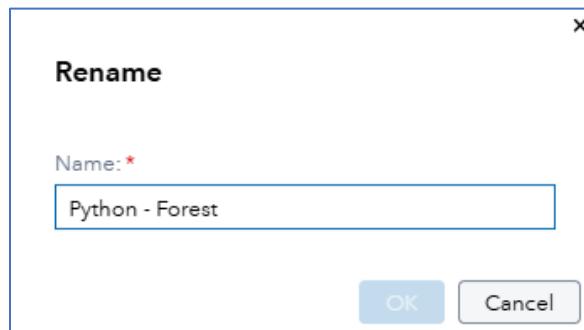
Deal?

Let's get started!

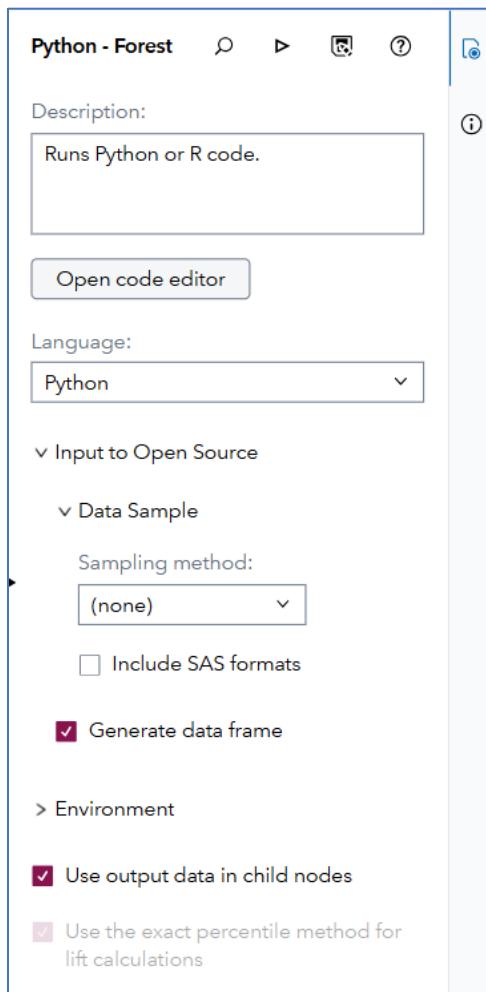
- Returning back to you SAS Model Studio pipeline, add another **Open Source** code. Again, there are several ways to do this – simply ensure that the node is added right after the Imputation node, here:



- Right-click on the node and rename the node to *Python – Forest*:



- Next, under **Node options**, ensure that the default language is still Python and make the usual changes: (1) choose **none** under **Sampling Method**, clear the checkmark for **Use SAS formats**, and check **Use the output data in child nodes**. Those cumulative setting changes:



- Now, select **Open code editor**, as its copy-and-paste time. Grab the following and put it into the Python editor for the **Training Code**:



```
from sklearn import ensemble

# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)

# Create X (features/inputs); drop partition indicator
X_enc = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_enc = X_enc.drop(dm_partitionvar, axis=1)

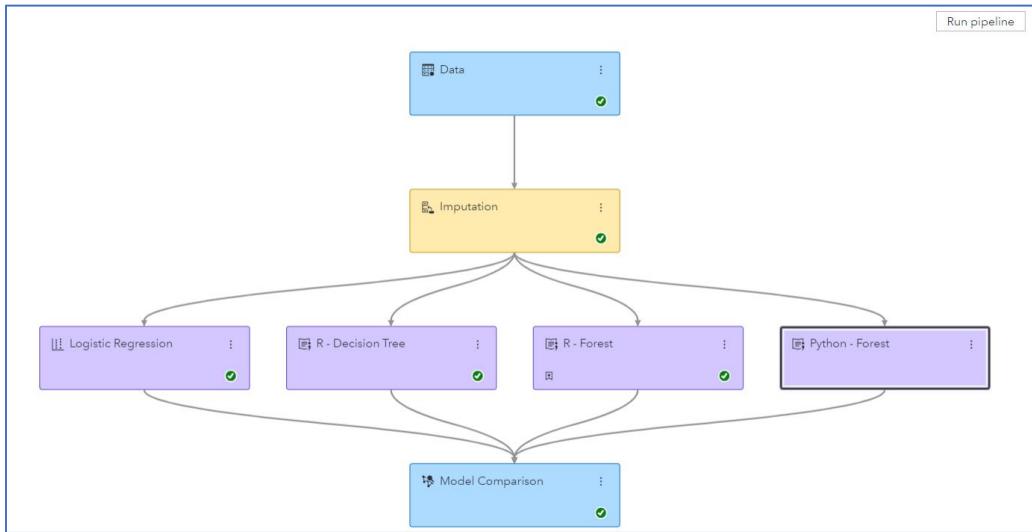
# Create y (labels)
y = dm_traindf[dm_dec_target]

# Fit RandomForest model w/ training data
params = {'n_estimators': 100, 'max_depth': 20, 'min_samples_leaf': 5}
dm_model = ensemble.RandomForestClassifier(**params)
dm_model.fit(X_enc, y)
print(dm_model)

# Save VariableImportance to CSV
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)

# Score full data
fullX_enc = fullX_enc.drop(dm_partitionvar, axis=1)
dm_scoreddf = pd.DataFrame(dm_model.predict_proba(fullX_enc),
columns=['P_CHURNO', 'P_CHURN1'])
```

- Save that code and click **Close** to exit out of the code editor. Like the other times, let's move our new node into the Supervised Learning pathway. Again, that process is to right-click, select **Move >> Supervised Learning**. Your new pipeline appears as:



- We want to see a green checkmark next to the Python-Forest node, so find our friend:

**Run pipeline**

- Click it. After the new model runs, return to the **Model Comparison** node. Right-click and view the **Results**. How did we do?

Model Comparison						
Cha...	Name	Algorithm Name	↓ KS (Y...)	Accuracy	Averag...	Area U...
★	Python - Forest	Open Source Code	0.5859	0.9348	0.0601	0.8105
	R - Forest	Open Source Code	0.5794	0.9362	0.0613	0.8099
	Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029
	R - Decision Tree	Open Source Code	0.4718	0.9148	0.0736	0.7778

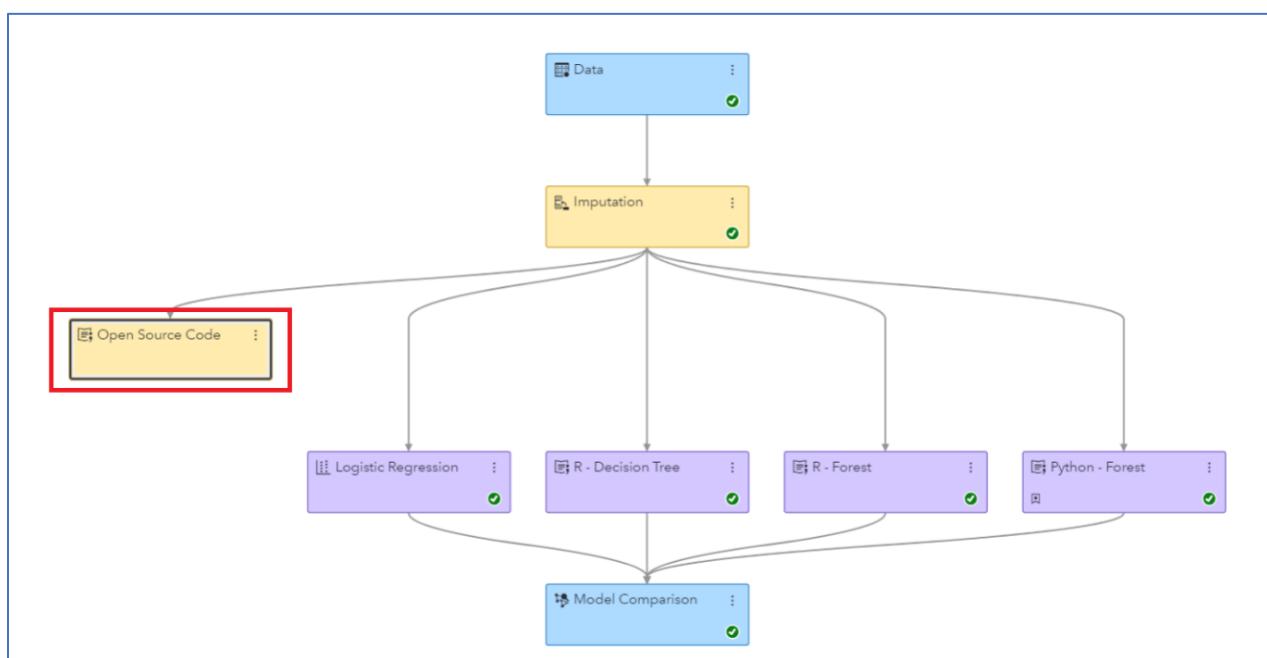
- Model models. Yay! Looks like *Python – Forest* performs marginally better than the R-forest. More progress!
- Because we can, let's add two more machine learning models to our pipeline. Take a break if you need it, and I'll see you in the next section!

## Model 4: Python – Gradient Boosting

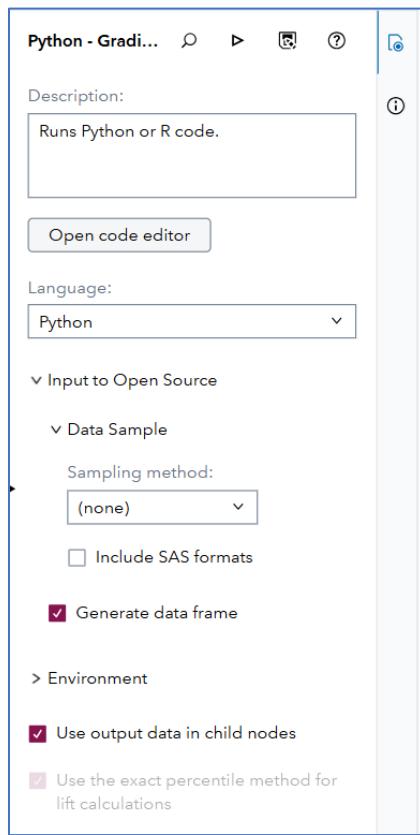
Another great tree-based algorithm to use in predictive modeling is a Gradient Boosting model. It works differently than a random forest. But focus on iterative algorithms. But while random forest examines a series of more traditional decision trees, gradient boosting models focus on the cases when the algorithm is getting the prediction modeling wrong, and then boosts (or highly weights) those observations so it does a better job of getting them right in the next iteration. Kind of cool, right?

Let's run now run that gradient boosting model in Python:

- We start, again, in our SAS Model Studio pipeline. And I hope that these steps are all becoming a bit familiar. Start by adding a new **Open Source Code** node to your pipeline, right after the **Imputation** node:



- Rename the node **Python – Gradient Boosting** and then, under **Node options** make the usual 4 changes: (1) **Language = Python**, (2) **Sampling Method = none**, (3) no to **Use SAS formats**, and (4) yes to **Use the output data in child nodes**. Those cumulative changes:



- Now the code. And we're moving right along. Grab the following and paste into the code into the **Training Code** editor:

```
from sklearn import ensemble
import pandas as pd

# Assuming dm_inputdf, dm_traindf, dm_nodedir, and other variables are defined
# elsewhere in your code

# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)

# Create X (features/inputs); drop partition indicator
X_enc = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_enc = X_enc.drop(dm_partitionvar, axis=1)
```

```

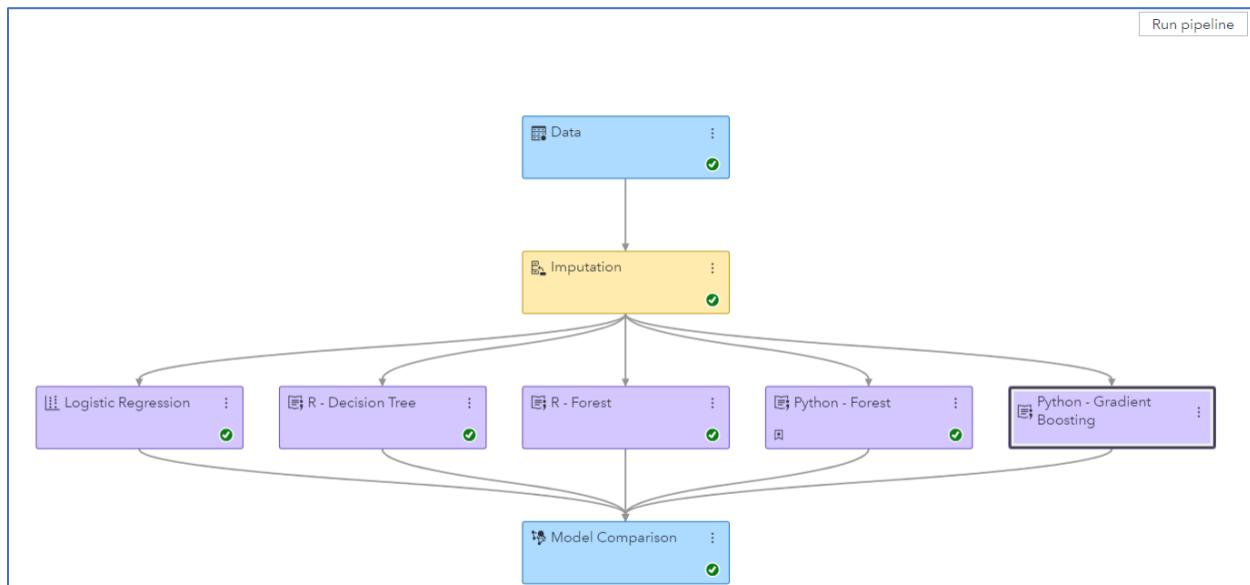
# Create y (labels)
y = dm_traindf[dm_dec_target]

# Fit Gradient Boosting model w/ training data
params = {'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.1}
dm_model = ensemble.GradientBoostingClassifier(**params)
dm_model.fit(X_enc, y)
print(dm_model)

# Save Variable Importance to CSV
feature_importances = pd.DataFrame(list(zip(X_enc.columns,
dm_model.feature_importances_)),
columns=['Variable Name', 'Importance'])
feature_importances.to_csv(dm_nodedir + '/rpt_var_imp.csv', index=False)

# Score full data
fullX_enc = fullX_enc.drop(dm_partitionvar, axis=1)
dm_scoreddf = pd.DataFrame(dm_model.predict_proba(fullX_enc),
columns=['P_CHURNO', 'P_CHURN1'])
  
```

- Like the other times, save that new code and **Close** the editor. And let's go ahead and move our new node into the **Supervised Learning** pathway now. Your new pipeline:



- Time to hit that **Run pipeline** button again:

Run pipeline

- And how do the results look? Well, right-click on the **Model Comparison** node and select **Results**.

Model Comparison						
Cha...	Name	Algorithm Name	↓ KS (Y...)	Accuracy	Averag...	Area U...
★	Python - Forest	Open Source Code	0.5859	0.9348	0.0601	0.8105
	R - Forest	Open Source Code	0.5794	0.9362	0.0613	0.8099
	Python - Gradient Boosting	Open Source Code	0.5766	0.9351	0.0572	0.8117
	Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029
	R - Decision Tree	Open Source Code	0.4718	0.9148	0.0736	0.7778

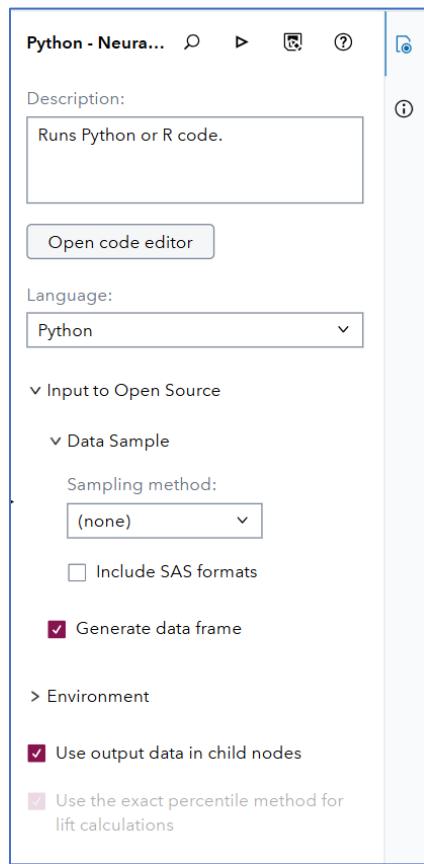
- The Python – Forest is still the champion. The *Python – Gradient Boosting* Model performs well but doesn't have a higher KS (Youden) statistic. Oh well. Click **Close** in the **Model Comparison** results window when you're done exploring.
- Let's add just one more model to our pipeline...

## Model 5: Python – Neural Network

Confession: I'm showing off a bit. But, that's OK... I hope that you're still learning something along the way 😊

Bravado aside, Neural Networks are becoming an increasingly popular machine learning tool. So, I'd feel a bit remiss if we didn't add one of those to our pipeline. Let's do it!

- Return one last time to the main SAS Model Studio pipeline canvas and add a new **Open Source Code** node to your pipeline.
- And let's do all that setup stuff one more time. Rename the node *Python – Neural Network*. Under **Node options** make the usual 4 changes: (1) **Language = Python**, (2) **Sampling Method = none**, (3) no to **Use SAS formats**, and (4) yes to **Use the output data in child nodes**. Those changes:



- Next step = code! Grab the following and paste away into the **Training Code** editor:

```

import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers

# Assuming dm_inputdf, dm_traindf, dm_nodedir, dm_partitionvar,
dm_partition_train_val, and dm_dec_target are defined

# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)

# Create X (features/inputs); drop partition indicator
X_train = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_train = X_train.drop(dm_partitionvar, axis=1)

```

```
# Create y (labels)
y_train = dm_traindf[dm_dec_target]

# Neural network model
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dropout(0.2), # Adding dropout for regularization
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2), # Adding dropout for regularization
    layers.Dense(1, activation='sigmoid') # Change to 'softmax' and modify units if it's a
multi-class problem
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy', # Change to 'categorical_crossentropy' for multi-class
problems
              metrics=['accuracy'])

# Fit the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

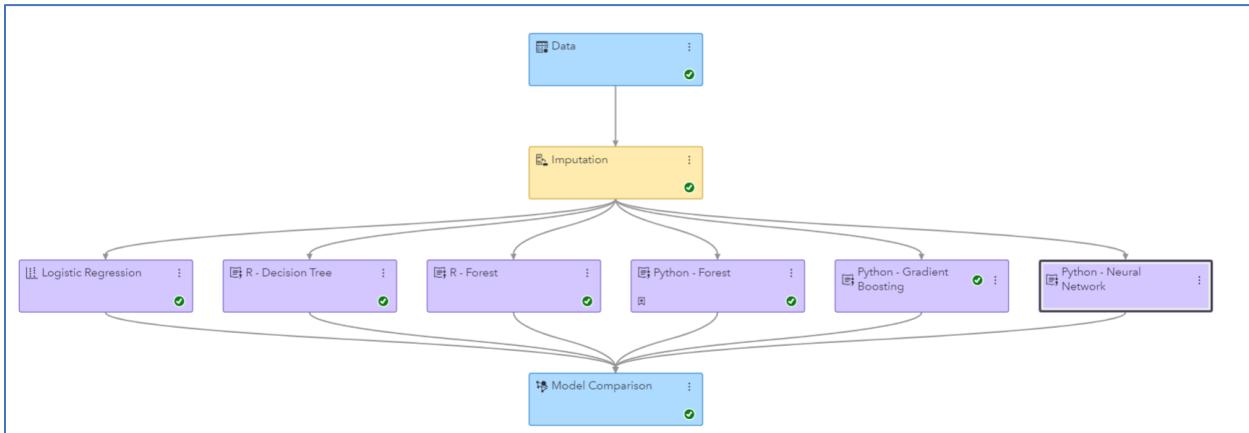
# Save the model
model.save(dm_nodedir + '/neural_network_model.h5')

# Prepare full data for scoring
X_full = fullX_enc.drop(dm_partitionvar, axis=1)

# Score full data using the trained model
probabilities = model.predict(X_full).flatten() # Get the probability of the positive class
dm_scoreddf = pd.DataFrame({
    'P_CHURN0': 1 - probabilities, # Probability of the negative class
    'P_CHURN1': probabilities # Probability of the positive class
})

print(dm_scoreddf.head())
```

- You might be thinking: *I know the next step!* And if you were to say, “move the node to the **Supervised Learning** pathway”, you would be correct. Do that now and your pipeline should appear as:



- That's a proper predictive modeling pipeline! [insert applause] Then press that **Run pipeline** button:

[Run pipeline](#)

- Let it run. How did we do? The answer is in the **Model Comparison** node, under **Results**:

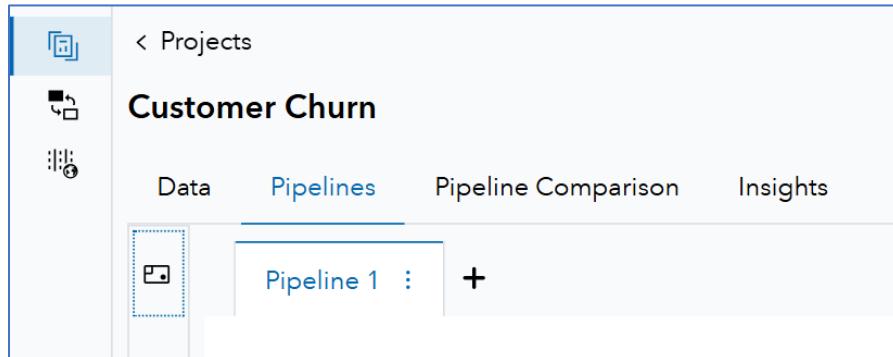
Model Comparison						
Cha...	Name	Algorithm Name	↓ KS (Y...)	Accuracy	Averag...	Area U...
★	Python - Forest	Open Source Code	0.5859	0.9348	0.0601	0.8105
	R - Forest	Open Source Code	0.5794	0.9362	0.0613	0.8099
	Python - Gradient Boosting	Open Source Code	0.5766	0.9351	0.0572	0.8117
	Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029
	R - Decision Tree	Open Source Code	0.4718	0.9148	0.0736	0.7778
	Python - Neural Network	Open Source Code	0	0.8787	0.1066	0.5000

- Yikes. That was not a good model, as measured by the model Accuracy (the KS Youden statistics wasn't even produced, by default). But, we ran it nonetheless and can say that we added a Neural Network Analysis to our modeling. Click **Close** when you're done exploring the **Model Comparison** results.

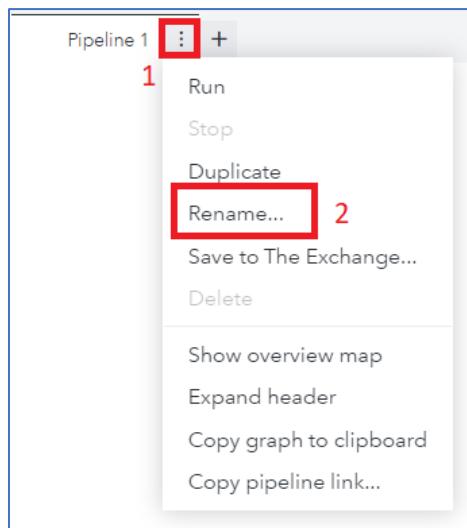
## Part 3: Run a SAS Model Studio Pipeline (for fun)

Today is not the day to walk through the full power of SAS Model Studio. Sorry. But we can still show you one of the cool features of SAS Model Studio: and that's how easy it is to add a SAS Model Studio pipeline template.

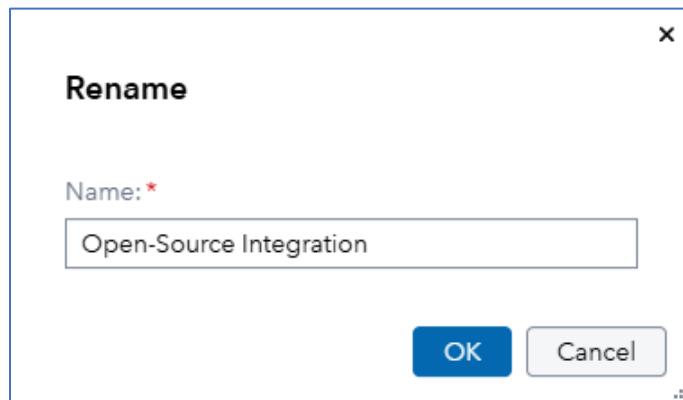
- Check out the upper-right hand corner of SAS Model Studio. More specifically here:



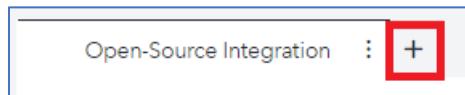
- See that little snowman right next to *Pipeline 1*? Click it and then select **Rename**:



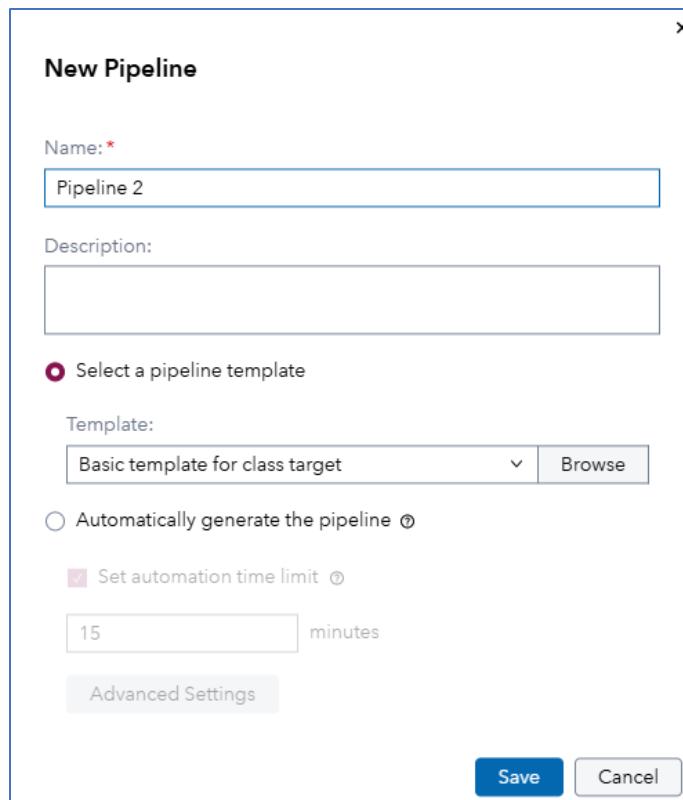
- Change the name of the pipeline to *Open-Source Integration*, as such:



- Click **OK**. And know that wasn't the cool part. Just something I needed to do. And we are in the right area, so find this button:



- That's the **Add new pipeline** button. Click it! And the **New Pipeline** button will appear:



- Let's make a couple of changes. Update **Name** to **Advanced Template**. Next, find the **Browse** button under **Select a pipeline template**:

**New Pipeline**

Name: \*

Description:

Select a pipeline template

Template:

Automatically generate the pipeline ?

Set automation time limit ?

minutes

- Again, check out all those pipeline options in the **Browse Templates** window! Click on **Advanced template for class target**, which should be the first option available:

**Browse Templates**

Template Name	Description	Owner	Last Modified
Advanced template for class target	Data mining pipeline that extends the intermediate template for a class target by adding neural network, forest, and gradient boosting models. An ensemble model is also provided.	SAS Pipeline	March 27, 2025 at 09:52:21 AM
Advanced template for class target with autotuning	Data mining pipeline for a class target that contains autotuned tree, forest, neural network, and gradient boosting models.	SAS Pipeline	March 27, 2025 at 09:52:21 AM
Advanced template for interval target	Data mining pipeline that extends the intermediate template for an interval target by adding neural network, forest, GAM, and gradient boosting models. An ensemble model is also provided.	SAS Pipeline	March 27, 2025 at 09:52:23 AM
Advanced template for interval target with autotuning	Data mining pipeline that extends the intermediate template for an interval target by adding GAM and autotuned tree, forest, neural network, and gradient boosting models. An ensemble model is also provided.	SAS Pipeline	March 27, 2025 at 09:52:22 AM
Basic template for class target	Data mining pipeline that contains a Data, Imputation, Logistic Regression, and Model Comparison node connected.	SAS Pipeline	March 27, 2025 at 09:52:24 AM

- Once *Advanced template for class target* is selected, click **OK**.
- Finally, click **Save** in the **New Pipeline** window, here:

**New Pipeline**

Name: \*  
Advanced Template

Description:

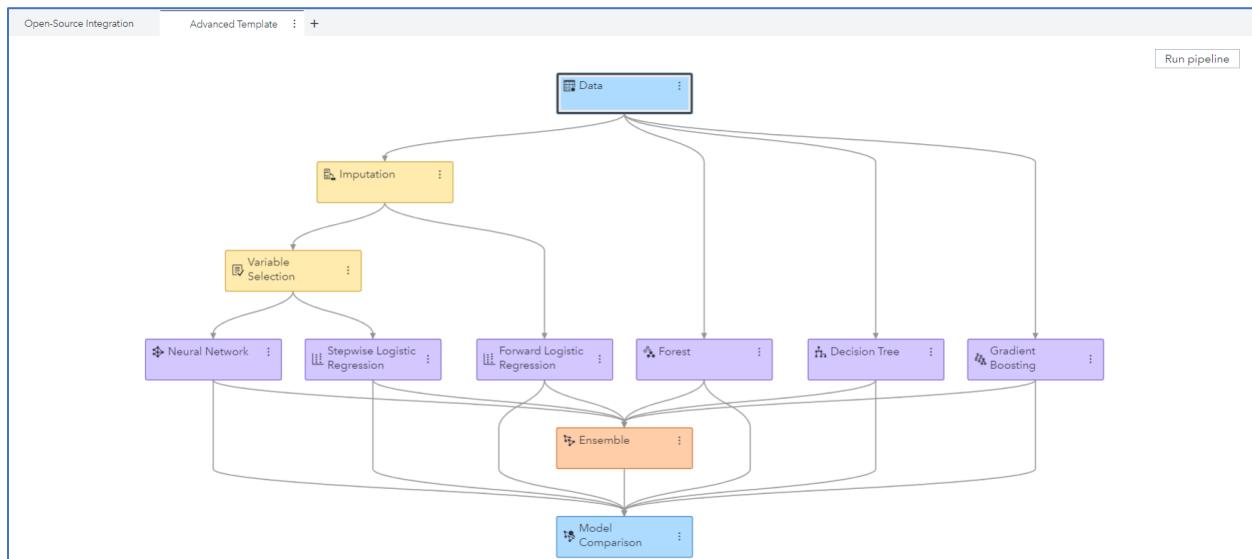
Select a pipeline template

Template:  
Advanced template for class target

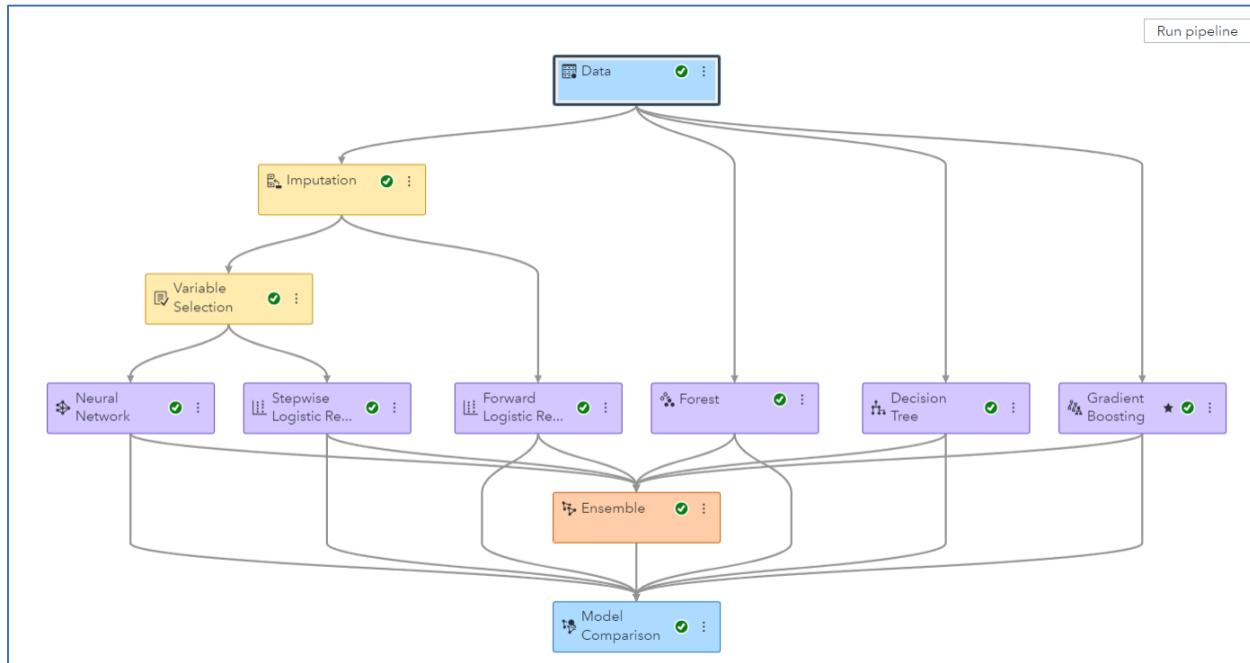
Automatically generate the pipeline ?

Set automation time limit ?  
15 minutes

- The pipeline will open in a new tab. And what a pipeline it is!



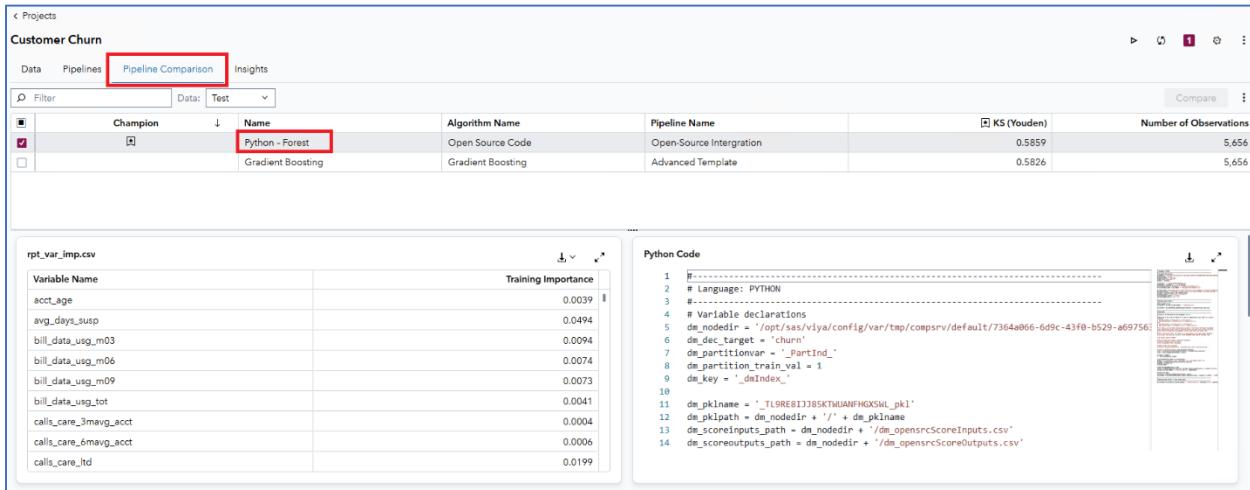
- Within this single pipeline, we've got some data pre-processing, variable selection, and – count 'em – 7 predictive models. Nice! Click **Run pipeline...** and grab a coffee.
- Green checkmarks will mean that everything ran successfully:



- We can dive more into the modeling another day. For now, simply open the **Results** window in the **Model Comparison** node and see how the models performed:

Model Comparison						
Ch...	Name	Algorithm Name	↓ KS (Youd...	Accuracy	Averag...	Area U...
★	Gradient Boosting	Gradient Boosting	0.5826	0.9390	0.0548	0.8103
	Forest	Forest	0.5653	0.9335	0.0601	0.8143
	Ensemble	Ensemble	0.5647	0.9318	0.0608	0.8088
	Neural Network	Neural Network	0.5505	0.9236	0.0665	0.8070
	Forward Logistic Regression	Logistic Regression	0.5450	0.9293	0.0630	0.8029
	Stepwise Logistic Regression	Logistic Regression	0.5434	0.9291	0.0631	0.8029
	Decision Tree	Decision Tree	0.3842	0.9135	0.0777	0.6963

- Looks like the Gradient Boosting model is our top performer in the *Advanced template for class target* pipeline. Explore a bit more, then click **Close** in the Model Comparison Results window when you're satiated.
- Question: did any of these models beat the champion model in our *Open-Source Integration* tab? Not sure how to check that? Well, I'm here to help! Click on the **Pipeline Comparison** tab, here:



The screenshot shows the SAS Model Studio interface for a project titled "Customer Churn". The "Pipeline Comparison" tab is selected, displaying a table comparing two pipelines: "Python - Forest" and "Gradient Boosting". The "Python - Forest" pipeline is highlighted as the champion model. Below the table are two code snippets: "rpt\_var\_imp.csv" showing variable importance and "Python Code" showing the corresponding Python script.

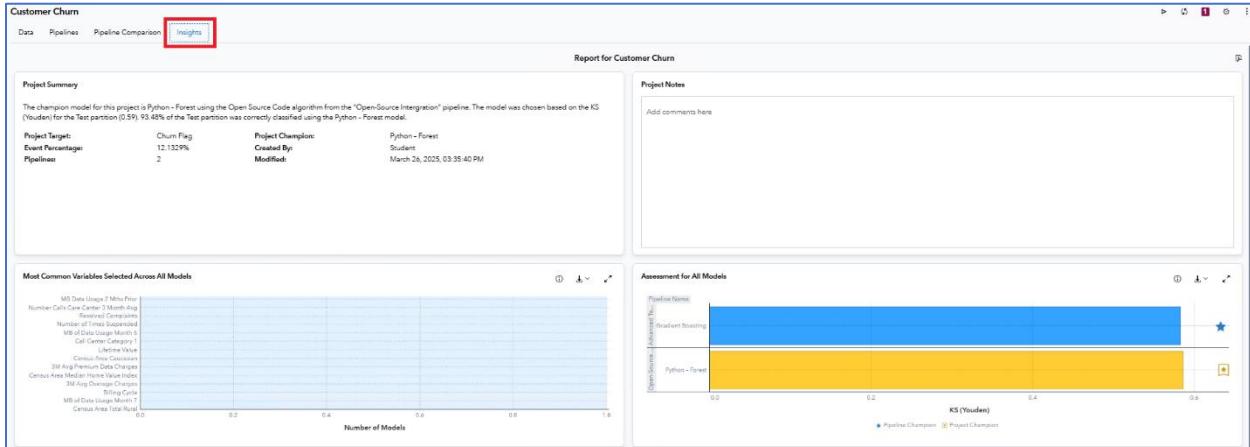
Champion	Name	Algorithm Name	Pipeline Name	KS (Youden)	Number of Observations
<input checked="" type="checkbox"/>	Python - Forest	Open Source Code	Open-Source Integration	0.5859	5,656
<input type="checkbox"/>	Gradient Boosting	Gradient Boosting	Advanced Template	0.5826	5,656

```

rpt_var_imp.csv
Variable Name          Training Importance
acct_age                0.0039
avg_days_susp            0.0494
bill_data_usg_m03         0.0094
bill_data_usg_m06         0.0074
bill_data_usg_m09         0.0073
bill_data_usg_tot          0.0041
calls_care_3mavg_acct      0.0004
calls_care_6mavg_acct      0.0006
calls_care_ltd              0.0199

Python Code
1 # -----
2 # Language: PYTHON
3 #
4 # Variable declarations
5 dm_nodedir = '/opt/sas/viya/config/var/tmp/compsrv/default/7364a066-6d9c-43f0-b529-a69756'
6 dm_dt_target = 'churn'
7 dm_partitionvar = '_PartInd'
8 dm_partition_train_val = 1
9 dm_key = '_dmIndex.'
10
11 dm_pklnase = '_Tl9REB1J385KTMARFH0SGML_pk1'
12 dm_pkxpath = dm_nodedir + '/' + dm_pklnase
13 dm_scoreinputs_path = dm_nodedir + '/dm_opensrcScoreInputs.csv'
14 dm_scoreoutputs_path = dm_nodedir + '/dm_opensrcScoreOutputs.csv'
  
```

- Looks like the *Python – Forest* is the champion model across the two pipelines, by the smallest of margins. Check out the KS (Yoden) stats: 0.5859 versus 0.5826. But, hey, that's probably close enough to call a tie 😊
- Just because I want to show you one more thing, click on the **Insights** tab, found here:



The screenshot shows the "Insights" tab in SAS Model Studio. It includes sections for "Project Summary", "Most Common Variables Selected Across All Models", and "Assessment for All Models".

**Project Summary:**

- Project Target: Churn Flag
- Event Percentage: 12.132%
- Pipelines: 2
- Project Champion: Python - Forest
- Created By: Student
- Modified: March 26, 2025, 03:35:40 PM

**Most Common Variables Selected Across All Models:**

Variable	Number of Models
MS Data Usage 2 Month Prior	1.0
Number Calls Care Center 1 Month Avg	1.0
Number of Times Suspended	1.0
MS Avg Premium Data Charges	1.0
Call Center Category 1	1.0
Lifetime Value	1.0
Census Area Total Runge	1.0
3M Avg Premium Data Charges	1.0
Census Area Total Runge	1.0
3M Avg Change Charges	1.0
Billing Cycle	1.0
MS Util Data Usage 2 Month Prior	1.0

**Assessment for All Models:**

Pipeline Name	KS (Yoden)
Gradient Boosting	~0.58
Python - Forest	~0.5859

- And, yes, this is simply nothing more than an excuse to show you another one of my favorite features in SAS Model Studio: the **Insights** tab summarizes your project and will help make your work more accessible to a general audience. Because, after all, you are the machine learning expert!



## Appendix: Pulling Back the Curtain: How I Created (part of) This

For many of us in the teaching and learning realm, we're learning how to incorporate new tools into our projects, such as ChatGPT. If you're interested in learning how I used ChatGPT3.5 to help me write a significant part of this asset, please see the conversation below. Please note that text from ChatGPT3.5 is in blue.

You can also see the full chat here: <https://chatgpt.com/share/c8b4d6ff-a9f4-4cde-bd60-2c6ce677c073>

### R + Python Code for SAS Model Studio

May 20, 2024

*This conversation may reflect the link creator's personalized data, which isn't shared and can meaningfully change how the model responds.*

I have the following python code which runs a random forest model. Can you update the code to run a gradient boosting model? Thanks! The code:

```
from sklearn import ensemble

# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)

# Create X (features/inputs); drop partition indicator
X_enc = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_enc = X_enc.drop(dm_partitionvar, axis=1)

# Create y (labels)
y = dm_traindf[dm_dec_target]

# Fit RandomForest model w/ training data
params = {'n_estimators': 100, 'max_depth': 20, 'min_samples_leaf': 5}
```



```
dm_model = ensemble.RandomForestClassifier(**params)
dm_model.fit(X_enc, y)
print(dm_model)

# Save VariableImportance to CSV
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)

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

#### ChatGPT

To update your existing code for running a Random Forest model to use a Gradient Boosting model instead, you'll switch from using `RandomForestClassifier` to `GradientBoostingClassifier`. Both are available from `sklearn.ensemble`. Here's the revised version of your code with the necessary changes:

```
from sklearn import ensemble
import pandas as pd
```

```
# Assuming dm_inputdf, dm_traindf, dm_nodedir, and other variables are defined elsewhere in
your code
```

```
# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]
```

```
# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)
```

```
# Create X (features/inputs); drop partition indicator
X_enc = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_enc = X_enc.drop(dm_partitionvar, axis=1)
```

```
# Create y (labels)
y = dm_traindf[dm_dec_target]
```

```
# Fit Gradient Boosting model w/ training data
params = {'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.1}
```



```
dm_model = ensemble.GradientBoostingClassifier(**params)
dm_model.fit(X_enc, y)
print(dm_model)

# Save Variable Importance to CSV
feature_importances = pd.DataFrame(list(zip(X_enc.columns, dm_model.feature_importances_)),
                                    columns=['Variable Name', 'Importance'])
feature_importances.to_csv(dm_nodedir + '/rpt_var_imp.csv', index=False)

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

Key Changes Explained:

1. **Model Type:** Changed from `RandomForestClassifier` to `GradientBoostingClassifier`.
2. **Model Parameters:** Adjusted default parameters to fit typical usage for gradient boosting:
  - `max_depth`: Often shallower than for random forests. Set to 3 here.
  - `learning_rate`: Added to control the contribution of each tree in boosting. Typical starting value is 0.1.
3. **Variable Importance Output:** Gradient boosting also supports feature importance, so this part remains similar, just ensure `X_enc.columns` is correctly referenced for column names.

Make sure all necessary libraries are imported and that your dataframe paths and other variable definitions are correctly set up in your script as the given snippet assumes these are predefined.

Excellent. Thank you!

ChatGPT

You're welcome! If you have any more questions or need further assistance with your model or anything else, feel free to ask. Happy coding!

Yay - that was super helpful. Thanks again. What about creating some code that estimates a standard neural network model?

ChatGPT



Sure! I can help you set up a standard neural network model using Keras, a popular library for building neural networks in Python. Below, I'll provide a simple example of a neural network designed for a classification task. This example assumes you have already prepared your data similarly to how you handled it for the random forest and gradient boosting models.

First, make sure you have Keras installed. If not, you can install it with TensorFlow, which includes Keras, by running:

```
pip install tensorflow
```

Here's a basic setup for a neural network model:

```
python
```

```
import numpy as np
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers

# Assuming dm_inputdf, dm_traindf, and other variables are defined elsewhere in your code

# Prepare the data
# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)

# Select the training data
X_train = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val].drop(dm_partitionvar,
axis=1)
y_train = dm_traindf[dm_dec_target]

# Convert labels to one-hot encoding if it's a multi-class classification
y_train = keras.utils.to_categorical(y_train, num_classes)

# Define the model architecture
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])
```

```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model if you have a test set available
# model.evaluate(X_test, y_test)

# Save the model
model.save('path_to_save_model')
```

#### Key Points of the Code:

1. **Data Preparation:** Similar to your previous setup, with dummies for categorical variables.
2. **Model Architecture:** The model consists of two hidden layers with 64 neurons each and ReLU activation. The output layer uses softmax activation for multi-class classification. Adjust `num_classes` based on your label classes.
3. **Compilation:** The model uses the Adam optimizer and categorical crossentropy loss, suitable for multi-class classification tasks.
4. **Training:** The model is trained for 10 epochs with a batch size of 32, and 20% of the training data is used for validation.

This is a basic neural network model. Depending on your specific dataset and task, you may need to adjust the model architecture, data preprocessing, and training parameters.

Very nice! But I meant updating the following code to instead run a neural network model:

```
# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)

# Create X (features/inputs); drop partition indicator
X_enc = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_enc = X_enc.drop(dm_partitionvar, axis=1)
```



```
# Create y (labels)
y = dm_traindf[dm_dec_target]

# Fit RandomForest model w/ training data
params = {'n_estimators': 100, 'max_depth': 20, 'min_samples_leaf': 5}
dm_model = ensemble.RandomForestClassifier(**params)
dm_model.fit(X_enc, y)
print(dm_model)

# Save VariableImportance to CSV
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)

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

### ChatGPT

Certainly! Let's update your code to use a basic neural network model instead of a RandomForest. We'll use TensorFlow and Keras for building and training the neural network. The neural network architecture will be simple, with fully connected (dense) layers. Here's how you can adapt your current setup:

python

[Copy code](#)

```
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers

# Assuming dm_inputdf, dm_traindf, dm_nodedir, dm_partitionvar, dm_partition_train_val, and
dm_dec_target are defined

# Get full data with inputs + partition indicator
dm_input.insert(0, dm_partitionvar)
fullX = dm_inputdf.loc[:, dm_input]

# Dummy encode class variables
fullX_enc = pd.get_dummies(fullX, columns=dm_class_input, drop_first=True)
```

```

# Create X (features/inputs); drop partition indicator
X_train = fullX_enc[fullX_enc[dm_partitionvar] == dm_partition_train_val]
X_train = X_train.drop(dm_partitionvar, axis=1)

# Create y (labels)
y_train = dm_traintdf[dm_dec_target]

# Neural network model
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dropout(0.2), # Adding dropout for regularization
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2), # Adding dropout for regularization
    layers.Dense(1, activation='sigmoid') # Change to 'softmax' and modify units if it's a multi-class
problem
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy', # Change to 'categorical_crossentropy' for multi-class problems
              metrics=['accuracy'])

# Fit the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Save the model
model.save(dm_nodedir + '/neural_network_model.h5')

# Prepare full data for scoring
X_full = fullX_enc.drop(dm_partitionvar, axis=1)

# Score full data using the trained model
dm_scoreddf = pd.DataFrame(model.predict(X_full), columns=['Probability'])

print(dm_scoreddf.head())
Key Adjustments Made:

```

1. **Model Architecture:** Added a neural network with two dense layers, each followed by dropout for regularization. Adjust the number of layers, units, and dropout rate based on your specific needs and dataset characteristics.



2. **Output Layer:** Configured for binary classification with a sigmoid activation function. If your task is a multi-class classification, use softmax and adjust the number of units in the final layer to match the number of classes.
3. **Loss Function:** Set to binary crossentropy, typical for binary classification problems. Use categorical crossentropy for multi-class classification.
4. **Model Saving:** The model is saved in the HDF5 format.
5. **Scoring:** Outputs the probability predictions for each instance in the full dataset.

Ensure to adapt this template according to the specific requirements of your dataset and problem (e.g., number of classes, activation functions, etc.).

I ran that code and got the following error: "ERROR: The model cannot be assessed because there are 2 posterior variables that could not be found in the scored data set: P\_BAD0 P\_BAD1". How do I update the code you provided to explicitly create P\_BAD0 and P\_BAD1?

#### ChatGPT

The error you received indicates that your scoring process or downstream application expects two specific columns, **P\_BAD0** and **P\_BAD1**, which usually represent the probabilities of each class in a binary classification problem. If the model is set to output a single probability (as is common with binary classifiers using sigmoid activation), this will typically be the probability of the positive class (often **P\_BAD1**).

Here's how you can modify the scoring part of the neural network code to explicitly generate these two columns:

```
python
Copy code
# Score full data using the trained model
probabilities = model.predict(X_full).flatten() # Get the probability of the positive class
dm_scoreddf = pd.DataFrame({
    'P_BAD0': 1 - probabilities, # Probability of the negative class
    'P_BAD1': probabilities # Probability of the positive class
})
print(dm_scoreddf.head())
```

#### Explanation:

1. **Predictions:** The `model.predict(X_full)` will output the probability that each sample belongs to the positive class (**P\_BAD1**).



2. **Calculating P\_BAD0:** You calculate `P_BAD0` as `1 - P_BAD1`, assuming that the sum of probabilities for both classes is 1, which is a typical scenario in binary classification.

This adjustment will ensure that your dataframe `dm_scoreddf` contains two columns specifically named `P_BAD0` and `P_BAD1`, corresponding to the predicted probabilities for each class, as required by your downstream application or scoring process.

You're doing great, ChatGPT! I now have some R code that I'd like to convert. It currently runs an R Forest. But I'd like the equivalent code to run a Gradient Boosting model. Can you create that new code? Here is the old code:

```
# RandomForest  
dm_model <- randomForest(dm_model_formula, ntree=100, mtry=5, data=dm_traindf,  
importance=TRUE)  
  
# Score  
pred <- predict(dm_model, dm_inputdf, type="prob")  
dm_scoreddf <- data.frame(pred)  
colnames(dm_scoreddf) <- c("P_BAD0", "P_BAD1")  
  
# Print/plot model output  
plot(dm_model, main='randomForest MSE Plot')  
dev.off()  
  
write.csv(importance(dm_model), file="rpt_forestIMP.csv", row.names=TRUE)
```

#### ChatGPT

To convert your existing R code from using `randomForest` to running a gradient boosting model, you can use the `gbm` package in R, which provides functions for fitting generalized boosted regression models. Below, I will provide the equivalent R code to run a Gradient Boosting model and perform similar tasks as your original RandomForest code.

First, make sure you have the `gbm` package installed. If not, you can install it with:

R

[Copy code](#)

```
install.packages("gbm")
```

Here's the R code for a Gradient Boosting model using `gbm`:

R

[Copy code](#)

## Library(gbm)

```
# Gradient Boosting Model
set.seed(123) # for reproducibility
dm_model <- gbm(dm_model_formula,
  data=dm_traindf,
  distribution="bernoulli", # Use "multinomial" for more than two classes
  n.trees=100,
  interaction.depth=1,
  n.minobsinnode=10,
  shrinkage=0.1,
  cv.folds=5,
  verbose=FALSE)

# Summary of the model
summary(dm_model)

# Prediction
pred <- predict(dm_model, dm_inputdf, type="response", n.trees=100)
dm_scoreddf <- data.frame(P_BAD1=pred, P_BAD0=1-pred)
colnames(dm_scoreddf) <- c("P_BAD0", "P_BAD1")

# Print/plot model output
plot(dm_model, main='GBM Deviance Plot')

# Save variable importance
importance <- summary(dm_model, cBars=10, plot=FALSE) # Adjust cBars to control the
# number of variables displayed
write.csv(importance, file="rpt_gbmIMP.csv", row.names=TRUE)
```

Key Changes and Details:

1. **Model Setup:** The `gbm` function is used to fit a gradient boosting model. The `distribution` parameter is set to "`bernoulli`" for binary classification (use "`multinomial`" for multi-class scenarios).
2. **Hyperparameters:**
  - `n.trees`: Number of trees to use in the model.
  - `interaction.depth`: Depth of each tree (often much shallower than random forests).
  - `n.minobsinnode`: Minimum number of observations in the nodes.
  - `shrinkage`: Learning rate.

- **cv.folds:** Number of folds for cross-validation.
3. **Prediction:** Predictions are made using the `predict` function. The `type="response"` argument in the `predict` function ensures you get probability estimates. These are then used to calculate `P_BAD0` and `P_BAD1`.
  4. **Plotting:** A plot is generated to show the deviance of the model over iterations, which can help in assessing model fit and overfitting.
  5. **Variable Importance:** The variable importance is saved similarly to how it was done with the RandomForest.

This code should get you started with gradient boosting in R. Adjust the model's parameters and settings based on your specific data and the results of initial model evaluations.

Thanks! How about one more. Can you adjust the following R code to run a decision tree instead? Thanks!

```
library(randomForest)

# RandomForest
dm_model <- randomForest(dm_model_formula, ntree=100, mtry=5, data=dm_traindf,
importance=TRUE)

# Score
pred <- predict(dm_model, dm_inputdf, type="prob")
dm_scoreddf <- data.frame(pred)
colnames(dm_scoreddf) <- c("P_BAD0", "P_BAD1")

# Print/plot model output
plot(dm_model, main='randomForest MSE Plot')
dev.off()

write.csv(importance(dm_model), file="rpt_forestIMP.csv", row.names=TRUE)
```

### ChatGPT

To convert your existing R code from using a RandomForest to a Decision Tree model, you can use the `rpart` package, which is widely used for building classification and regression trees in R. Below, I'll provide the adjusted code to use a decision tree model instead of RandomForest.

First, ensure that you have the `rpart` package installed. If not, you can install it by running:

R

[Copy code](#)

```
install.packages("rpart")
```

Here's the R code for a Decision Tree using `rpart`:

[R](#)[Copy code](#)

```
library(rpart)
```

```
# Decision Tree Model
```

```
dm_model <- rpart(dm_model_formula, data=dm_traindf, method="class")
```

```
# Prediction
```

```
pred <- predict(dm_model, dm_inputdf, type="prob")
```

```
dm_scoreddf <- data.frame(pred)
```

```
colnames(dm_scoreddf) <- c("P_BAD0", "P_BAD1")
```

```
# Print/plot model output
```

```
plot(dm_model, main='Decision Tree Plot')
```

```
text(dm_model, use.n=TRUE, cex=0.6) # Adds node labels with sample sizes to the plot
```

```
# Variable importance is not as straightforward in rpart as in randomForest, but we can retrieve overall variable usage
```

```
importance <- as.data.frame(dm_model$variable.importance, row.names=TRUE)
```

```
write.csv(importance, file="rpt_treeIMP.csv", row.names=TRUE)
```

#### Key Changes and Details:

1. **Model Setup:** The `rpart` function is used for constructing a decision tree. The `method="class"` parameter specifies that the model is for classification. For regression tasks, you would use `method="anova"`.
2. **Prediction:** The `predict` function is used with `type="prob"` to obtain predicted class probabilities, which are then organized into `P_BAD0` and `P_BAD1`.
3. **Plotting:** The `plot` and `text` functions are used to visualize the decision tree structure. The `text` function adds detailed node labels to help interpret the tree.
4. **Variable Importance:** `rpart` does not provide an automatic importance metric like `randomForest`. However, the `variable.importance` attribute in the model object gives a measure of variable importance based on the improvement to the model's fit contributed by splits on each variable.

This modified code will fit a decision tree to your data, predict class probabilities, and plot the decision tree structure, providing a straightforward transition from the RandomForest model. Adjust parameters and visualization options as necessary to optimize the model and enhance interpretability based on your specific dataset and needs.



Great - thanks!

ChatGPT

You're welcome! If you have any more questions or need further assistance with your R code or anything else, feel free to ask. Happy coding!