SAS Viya Workbench Hackathon Bootcamp | Step-By-Step Instructions

SAS Innovate 2025: Orlando, FL
Tuesday, May 6 | 1 p.m. – 4 p.m.

Backdrop

Welcome to the SAS Viya Workbench Hackathon Bootcamp! This special annual event welcomes developers, programmers and data scientists to SAS' technology to test-drive the latest innovations. With technical specialists from SAS, you'll have a chance to learn how Viya Workbench will simplify and streamline your model building processes. We'll introduce concepts that you can use to solve business problems as well as give you an opportunity to learn from a community of your peers.

The code for today's event comes from SAS Education's new course, Modern Data Science with SAS Viya Workbench and Python. The full course is much longer and covers the entire analytics lifecycle – from data wrangling to model deployment. However, this activity will focus on the computationally intensive portion – and admittedly the fun part. So, please enjoy the modeling sections of the course and provide your feedback along the way. Thanks!

The Scenario

Welcome to your first day at Telco, a fast-rising telecom company, that is heavily dependent upon monthly subscriptions.

As a data scientist in the Customer Retention Department, you use machine learning models to analyze customer churn. "Churn" simply means that a customer has canceled their premium cellular subscription. And since it's often more difficult to find a new customer than to keep an existing one, the company relies on your expertise to identify which clients are on the cusp of churning as to find ways to retain them.

Your onboarding will take several weeks, but we'll get you started today by examining the work of your predecessor. This work will expose you to the typical activities and challenges you'll face in your daily job.

For our lesson today, we will walk you through our customer retention project, which follows the classic analytics lifecycle. The idea isn't to have you understand everything happening within the data prep and modeling phases, rather, the goal is to expose you to the wide data engineering and data science universe so that you can follow-up with more detailed analysis later.

Finally, we'll introduce you to your new coding environment, SAS Viya Workbench. It's a lightweight, standalone development, modeling and coding environment for Python, R and SAS coders.
I hope that you're as excited as I am… so let's get started!

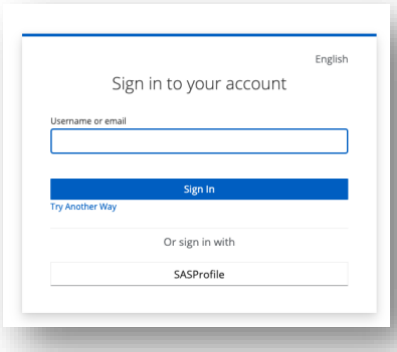
Table of Contents

How to Get Started in SAS Viya Workbench

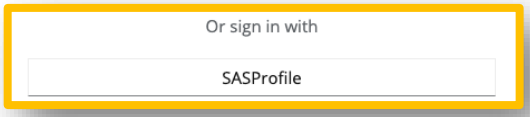
Our first section is all about access to software and setting up the data required for your coding adventure. So, this section is truly the beginning… and needs to be followed exactly as specified below. And then you are just a few clicks away from running all the programs required to complete today's coding challenge.

Like any good analytics adventure, we start by logging in <see provided URL>

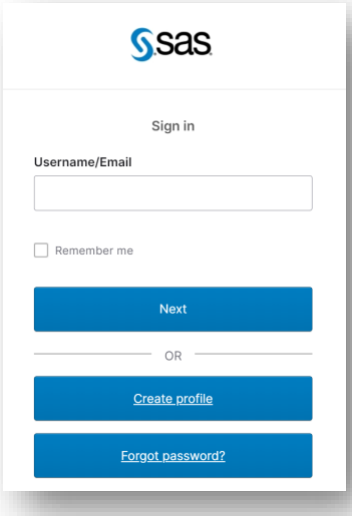
- The following login screen should then appear:



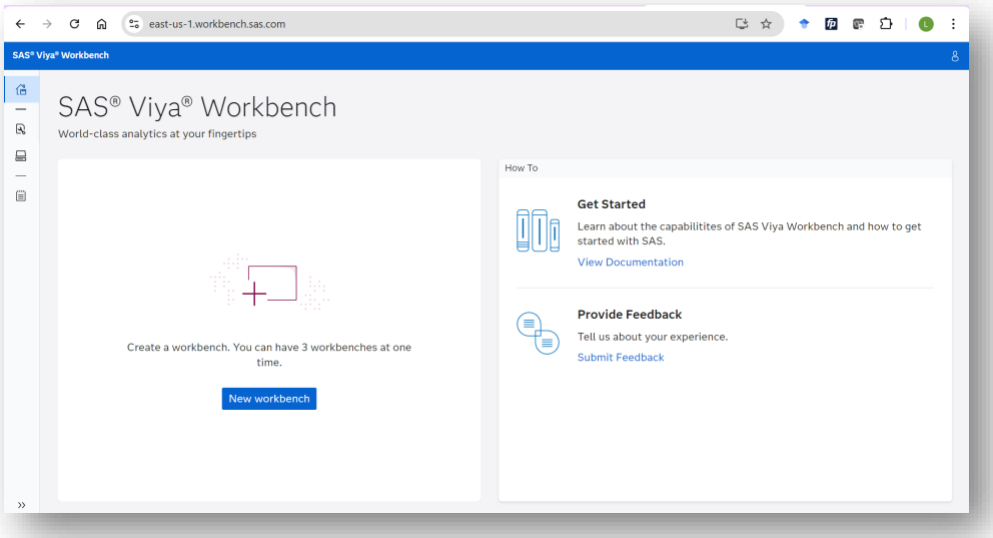
- Click "SASProfile"



- You will then be promoted to sign in with your SAS Profile

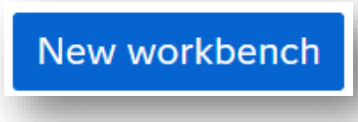


- Enter your credentials instead. With access, you should land here:

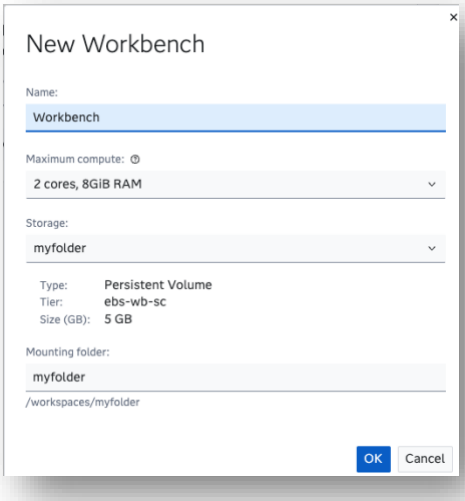


- *EXPLORE anything you find interesting*. For example, check out the **Get Started** resources including Documentation, or those **Workbenches** and **Storage** icons on the left side. In other words, just become more and more comfortable with the landing page.

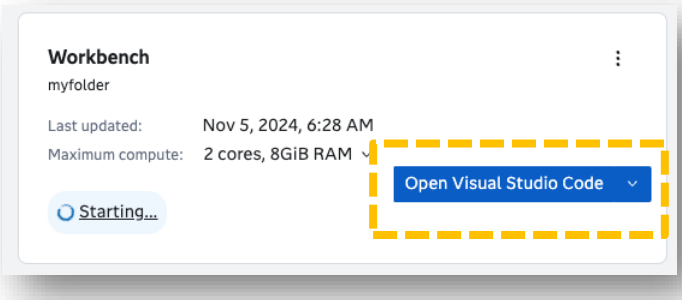
- And when you're ready, click that **New workbench** button:



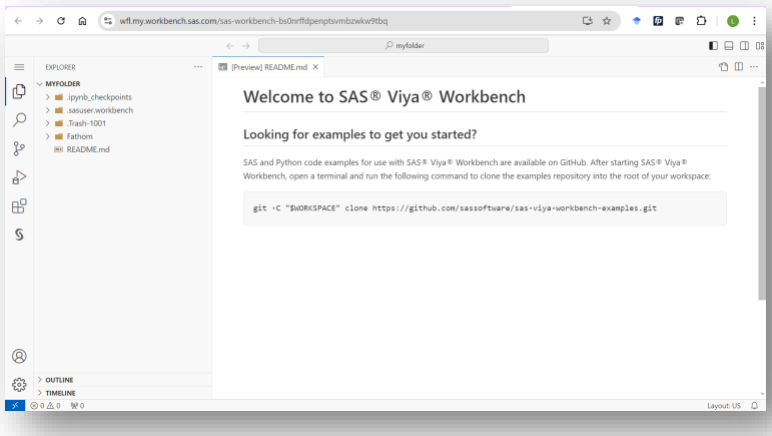
- The following appears. Keep the Name & other settings at their default and then click **OK**.



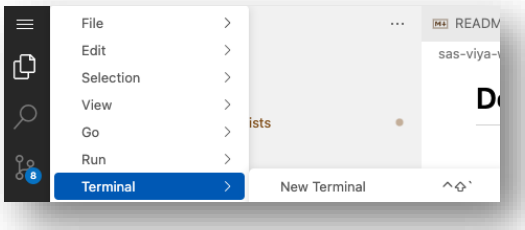
- Next, click the **Open Visual Studio Code** button:



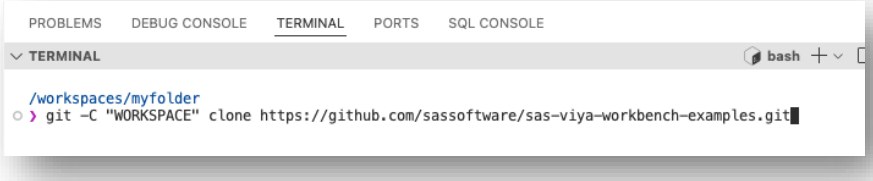
- Your workbench container loads! And welcome to **Visual Studio**!



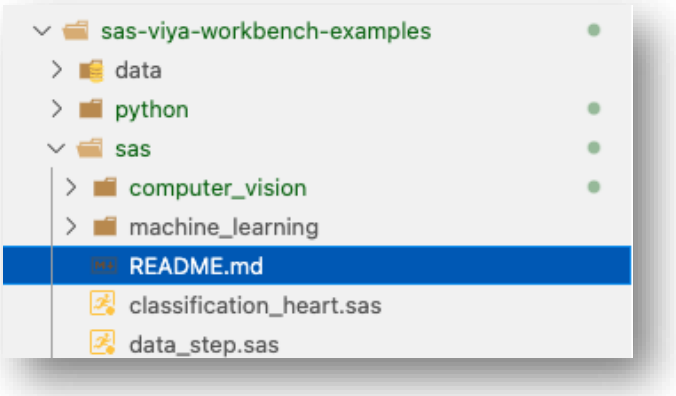
- Next, we will load examples from the SAS Viya Workbench GitHub Repository
- Open a Terminal



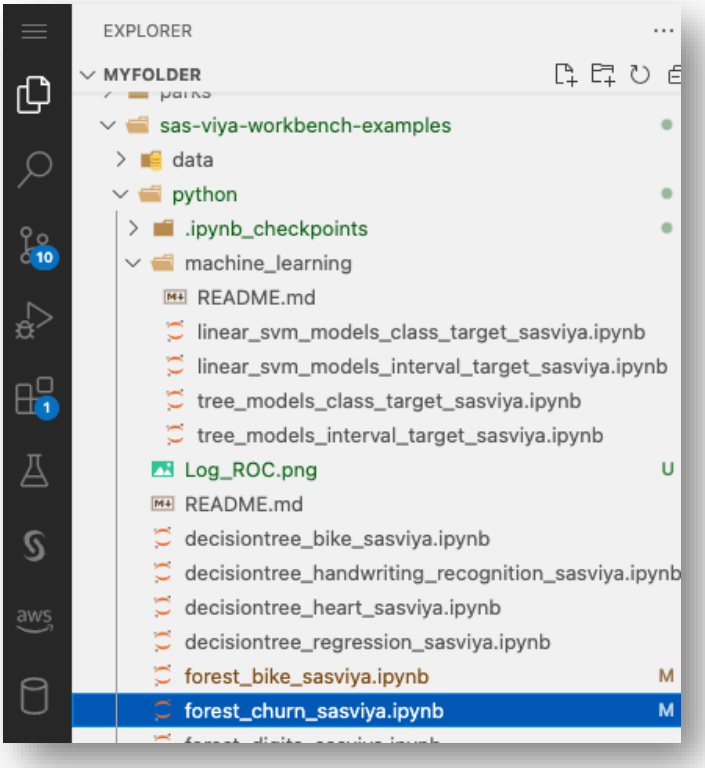
- Paste this command into the terminal window & hit enter
  git -C "$WORKSPACE" clone https://github.com/sassoftware/sas-viya-workbench-examples.git



- Your file explorer (left-side of VSCode screen) should look like the following when complete:

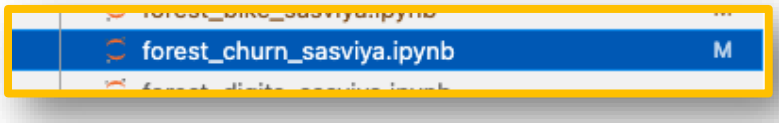


- In the upcoming sections, we will walk through a few specific examples. Throughout the day, feel free to try the other examples we have provided for you in this repository!!!

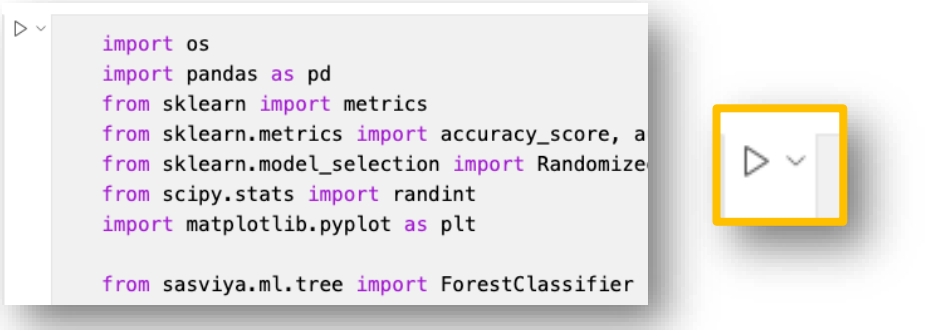


Run Your First Model: “Random Forest”

- Open forest_churn_sasviya.ipynb

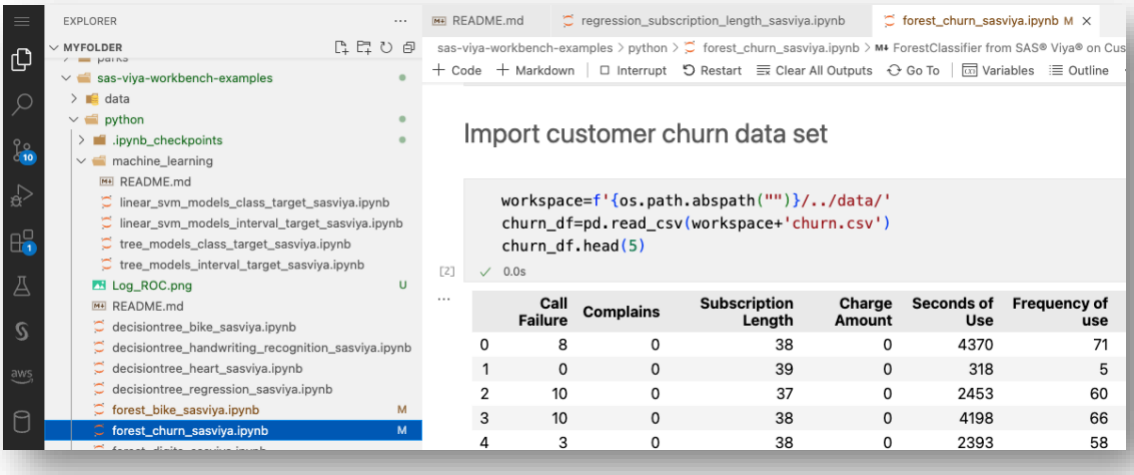


- We will run each cell within this notebook



- You may need to confirm the python version to run. Select “Python 3.11.9” to proceed

- Take a quick pause once you have loaded the churn data set into a data frame (df)



If you are feeling lost, take time to familiarize yourself with loading data in pandas
https://pandas.pydata.org/docs/user_guide/10min.html

Tip: Create a new .ipynb file and load sample data into a pandas data frame (df)

- OK – once we have loaded our data into a data frame, we're going to split the data into Training/Testing sets. We then train a baseline model immediately! Here's your first Random Forest Model we've called "rf"

```
rf = ForestClassifier(n_estimators=100,
                      max_depth=5,
                      min_samples_leaf=1,
                      max_features=None,
                      criterion='gini',
                      random_state=0)

rf.fit(X_train, y_train)
```

[4] Python

```
ForestClassifier(criterion="gini", max_depth=5, min_samples_leaf=1, random_state=0)
```

If you have never built a Machine Learning model, take a look at scikitlearn's documentation:
https://scikit-learn.org/stable/getting_started.html
Click here to learn more about the modeling techniques used in this example.

- We now have a baseline model, but we need to evaluate it against the Test data set.

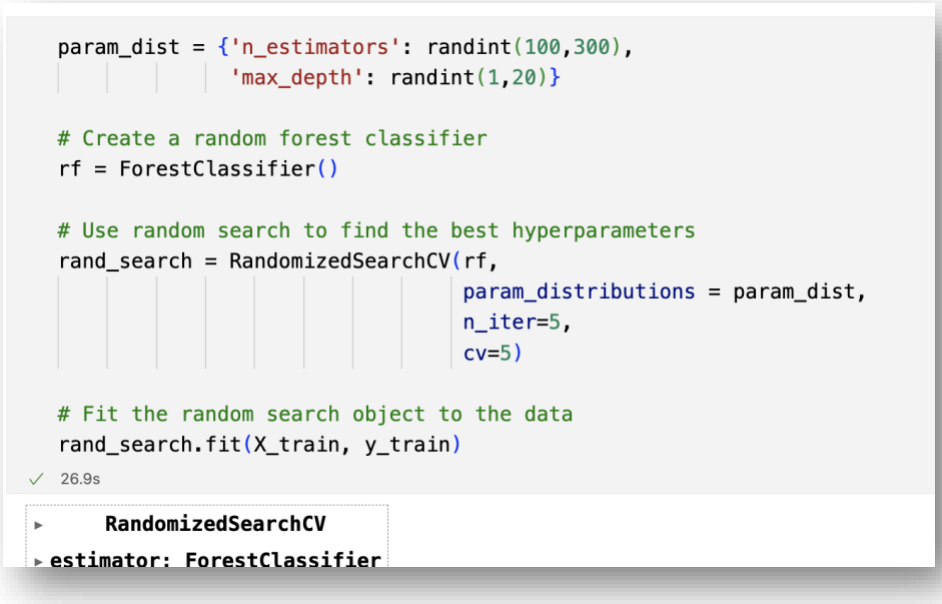
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Accuracy:", '{:.4f}'.format(accuracy))
print("Precision:", '{:.4f}'.format(precision))
print("Recall:", '{:.4f}'.format(recall))
```

✓ 0.0s

```
Accuracy: 0.9317
Precision: 0.8256
Recall: 0.7172
```

The block of code above shows our model performed pretty well. 93% accuracy score is high. Experienced data scientists will be quick to explain this demonstrates we are using a classic example data set that is very clean. Real life models are often never this accurate – especially with minimal effort.

- Even if our model looks pretty good, let's see if we can improve it with hyperparameter tuning.



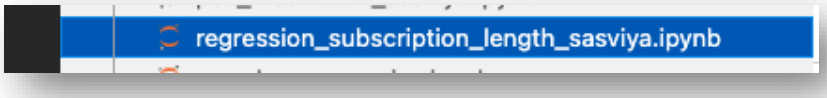
Did you notice how this step took a little more time than the previous steps? Optimal parameter selection can be tremendously valuable, but in many cases the cost of identifying the ideal parameters is expensive. In future examples we will dive into performance.

- Our first example wraps up by diving further into the analytics lifecycle by using a popular technique that determines the most important features of the model, which can be incredibly helpful in building subsequent models / defining further experiments.

- CONGRATS – YOU HAVE COMPLETED THE FIRST EXAMPLE. See the next page for another example

Run a Second Model: Focus on Visual Exploration

Open “regression_subscription_length_sasviya.ipynb”



- Data Exploration and Visualization is often the first place to start in the analytics lifecycle. In the first example, we dove right into modeling. Let’s slow down and understand our data better!

- First, we will use the pandas “describe” method

View the distribution of the data

```
numeric_X_df = subscrlength_df.select_dtypes(exclude=['object'])
numeric_X_df.describe().T
```

✓ 0.0s

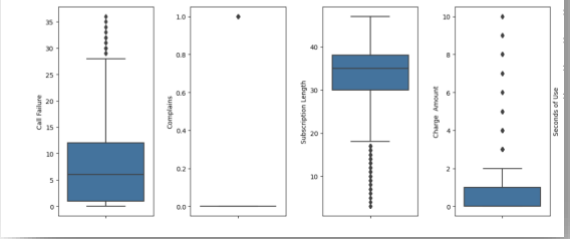
	count	mean	std	min	25%	50%	75%	max
Call Failure	3150.0	7.627937	7.263886	0.0	1.00000	6.00	12.00000	36.00
Complains	3150.0	0.076508	0.265851	0.0	0.00000	0.00	0.00000	1.00
Subscription Length	3150.0	32.541905	8.573482	3.0	30.00000	35.00	38.00000	47.00
Charge Amount	3150.0	0.942857	1.521072	0.0	0.00000	0.00	1.00000	10.00
Seconds of Use	3150.0	4472.459683	4197.908687	0.0	1391.25000	2990.00	6478.25000	17090.00
Frequency of use	3150.0	69.460635	57.413308	0.0	27.00000	54.00	95.00000	255.00
Frequency of SMS	3150.0	73.174921	112.237560	0.0	6.00000	21.00	87.00000	522.00
Distinct Called Numbers	3150.0	23.509841	17.217337	0.0	10.00000	21.00	34.00000	97.00
Age Group	3150.0	2.826032	0.892555	1.0	2.00000	3.00	3.00000	5.00
Tariff Plan	3150.0	1.077778	0.267864	1.0	1.00000	1.00	1.00000	2.00
Status	3150.0	1.248254	0.432069	1.0	1.00000	1.00	1.00000	2.00

- Now let’s visualize the data – we will use the popular python package, **seaborn** which is referenced as “sns” in the following examples

  - Box plot – look for outliers

```
fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in numeric_X_df.items():
    sns.boxplot(y=k, data=numeric_X_df, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```

✓ 1.5s

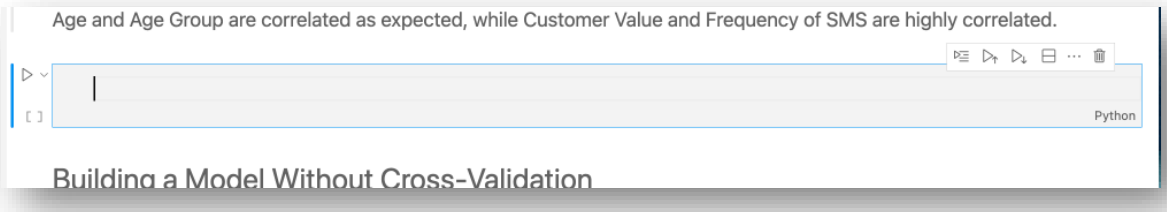


- Heat Map – visualize the correlation between variables

CHALLENGE:

- **Build your own plot with seaborn**
  - Add a cell into your notebook after the Heat Map
    - Before

    - After

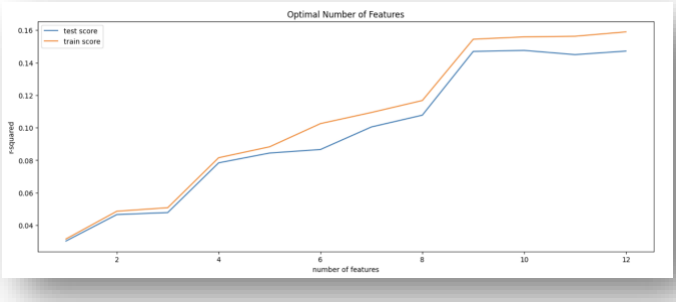


- *If you aren't sure what to try, paste the following into the cell and run it:*

```
for k,v in numeric_X_df.items():
  sns.histplot(data=numeric_X_df, x=v, multiple="stack")
  index += 1
  plt.show()
```

- Now that we have a sense for the data, we will start preparing a model
    - Feature Selection with a technique called Recursive Feature Elimination
    - Hyperparameter tuning with Grid Search Cross-Validation

- After we created the model, we plot the cross-validation results.



- The R2 value does not increase beyond 10 features, so we put this as a limit in the final model

```
# final model
n_features_optimal = 10

lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, n_features_to_select=n_features_optimal)
rfe = rfe.fit(X_train, y_train)

# predict prices of X_test
y_pred = lm.predict(X_test)
r2 = sklearn.metrics.r2_score(y_test, y_pred)
print('r2:', '{:.4f}'.format(r2))
```

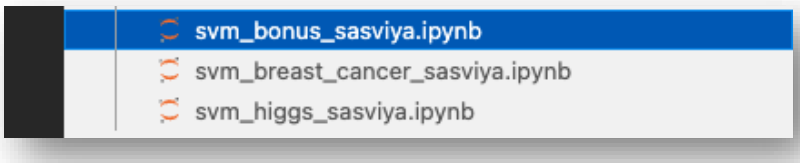
```
r2: 0.1411
```

- CONGRATS – YOU HAVE COMPLETED THE SECOND EXAMPLE. See the next page for another example

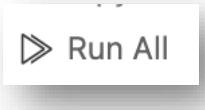
Run a Third Model: SAS Viya Comparison with scikit-learn

- *Scenario: Your coworker has built a very cool model to project your bonus next year. The better the model can fit the training data, the more likely you will see a better bonus. Let's see if your coworker made the right decision to leverage sasviya models rather than scikit-learn models*

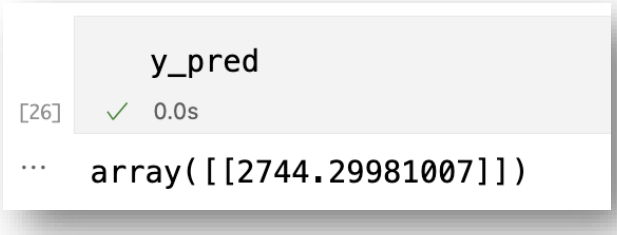
- Open "svm_bonus_sasviya.ipynb"



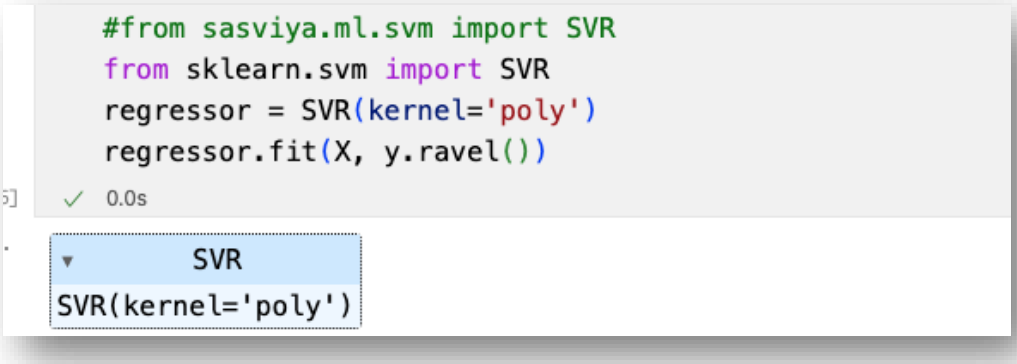
- First, run the whole notebook.



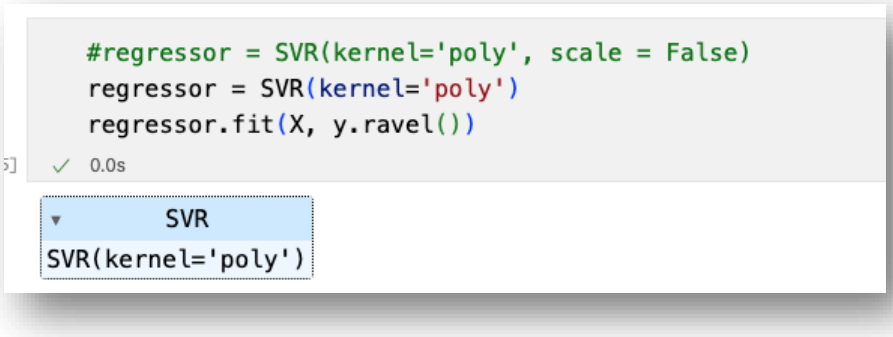
- Take note of at the output that was generated using the sasviya models.
  - y_pred = 2744 is within the expected range so the model looks valid/useful



- Modify this import statement to swap over to scikit-learn



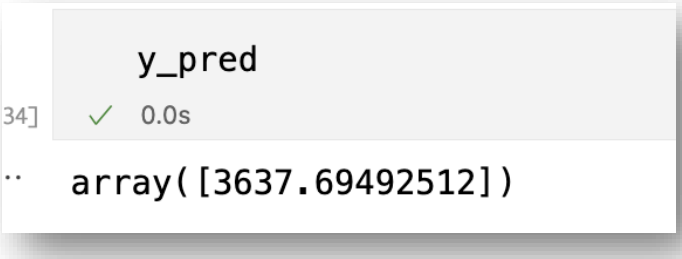
- Next, we'll need to remove the parameter "scale = False"



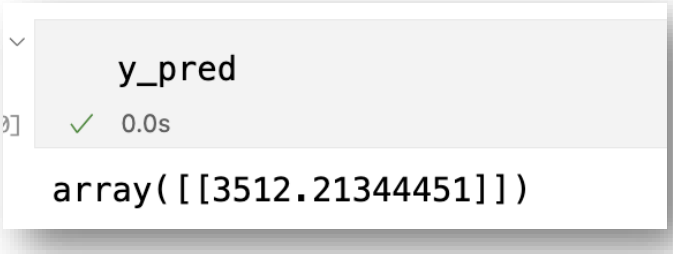
- This is an example where SAS contains more parameters than scikit-learn. If you are curious, here's more documentation about the SVR parameters.

- Now let's re-run the model.

- The first evaluation looks to be outside of our acceptable range. We are at risk of over estimating the bonus for select job types



- Even after additional clean-up we only see the following value, which still is outside the acceptable range



- CONGRATS – YOU HAVE COMPLETED THE THIRD EXAMPLE. See the next page for another example

Try it with SAS Code

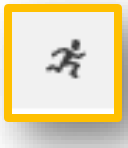
- Now that you have seen the power of the sasviya python package, let's take a look at a few examples of SAS code

First SAS Example. The Classic Cowboy Hat

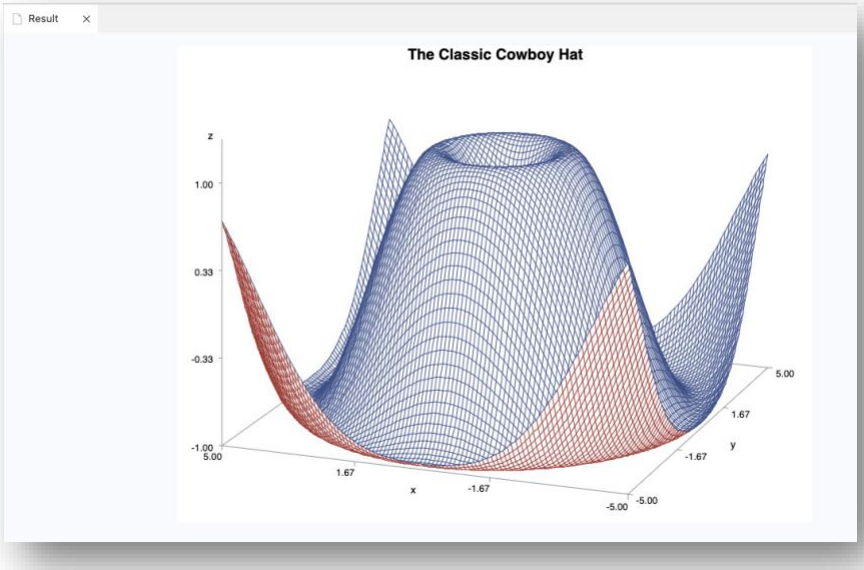
- Create a new file called "hello.sas"
- Copy and Paste these next 4 lines

```
title "The Classic Cowboy Hat";
data a; do x=-5 to 5 by .1; do y=-5 to 5 by .1;
z=sin(sqrt(x*x+y*y)); output; end; end;
proc g3d; Plot x*y=z; run;
```

  - Click Run (look for running person symbol in upper-right corner)!



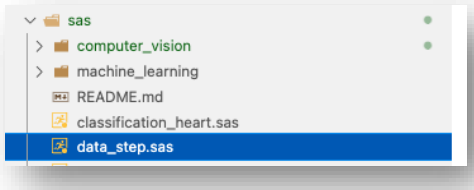
- The result is a fun looking visual that looks a lot like a cowboy hat!



- Data Step is a powerful feature of the SAS Language. Let's take a closer look in the next example.

Second SAS Example: data step

- Let's open "data_step.sas"



- We once again use the data step to load our data and use it to prepare a model
- The example first uses data step to load data from a file
- Then we use data step to create a data set

```
data employees;
    input name :$8. jobcode;
datalines;
Arthur 3
Bob    1
Carol  5
David  3
Edison 7
;
```

- Next, we use a SAS procedure "proc sort" to sort the data
    - A SAS procedure (aka PROC) showcases the SAS Language – think of running a proc as calling library of analytical routines tailored for a massive amount of use-cases.
    - Click here to explore the SAS procedures available in Workbench today.

- We then use data step to merge both data sets together

```
data employees_bonus;
    merge employees(in=in_employees) bonus;
    by jobcode;
    if in_employees;
run;
```

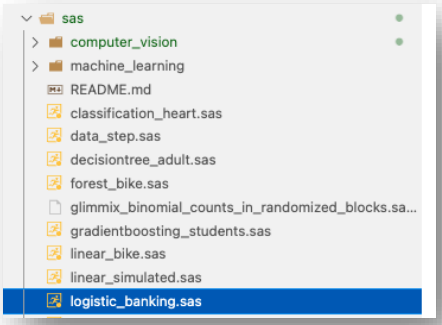
- Scroll down to line 128. Take a look at “proc sql”

```
128  proc sql;
129      create table employees_bonus2 as
130      select employees.*, bonus from employees as emp left join bonus as
131      on emp.jobcode = bon.jobcode;
132  quit;
133
134  title2 'Merged data from employees and bonus by PROC SQL';
135  proc print data=employees_bonus2;
136  run;
```

- “proc sql” executes SQL through the SAS language.
    - See the Advanced Challenge to learn more about proc SQL

Third SAS Example: Run a SAS Model

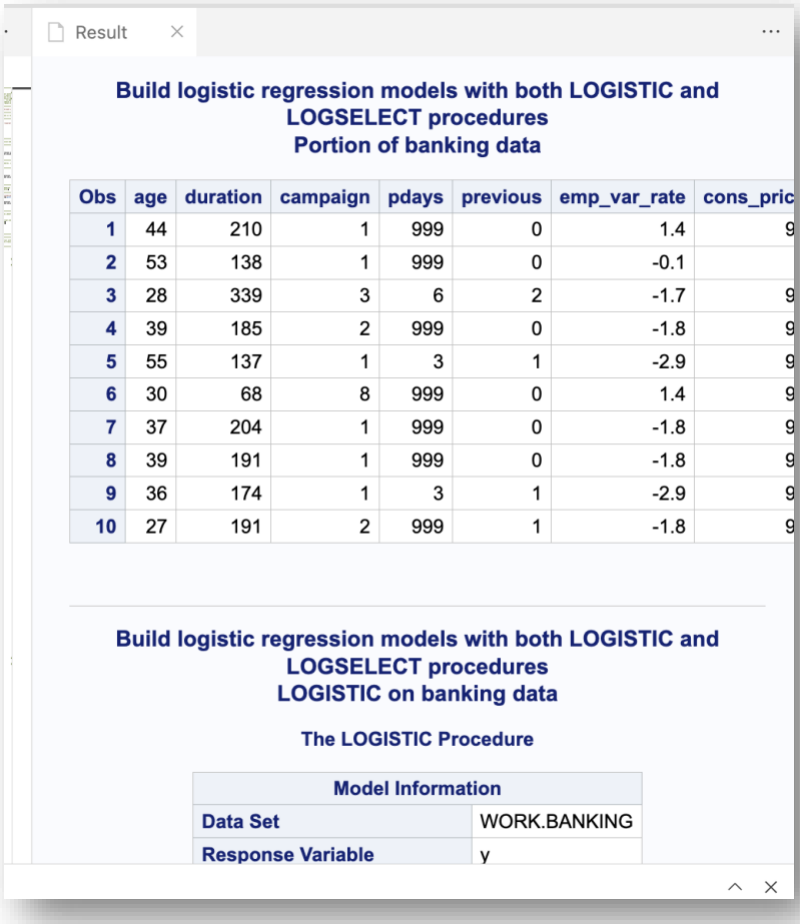
- Now that we’re familiar with SAS Procedures, let’s keep moving ahead and put it together with the python modeling techniques we just saw
- Let’s open “logistic_banking.sas”



- We load data with *proc import* and display the first 10 observations with *proc print*

```
options nosource;
proc import datafile="&WORKSPACE_PATH./sas-viya-workbench-examples/data/banking.csv"
   out=banking dbms=csv replace;
run;
options source;

title2 'Portion of banking data';
proc print data=banking(obs=10);
run;
```

- We then build a baseline logistic regression model.
- First using *proc logistic* and the next *proc logselect*.
    - ***The logselect example showcases the newest analytics from SAS Viya that is multi-threaded for faster performance.***



Result

Build logistic regression models with both LOGISTIC and LOGSELECT procedures
Portion of banking data

Obs	age	duration	campaign	pdays	previous	emp_var_rate	cons_pric
1	44	210	1	999	0	1.4	9
2	53	138	1	999	0	-0.1	
3	28	339	3	6	2	-1.7	9
4	39	185	2	999	0	-1.8	9
5	55	137	1	3	1	-2.9	9
6	30	68	8	999	0	1.4	9
7	37	204	1	999	0	-1.8	9
8	39	191	1	999	0	-1.8	9
9	36	174	1	3	1	-2.9	9
10	27	191	2	999	1	-1.8	9

Build logistic regression models with both LOGISTIC and LOGSELECT procedures
LOGISTIC on banking data

The LOGISTIC Procedure

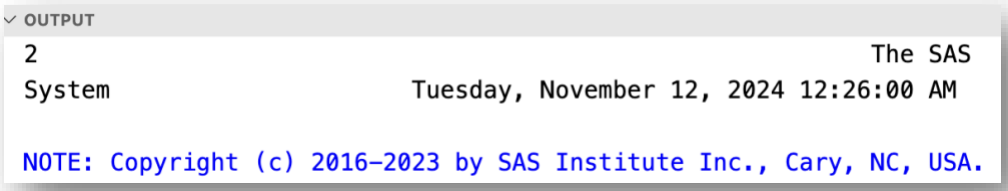
Model Information	
Data Set	WORK.BANKING
Response Variable	y

- Take time to review the results table.

- SAS makes it easy to generate output by providing a lot of details by default.

Task Timing		
Task	**Seconds**	**Percent**
Setup and Parsing	0.10	12.97%
Levelization	0.19	24.38%
Model Initialization	0.00	0.22%
SSCP Computation	0.21	26.01%
Model Fitting	0.29	36.41%
Display	0.00	0.00%
Cleanup	0.00	0.00%
Total	0.79	100.00%

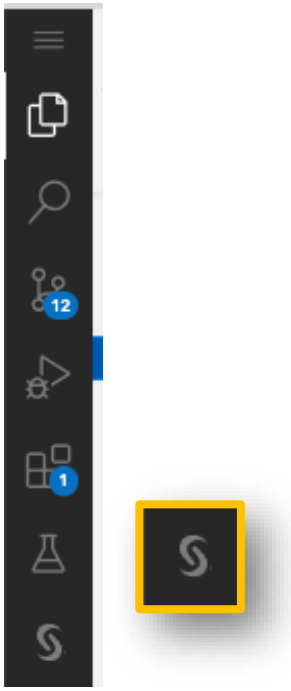
- Also take a look at the SAS Log!



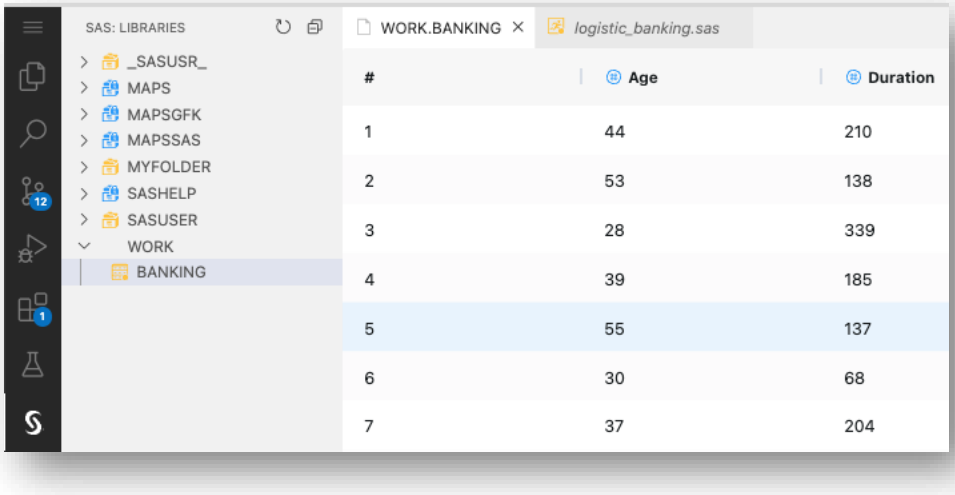

```
A short summary of the key results from the LOGISTIC and LOGSELECT procedures.

From LOGISTIC:
Model Fit Statistics
Criterion      Intercept Only   Intercept and Covariates
-2 Log L       28998.724         17077.827
AIC            29000.724         17183.827

From LOGSELECT:
Fit Statistics
-2 Log Likelihood            17078
AIC (smaller is better)      17184
```

- You may be wondering, "*….but what about the data? Where did all of this output come from?*"

- Click on the SAS icon on the left-hand menu to view the data that was used by the SAS Compute engine



- Expand the Work Directory to view the Banking data table that was used in the examples



- CONGRATS – YOU HAVE COMPLETED THE FINAL EXAMPLE. If you are feeling adventurous, take a look at the Advanced Activities!

Try it with R Code

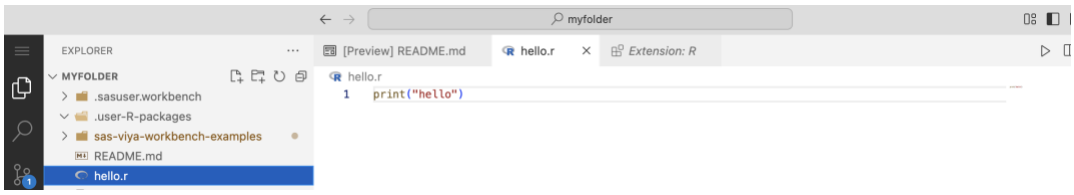
- Now that you have seen the power of SAS and Python, let's try running R code

First R Example. Explore UCI data

https://cran.r-project.org/web/packages/ucimlrepo/index.html

- Create a new file called "hello.r"
- Copy and Paste these next 4 lines

```
.....
.....
....
```



- Click "Play" (look for Arrow symbol in upper-right corner)!

Create an R Shiny Application

- Shiny is a powerful package that allows you to create simple interactive applications
- Enable it with a simple command: *install.packages("shiny")*
- https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/

Advanced Activities

Welcome to the Advanced Section! A few advanced python topics are included below along with additional training materials for learning SAS.

Use the Requests Library to Get Data from the Internet

- Try the following example pointed to a generic JSON record

```
import requests


# Making a GET request
r = requests.get('https://api.github.com/users/naveenkrnl')


# check status code for response received
# success code - 200
print(r)


# print content of request
print(r.content)
```

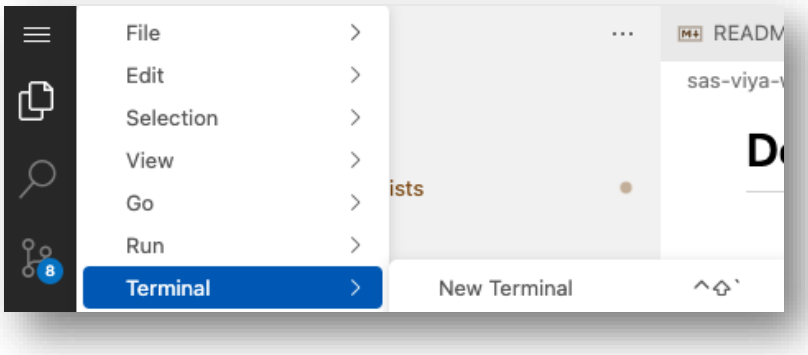
Source: https://www.geeksforgeeks.org/python-requests-tutorial/

- Next, use the pandas library to parse the response and load it into a data frame

Install your own Python Packages

- SAS Viya Workbench comes with a wide range of pre-installed packages, but you may want to install more packages.

- Open a Terminal (File → Terminal → New Terminal)



- Type the following command (insert a package name) and hit Enter

  ```
  pip install <package_name>
  ```

- Once you have installed your package(s), start using the packages by calling them within a python script.

Build your own SAS code

- Click here to explore the SAS procedures available in Workbench today.

Try a few these popular procs!

- Proc import

  Hint: use this syntax to easily load data from the existing examples

  ```
  &WORKSPACE_PATH./sas-viya-workbench-examples/data/
  ```

```
options nosource;
proc import datafile="&WORKSPACE_PATH./sas-viya-workbench-examples/data/banking.csv"
   out=banking dbms=csv replace;
run;
options source;

title2 'Portion of banking data';
proc print data=banking(obs=10);
run;
```

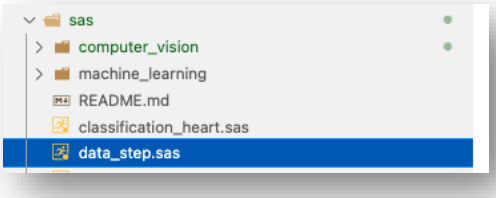
  Extra Challenge: use proc export to save some output (test your file i/o skills!)

- Proc freq (see documentation)

Advanced Challenge: Write your own PROC SQL

- Link to [proc sql documentation]

- Let's open "data_step.sas"



- Scroll down to line 128. Take a look at "proc sql"

```
128  proc sql;
129      create table employees_bonus2 as
130      select employees.*, bonus from employees as emp left join bonus as
131      on emp.jobcode = bon.jobcode;
132  quit;
133
134  title2 'Merged data from employees and bonus by PROC SQL';
135  proc print data=employees_bonus2;
136  run;
```

  - If you already know SQL, go ahead and try writing your own "proc sql" statement!
    - Can you create a new table?
    - Join a new table with the original?
    - *Put SAS to the test – are there any SQL actions you cannot perform?*

ODS Graphics

- SAS ODS (Output Delivery System) is a powerful and easy to use reporting and visualization engine. You already saw the reporting engine in action with the examples above. We'll focus here on the visualization

- First, take a moment to review the doc: **Creating Output and Graphics (doc home)**

- *Next, try a few Mini-challenges with proc sgplot*:

  - Create an Overlay Plot with Three Series Plots Using the SGPLOT Procedure
  - Create a Map with a Bubble Plot and a Series Plot Overlay

The following are some SG procedure examples overlaying a heatmap with a regression fit:

```
/* This is a range attribute map data set, used here to map colors to certain cholesterol ranges */
data cholmap;
   retain ID 'mymap';
   length rangecolor $ 6;
   input min $ max $ excludemax $ color $;
   cards;
   _MIN_ 200   true green
   200   240   true yellow
   240   _MAX_ false red2
;
run;

/* The bins in this plot are colored by the average cholesterol levels in each bin */
proc sgplot data=sashelp.heart rattrmap=cholmap;
   heatmap x=AgeCHDdiag y=weight / colorresponse=Cholesterol
                                   colorstat=mean rattrid=mymap;
   reg x=AgeCHDdiag y=weight / degree=3 nomarkers;
run;

/* Look at this same data broken down by smoking status */
proc sgpanel data=sashelp.heart rattrmap=cholmap;
   panelby smoking_status / novarname;
   heatmap x=AgeCHDdiag y=weight / colorresponse=Cholesterol
                                   colorstat=mean rattrid=mymap;
   reg x=AgeCHDdiag y=weight / degree=3 nomarkers;
run;

/* Look at this same data broken down by blood pressure status and gender – in a gridded panel */
proc sgpanel data=sashelp.heart rattrmap=cholmap;
   panelby bp_status sex / layout=lattice novarname onepanel;
   heatmap x=AgeCHDdiag y=weight / colorresponse=Cholesterol
                                   colorstat=mean rattrid=mymap;
   reg x=AgeCHDdiag y=weight / degree=3 nomarkers;
run;
```

Create your own Model Tournament from Scratch

- In the logistic banking example, we created 2 different logistic regression models. Typically in data science projects, a range of models are created to identify the best possible solution.

- Choose from a few other supervised machine learning models and compare the results with the logistic regression models.

  Hint: try these models
  - Gradient Boosting: proc gradboost
  - Decision Tree: proc treesplit