Technical Paper

Monitoring Turbofan Engine Degradation Using Stability Monitoring Procedures

THE POWER TO KNOW.

Release Information Content Version: 1.0 February 2019.

Contents

Introduction

Any machine that is operating in the field, from a coffee grinder in a kitchen to a spaceship far from Earth and everything in between, has components and subsystems that require maintenance in order to operate properly and without interruptions. The approach to maintenance always requires balancing concerns about cost and reliability. One approach is to collect enough data about operations (including failures) of multiple machines of the same type in order to build a robust statistical model of the expected life of the typical machine. If periodic maintenance is performed on the basis of a certain metric of a machine's life (such as miles driven or hours of operation), then more frequent maintenance means increased cost, and less frequent maintenance means increased probability of failure. Also, different parts of a machine have different expected life spans, which can complicate the maintenance schedule. An example is automobile maintenance: according to the manufacturer's recommendations for a particular vehicle, the oil and filter should be changed every 5,000 miles or 6 months (whichever comes first), and more comprehensive maintenance should be done every 30,000 miles.

A periodic maintenance approach requires the collection of sufficient data, including data about failures. Such an approach might not be applicable if a machine is relatively new or if the number of machines in the field is too small. Even for cars that are produced in large numbers and for a long time, adhering to the recommended maintenance schedule does not guarantee trouble-free operation: for example, the "check engine" light on a dashboard that is triggered by the car's computer in response to data from sensors (called on-board diagnostics) indicates the need for unscheduled maintenance. With the technical advancement of sensors, relying on sensor data for maintenance decisions becomes easier. This approach calls for rule-based maintenance. Some rules can be simple and can be based on data from a single sensor. For example, an engine should operate within a certain temperature range, and tires should maintain a certain range of pressure. However, some rules can be more complex and involve relationships among multiple sensors. For example, gas mileage depends on many factors, such as engine RPM, acceleration, oil temperature, tire pressure, cargo load, air temperature, and so on. Monitoring gas mileage in conjunction with other sensor readings can provide valuable information about the overall "health" of the car. You can build a statistical model for gas mileage as a function of the other sensors and use this model to monitor gas mileage. For example, deteriorating gas mileage that cannot be accounted for by other sensor data can be a sign of a problem that requires immediate attention, such as a worn belt or clogged fuel filter.

Stability Monitoring Procedures

Five stability monitoring procedures in SAS® Visual Forecasting (the SMSPEC, SMPROJECT, SMCALIB, SMSELECT, and SMSCORE procedures) implement a collection of anomaly detection methods that can be used to monitor the "health" of a system and detect anomalous behavior of various kinds of data, including sensor and event data such as data about failures. One of the goals of stability monitoring is to predict serious issues, such as catastrophic failures, before they occur in order to manage the cost and frequency of preventive maintenance.

The analytical approach of stability monitoring is based on the assumption that within a subset of sensor data, event data, or both (collectively called "signals") that are related to the system being monitored, there exists a "target" signal whose behavior can be explained by other signals (called "explanatory" signals). The main idea is that in a healthy system (that is, healthy during a stable period of time), there is a robust statistical relationship between the target signal and the explanatory signals. After the relevant statistical model is selected, fitted to historical data during stable periods, and saved using the SMCALIB procedure, it can then be used for monitoring data that are collected in a new time period. The stored model is used with new data that contain both target and explanatory signals to generate a forecast of a target signal from the explanatory signals (this process is called "scoring" and is performed using the SMSCORE procedure). The forecast is then compared to the actual values of the target signal, and according to a set of user-defined rules, anomalies in the behavior of the monitored system during the new time period can be flagged.

Stability monitoring can be used in a variety of domains, including the oil and gas industry (such as for oil well pumps), the power generation industry (such as for wind turbines), and so on.

The workflow of stability monitoring follows these steps:

1. Create one or more model specifications by using the SMSPEC procedure.
2. Create a project repository by using the SMPROJECT procedure. The project repository contains the full project description, which includes the following:
    a. training data table and its structure
    b. scoring data table and its structure
    c. one or more model specifications (generated by the SMSPEC procedure)
    d. statistical parameters such as holdout length and significance level
3. Run project calibration by using the SMCALIB procedure on the training data table.
4. Select one or more models for scoring by using the SMSELECT procedure on the basis of the calibration results.
5. Run project scoring (also called monitoring) by using the SMSCORE procedure on the scoring data table.

Turbofan Engine Data Example

This example demonstrates the use of stability monitoring procedures for analyzing turbofan engine data. The data come from the turbofan engine degradation simulation data set (Saxena and Goebel 2008), which is housed in a data repository that is maintained by the Prognostics Center of Excellence of the National Aeronautics and Space Administration (NASA).

The data were generated using NASA's Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) software, which simulates the behavior of a large commercial turbofan engine under various environmental, operating, and health conditions. The engine has five rotating parts, each of which can be subject to deterioration. The C-MAPSS software takes as inputs operating conditions such as altitude, speed, and sea-level temperature, in addition to engine health parameters. The outputs include various sensor responses. Each observation is a snapshot of a cycle (called a flight), and the data for each simulation are generated until engine failure. The full data in the repository consist of four training data sets with multiple engine simulations. The data set in this example is taken from training set #3 for engine 24. The example uses the following sensor responses: pressure at fan inlet (InletPressure); total pressure in bypass duct (TotPressure); corrected core speed (CoreSpeed); bleed enthalpy (Enthalpy); high-pressure (HPCBleed) and low-pressure (LPCBleed) turbine coolant bleeds; and ratio of fuel flow to static pressure at the high-pressure compressor outlet, which is used as the target variable (FuelRatio). As the engine degrades, the ability to achieve similar levels of operational performance is expected to be reflected in changes in fuel consumption. Therefore, a variable that is related to fuel consumption is selected as the target (the variable that is monitored to determine the health of the system). The following DATA step creates the data set **train**:

```
data train;
   input DateTime InletPressure TotPressure FuelRatio CoreSpeed Enthalpy HPCBleed
LPCBleed;
   datalines;
1 14.62 21.59 519.25 8116.67 390 38.78 23.2995
2 14.62 21.59 520.60 8124.89 391 39.12 23.3948
3 14.62 21.59 519.26 8112.68 390 38.89 23.3420
4 14.62 21.58 518.61 8119.49 389 38.83 23.2713
… more lines …
```

The data for this engine contain 494 observations (cycles). The first 90 cycles are used as the training period (for calibration using PROC SMCALIB), and three consecutive 50-cycle windows starting from cycle 91 are used for condition monitoring. The following DATA step loads the training data into a CAS session. This DATA step assumes that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.project_1;
   set train(obs=90);
   stable=1;
run;
```

Creating Model Specifications Using PROC SMSPEC

As you can see from the DATA step, stable periods for the training data need to be defined. Any observations that are marked as unstable are not used for the training.

Next, you need to create a model specification by using the SMSPEC procedure:

```
proc smspec;
   input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
   reg name=mycas.reg1;
   arima name=mycas.arima1;
run;
```

The previous statements request both ordinary least squares regression and ARIMA model specifications (stored as **mycas.reg1** and **mycas.arima1**, respectively). Both model specifications use all six explanatory variables, which are specified in the INPUT statement. Because the INPUT statement does not include a TRANSFORM= option or a REQUIRED= option, input variables are not transformed and none of them are required variables. The target variable does not need to be transformed; therefore, the TARGET statement is omitted.

Creating Project Repository Using PROC SMPROJECT

Next, you need to create a project repository by using the SMPROJECT procedure:

```
proc smproject name=mycas.myproj;
    target FuelRatio;
    input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
    model reg1 arima1;
run;
```

This PROC SMPROJECT call is run using mostly default values. It generates a project repository, **mycas.myproj** (specified in the NAME= option), that contains a single project whose ID is 1 by default (because no ID= option is specified in the PROC SMPROJECT statement). The project expects the training data table to be named **mycas.project_1** (because the DATA= option is not specified) and the scoring data set to be named **mycas.project_1_score** (because the SCOREDATA= option is not specified). The ALPHA= and HOLDOUT= options are omitted; therefore, the significance level is set at 0.05, and the holdout length is set at 10 observations. The project uses FuelRatio as a target variable (as specified in the TARGET statement), and it has six explanatory variables (specified in the INPUT statement). The DATETIME and STABLE statements are omitted; therefore, the datetime variable is named DateTime, and the stable flag variable is named Stable. The project uses two model specifications, **mycas.reg1** and **mycas.arima1**.

Calibrating the Project Using PROC SMCALIB

Next, you run the project calibration by using the SMCALIB procedure:

```
ods output holdoutstats=holdstat;
```

```
proc smcalib name = mycas.myproj;
   output modelfit   = mycas.mfit
          holdoutfit = mycas.hfit
          scoreinfo  = mycas.sinfo;
   store mycas.scoreout;
run;
```

The PROC SMCALIB statement looks for the **mycas.myproj** project repository. Because the PROJECT statement is omitted, all projects in that repository are calibrated. In this example, the repository contains only one project, whose ID is 1. The OUTPUT statement requests the following SAS® output tables to be generated: model fit, holdout fit, and scoring information. The STORE statement requests that information for each model necessary for scoring be stored in the binary table **mycas.scoreout**. This table is used in PROC SMSCORE during scoring, and it is also an analytic store (ASTORE) that can be used to score data in other SAS products, such as SAS® Event Stream Processing. ODS tables that contain the following information are also produced:

- calibration status
- fit statistics
- holdout statistics
- parameter estimates

The “Calibration Status” table indicates whether the calibration was successful for each model.

Calibration Status

Project ID	Model ID	Model Name	Status	RC	Message
1	0	Simple	0	0	Command successfully completed.
	1	arima1	0	0	Command successfully completed.
	2	reg1	0	0	Command successfully completed.

As you can see, three models were successfully calibrated. Two of the models were defined by the SMSPEC procedure, and the model whose ID is 0 is an internal model that uses only the target variable. Each model is given a numeric ID, where an ID of 0 is always an internal simple model and user models are numbered consecutively starting from 1. Model IDs are used later for scoring.

As you can see in the “Parameter Estimates” table, all variables except InletPressure are selected for both models 1 and 2.

Parameter Estimates

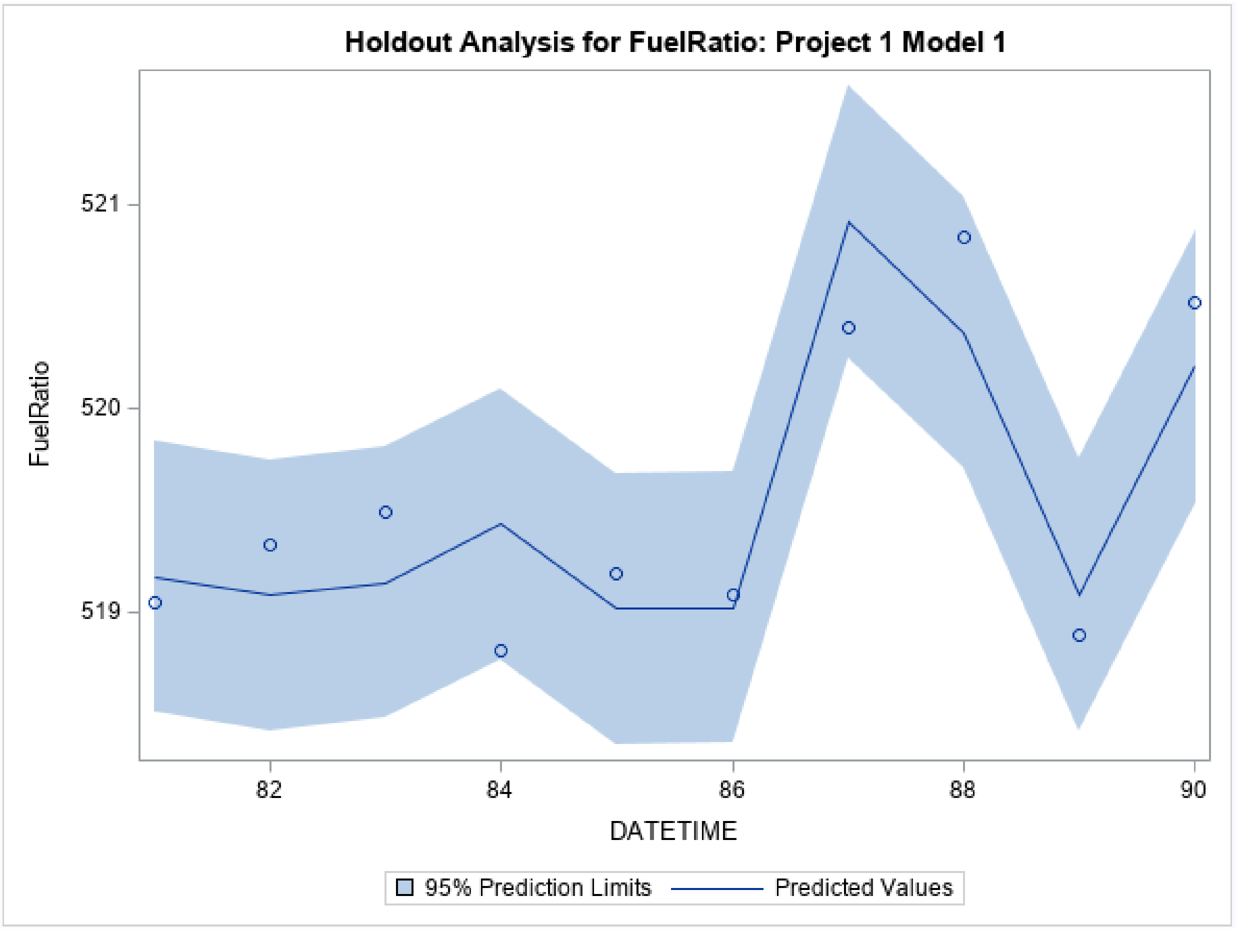
Project ID	Model ID	Variable	Transformation	Estimate	Standard Error	t Value	Pr > |t|
1	1	BypassTotPressure	None	27.2561	8.1585	3.34	0.0013
		Enthalpy	None	0.1385	0.0360	3.85	0.0002
		HPTCoolantBleed	None	1.3936	0.2823	4.94	<.0001
		LPTCoolantBleed	None	1.5820	0.4398	3.60	0.0006
		CoreSpeed	None	0.0398	0.007097	5.60	<.0001
	2	BypassTotPressure	None	35.3549	8.7616	4.04	0.0001
		Enthalpy	None	0.1412	0.0394	3.59	0.0006
		HPTCoolantBleed	None	1.2343	0.3315	3.72	0.0004
		LPTCoolantBleed	None	1.6191	0.5499	2.94	0.0042
		CoreSpeed	None	0.0420	0.008534	4.93	<.0001

Each model's potential for scoring is evaluated by holdout analysis, which is performed as part of the model calibration process. By default, the last 10 observations are used for the holdout analysis, and the significance level is set to 0.05. To request different values, you can specify the ALPHA= and HOLDOUT= options in the PROC SMPROJECT statement. The holdout analysis results are shown in the following table. Holdout statistics are slightly better for the ARIMA model, and both the ARIMA model and the REG model significantly outperform the default (Simple) model.

Holdout Statistics

Project ID	Model ID	Model Name	MAPE	RMSE
1	0	Simple	0.1185	0.7044
	1	arima1	0.0589	0.3514
	2	reg1	0.0593	0.3610

In addition to using holdout statistics, you can use the plot of holdout fit that is produced by PROC SMCALIB to evaluate a model's potential performance for scoring. The holdout fit for model 1 is shown in the following graph. All actual values are inside the 95% prediction band.



Selecting the Best Models Using PROC SMSELECT

To select one or more best-performing models for scoring, you can look at the holdout statistics and holdout fit plots. The selection can also be automated by using the SMSELECT procedure.

```
proc smselect name=mycas.myproj input=work.holdstat;
output out=work.mymodels;
run;
```

PROC SMSELECT uses the holdout statistics table **work.holdstat** (which is produced as an ODS output table when you run the SMCALIB procedure), specified in the INPUT= option. The model repository **mycas.myproj** must be specified in the NAME= option. The procedure reads the holdout statistics table and selects one or more (according to the value of the BEST= option from the SMPROJECT procedure) best models by using the holdout statistic criterion specified in the SELECT= option in PROC SMPROJECT. Both values can be overridden by specifying the corresponding options in the PROJECT statement in PROC SMSELECT. Because the PROJECT statement is omitted, the model selection is performed for all projects by using the model results found in the holdout statistics table. The IDs of the best-performing models for each project are stored in the **work.mymodels** table that is specified in the OUT= option in the OUTPUT statement. In this case, the output table has a single row, with model 1 for project 1.

Monitoring Using PROC SMSCORE

The scoring data for the first 50-cycle window (starting from cycle 91) are created in the following DATA step:

```
data mycas.project_1_score;
   set train(firstobs=91 obs=140);
run;
```

The name of the scoring data set, **mycas.project_1_score**, is stored in the project repository. This name can be modified using the SCOREDATA= option in the SMPROJECT procedure.

The following statements run the scoring for this window:

```
proc smscore       name = mycas.myproj
                   scorerepository=mycas.scoreout
                   models=work.mymodels;
   output out = mycas.score;
run;
```

The PROC SMSCORE statement looks for the project repository **mycas.myproj** (as specified in the required NAME= option) and for the binary file **mycas.scoreout** (which contains information necessary for scoring and is specified in the required SCOREREPOSITORY= option). The procedure performs scoring for the models whose IDs are stored in the **work.mymodels** table that is specified in the MODELS= option. As mentioned earlier, this table can contain the best-performing models for multiple projects. The PROJECT statement is omitted. The PROJECT statement can be used to specify which projects are to be scored (by using the ID= option); you can also directly specify models for scoring by using their IDs in the MODELID= option (in this case, the MODELS= option should be omitted from the PROC SMSCORE statement). Scoring produces predicted values of the target variable with prediction limits for each model within the project. The OUT= option in the OUTPUT statement stores the output in the **mycas.score** data table. The scoring results are also plotted in the following graph.

After that, you want to perform scoring for the next 50-cycle window, starting from cycle 141.

The data for scoring are created in the following DATA steps:

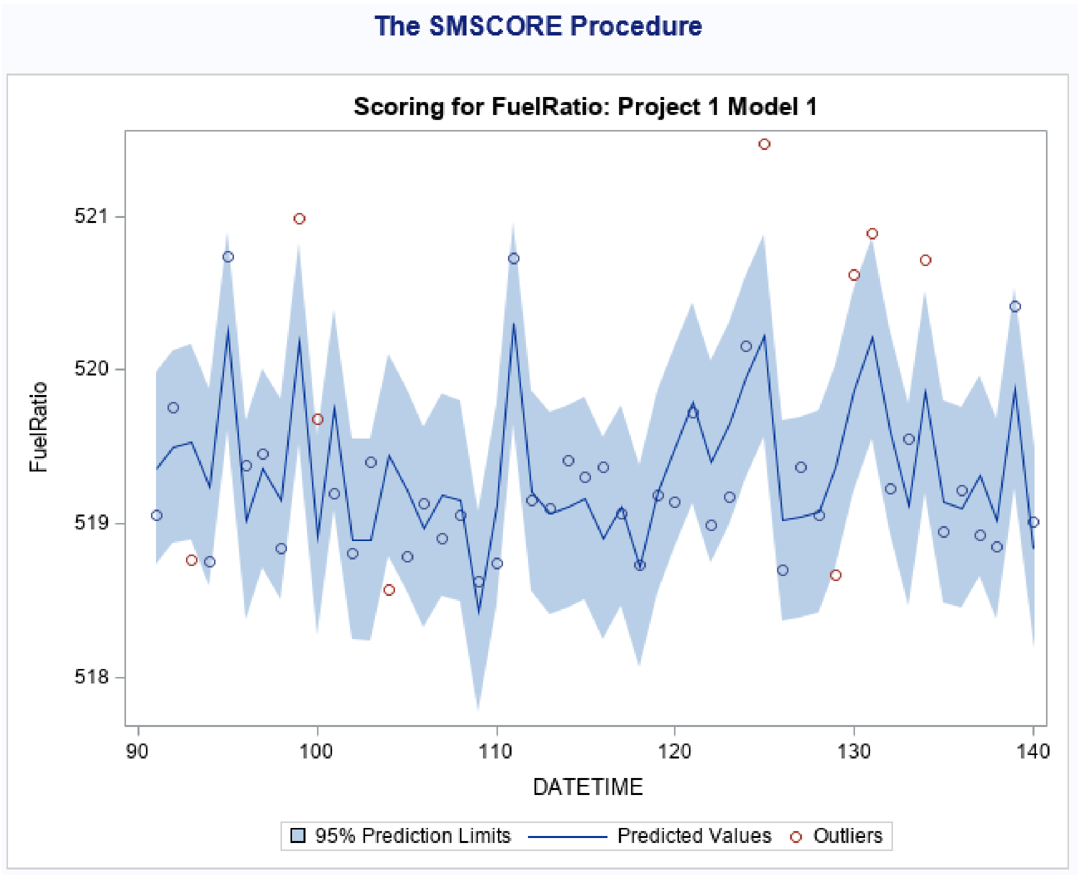
```
data mycas.project_1;
   set train(firstobs=91 obs=140);
run;

data mycas.project_1_score;
   set train(firstobs=141 obs=190);
run;
```

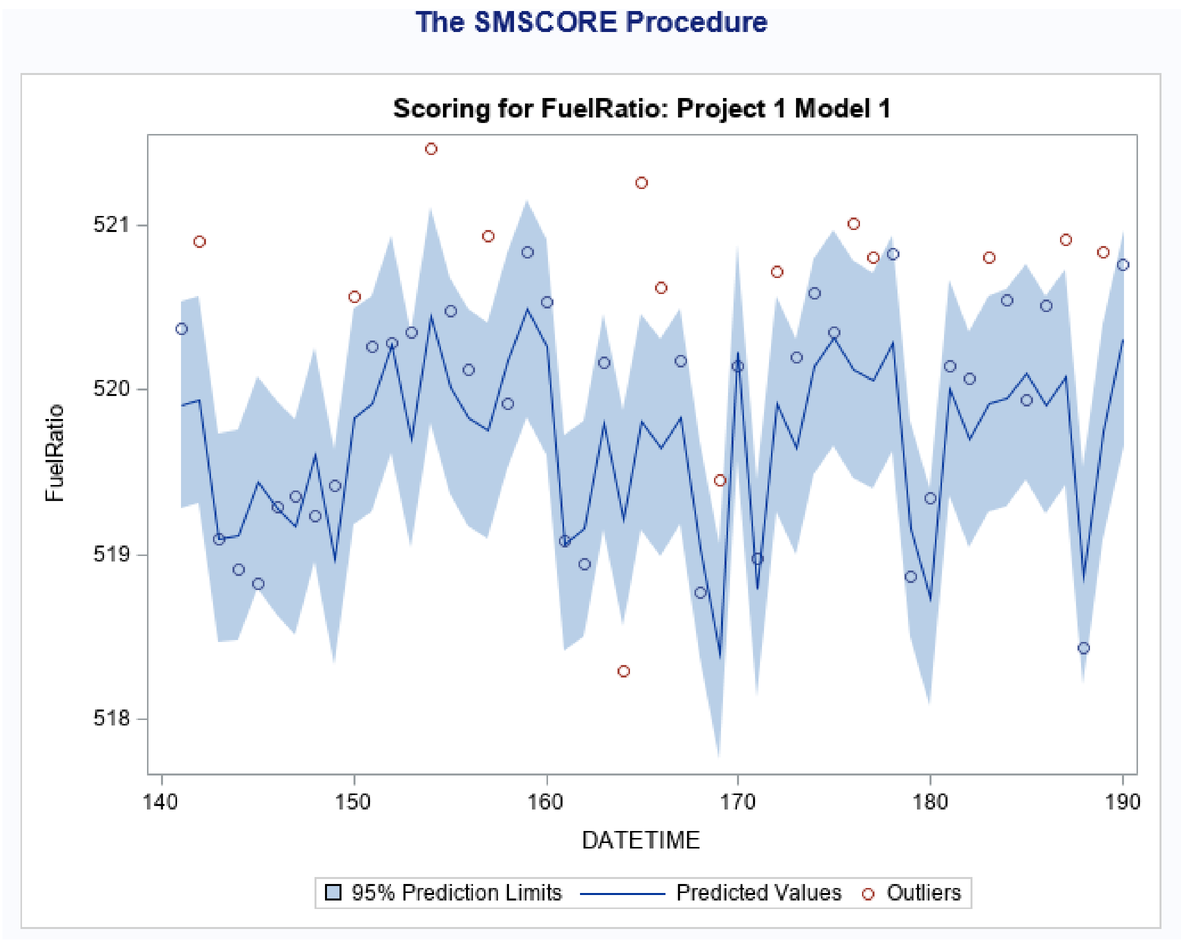
As you can see, the scoring data for the previous window (cycles 91–140) are used as historical data as required by the ARIMA model. The same call of the SMSCORE procedure is used to run scoring for these data:

```
proc smscore     name = mycas.myproj
                 scorerepository=mycas.scoreout
                 models=work.mymodels;

   output out = mycas.score;
run;
```



The scoring results for cycles 141–190 are shown in the following plot.



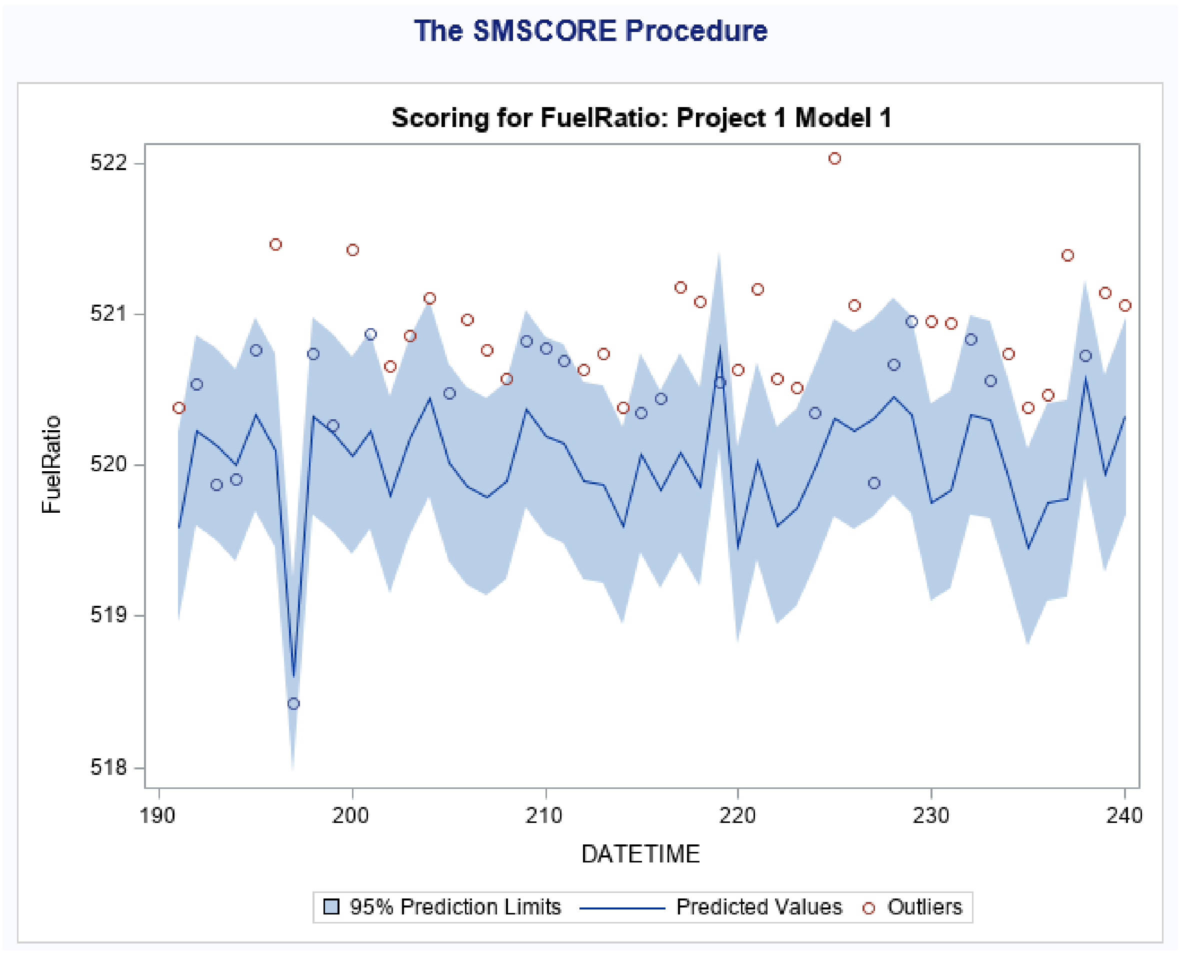
The following statements do the scoring for the next consecutive window, cycles 191–240:

```
data mycas.project_1;
   set train(firstobs=141 obs=190);
run;

data mycas.project_1_score;
   set train(firstobs=191 obs=240);
run;

    proc smscore
       name = mycas.myproj
       [illegible]oreout
       models = work.mymodels;
output out = mycas.score;
run;
```

The scoring results for cycles 191–240 are shown in the following plot.



The three consecutive scoring exercises show that the number of outliers increases from one scoring window to the next and there is a clear trend of increasing numbers of positive outliers (that is, actual values greater than the upper prediction limit). This trend indicates gradual structural changes in the model that manifest engine degradation. This example demonstrates how stability monitoring captures these gradual changes and signals the need for engine maintenance long before a catastrophic failure occurs (in this case, the catastrophic failure happens at cycle 494).

Conclusion

Stability monitoring procedures can be used to monitor the performance of a machine on the basis of sensor and event data in order to detect early signs of degradation before a catastrophic failure occurs. This approach can reduce the cost of maintenance and downtime associated with failures and repairs.

Resources

Saxena, A., and Goebel, K. (2008). “Turbofan Engine Degradation Simulation Data Set.” Accessed January 17, 2017. NASA Ames Prognostics Data Repository. NASA Ames Research Center, Moffett Field, CA. https://catalog.data.gov/dataset/turbofan-engine-degradationsimulation-data-set.

THE POWER TO KNOW.

To contact your local SAS office, please visit: sas.com/offices