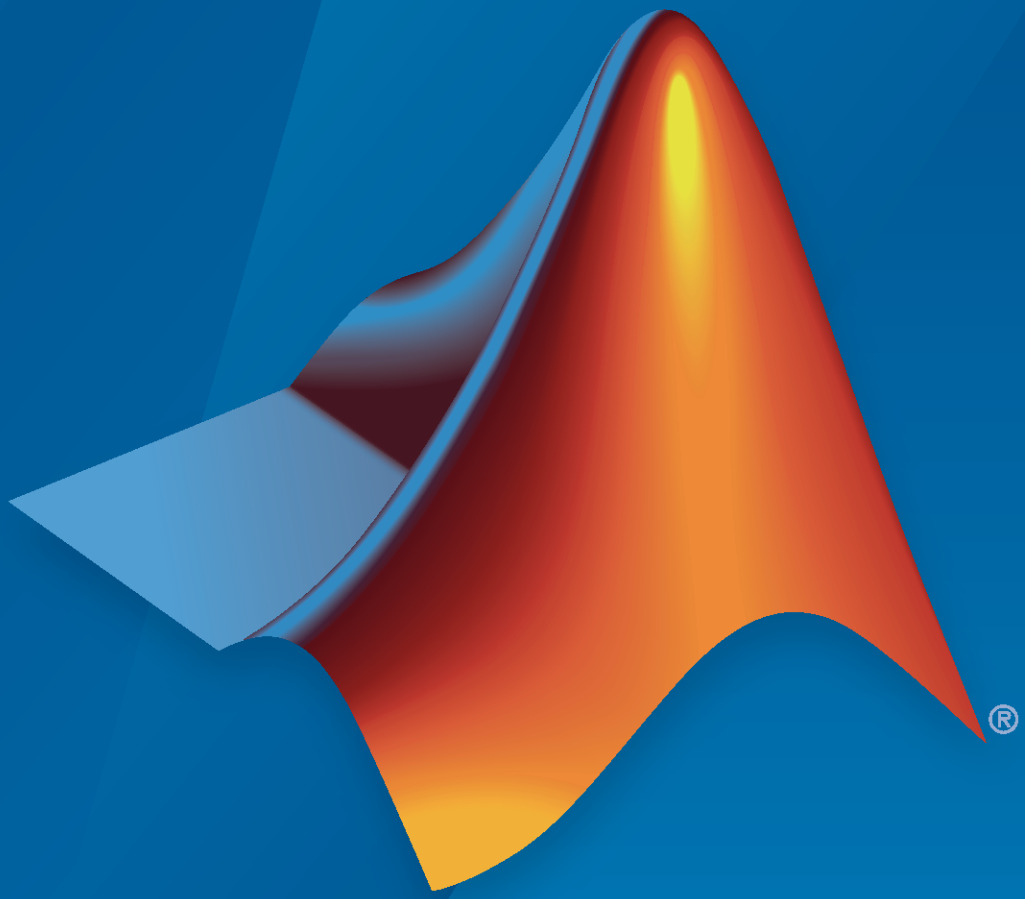
Predictive Maintenance Toolbox™

Getting Started Guide



MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Predictive Maintenance Toolbox™ Getting Started Guide

Revision History

March 2018	Online only	New for Version 1.0 (Release 2018a)
September 2018	Online only	Revised for Version 1.1 (Release 2018b)
March 2019	Online only	Revised for Version 2.0 (Release 2019a)
September 2019	Online only	Revised for Version 2.1 (Release 2019b)
March 2020	Online only	Revised for Version 2.2 (Release 2020a)
September 2020	Online only	Revised for Version 2.2.1 (Release 2020b)
March 2021	Online only	Revised for Version 2.3 (Release 2021a)
September 2021	Online only	Revised for Version 2.4 (Release 2021b)
March 2022	Online only	Revised for Version 2.5 (Release 2022a)
September 2022	Online only	Revised for Version 2.6 (Release 2022b)
March 2023	Online only	Revised for Version 2.7 (Release 2023a)

Contents

1 Introduction to Predictive Maintenance Toolbox

2 Getting Started with Diagnostic Feature Designer

Introduction to Predictive Maintenance Toolbox

Predictive Maintenance Toolbox Product Description

Design and test condition monitoring and predictive maintenance algorithms

Predictive Maintenance Toolbox lets you manage sensor data, design condition indicators, and estimate the remaining useful life (RUL) of a machine.

The toolbox provides functions and an interactive app for exploring, extracting, and ranking features using data-based and model-based techniques, including statistical, spectral, and time-series analysis. You can monitor the health of batteries, motors, gearboxes, and other machines by extracting features from sensor data. To estimate a machine's time to failure, you can use survival, similarity, and trend-based models to predict the RUL.

You can organize and analyze sensor data imported from local files, cloud storage, and distributed file systems. You can label simulated failure data generated from Simulink® models. The toolbox includes reference examples for motors, gearboxes, batteries, pumps, bearings, and other machines that can be reused for developing custom predictive maintenance and condition monitoring algorithms.

To operationalize your algorithms, you can generate C/C++ code for deployment to the edge or create a production application for deployment to the cloud.

Designing Algorithms for Condition Monitoring and Predictive Maintenance

Predictive maintenance allows equipment users and manufacturers to assess the working condition of machinery, diagnose faults, or estimate when the next equipment failure is likely to occur. When you can diagnose or predict failures, you can plan maintenance in advance, better manage inventory, reduce downtime, and increase operational efficiency.

Developing a predictive maintenance program requires a well-designed strategy to assess the working condition of the machinery and detect incipient faults in a timely manner. Doing so requires effective use of both available sensor measurements and your knowledge of the system. You must take many factors into consideration, including:

- The observed sources of faults and their relative frequency. Such sources can be the core components of the machine (such as impeller blades and flow valves in a pump), its actuators (such as an electric motor), or its various sensors (such as accelerometers and flow meters).
- Availability of process measurements through sensors. The number, type and location of sensors, and their reliability and redundancies all affect both algorithm development and cost.
- How various sources of faults translate to observed symptoms. Such cause-effect analysis can require extensive processing of data from the available sensors.
- Physical knowledge about the system dynamics. This knowledge might come from mathematical modeling of the system and its faults and from the insights of domain experts. Understanding system dynamics involves detailed knowledge of relationships among various signals from the machinery (such as input-output relationships among the actuators and sensors), the machine operating range, and the nature of the measurements (for example, periodic, constant or stochastic).
- The ultimate maintenance goal, such as fault recovery or development of a maintenance schedule.

Algorithms for Condition Monitoring and Prognostics

A predictive maintenance program uses condition monitoring and prognostics algorithms to analyze data measured from the system in operation.

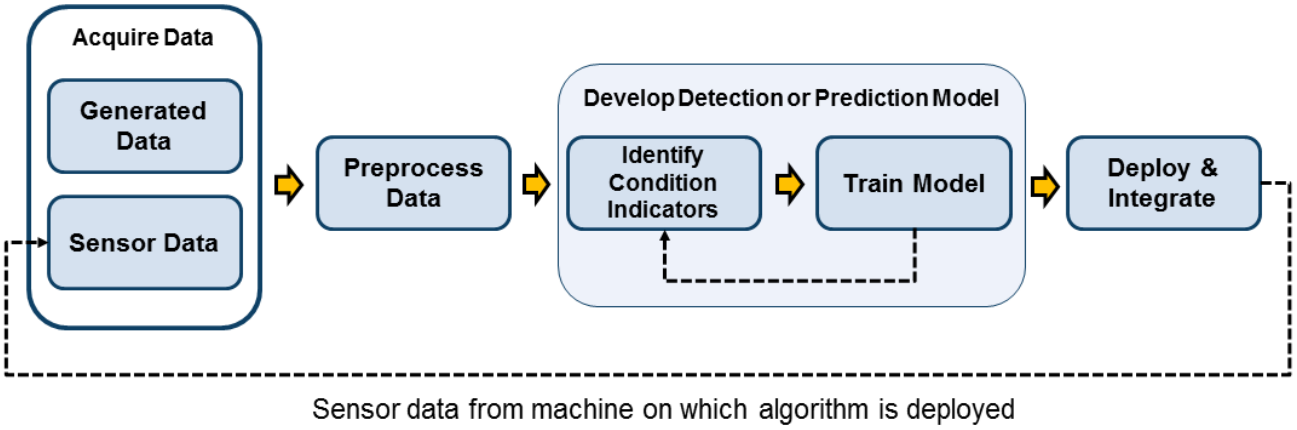
Condition monitoring uses data from a machine to assess its current condition and to detect and diagnose faults in the machine. Machine data is data such as temperature, pressure, voltage, noise, or vibration measurements, collected using dedicated sensors. A condition monitoring algorithm derives metrics from the data called condition indicators. A condition indicator is any feature of system data whose behavior changes in a predictable way as the system degrades. A condition indicator can be any quantity derived from the data that clusters similar system status together, and sets different status apart. Thus a condition-monitoring algorithm can perform fault detection or diagnosis by comparing new data against the established markers of faulty conditions.

Prognostics is forecasting when a failure will happen based on the current and past state of the machine. A prognostics algorithm typically estimates the machine's remaining useful life (RUL) or time-to-failure by analyzing the current state of the machine. Prognostics can use modeling, machine learning, or a combination of both to predict future values of condition indicators. These future values are then used to compute RUL metrics, which determine if and when maintenance should be performed. For the gearbox example, a prognostics algorithm might fit the varying peak vibration frequency and magnitude to a time series to predict their future values. The algorithm can then compare the predicted values to a threshold defining healthy operation of the gearbox, predicting if and when a fault will occur.

A predictive maintenance system implements prognostics and condition monitoring algorithms with other IT infrastructure that makes the end results of the algorithm accessible and actionable to end users who perform the actual maintenance tasks. Predictive Maintenance Toolbox provides tools to help you design such algorithms.

Workflows for Algorithm Development

The following illustration shows a workflow for developing a predictive maintenance algorithm.



Beginning with data that describes your system in a range of healthy and faulty conditions, you develop a detection model (for condition monitoring) or a prediction model (for prognostics). Developing such a model requires identifying appropriate condition indicators and training a model to interpret them. That process is very likely to be iterative, as you try different condition indicators and different models until you find the best model for your application. Finally, you deploy the algorithm and integrate it into your systems for machine monitoring and maintenance.

Acquire Data

Designing predictive maintenance algorithms begins with a body of data. Often you must manage and process large sets of data, including data from multiple sensors and multiple machines running at different times and under different operating conditions. You might have access to one or more of the following types of data:

- Real data from normal system operation
- Real data from system operating in a faulty condition
- Real data from system failures (run-to-failure data)

For instance, you might have sensor data from system operation such as temperature, pressure, and vibration. Such data is typically stored as signal or time series data. You might also have text data, such as data from maintenance records, or data in other forms. This data is stored in files, databases, or distributed file systems such as Hadoop®.

In many cases, failure data from machines is not available, or only a limited number of failure datasets exist because of regular maintenance being performed and the relative rarity of such incidents. In this case, failure data can be generated from a Simulink model representing the system operation under different fault conditions.

Predictive Maintenance Toolbox provides functionality for organizing, labeling, and accessing such data stored on disk. It also provides tools to facilitate the generation of data from Simulink models for predictive maintenance algorithm development. For more information, see “Data Ensembles for Condition Monitoring and Predictive Maintenance”.

Preprocess Data

Data preprocessing is often necessary to convert the data into a form from which condition indicators are easily extracted. Data preprocessing includes simple techniques such as outlier and missing value removal, and advanced signal processing techniques such as short-time Fourier transforms and transformations to the order domain.

Understanding your machine and the kind of data you have can help determine what preprocessing methods to use. For example, if you are filtering noisy vibration data, knowing what frequency range is most likely to display useful features can help you choose preprocessing techniques. Similarly, it might be useful to transform gearbox vibration data to the order domain, which is used for rotating machines when the rotational speed changes over time. However, that same preprocessing would not be useful for vibration data from a car chassis, which is a rigid body.

For more information on preprocessing data for predictive maintenance algorithms, see “Data Preprocessing for Condition Monitoring and Predictive Maintenance”.

Identify Condition Indicators

A key step in predictive maintenance algorithm development is identifying condition indicators, features in your system data whose behavior changes in a predictable way as the system degrades. A condition indicator can be any feature that is useful for distinguishing normal from faulty operation or for predicting remaining useful life. A useful condition indicator clusters similar system status together, and sets different status apart. Examples of condition indicators include quantities derived from:

- Simple analysis, such as the mean value of the data over time
- More complex signal analysis, such as the frequency of the peak magnitude in a signal spectrum, or a statistical moment describing changes in the spectrum over time
- Model-based analysis of the data, such as the maximum eigenvalue of a state space model which has been estimated using the data
- Combination of multiple features into a single effective condition indicator (fusion)

For example, you can monitor the condition of a gearbox using vibration data. Damage to the gearbox results in changes to the frequency and magnitude of the vibrations. The peak frequency and peak magnitude are thus useful condition indicators, providing information about the kind of vibrations present in the gearbox. To monitor the health of the gearbox, you can continuously analyze the vibration data in the frequency domain to extract these condition indicators.

Even when you have real or simulated data representing a range of fault conditions, you might not know how to analyze that data to identify useful condition indicators. The right condition indicators for your application depend on what type of system, system data, and system knowledge you have. Therefore, identifying condition indicators can require some trial and error, and is often iterative with the training step of the algorithm development workflow. Among the techniques commonly used for extracting condition indicators are:

- Order analysis
- Modal analysis
- Spectrum analysis
- Envelope spectrum
- Fatigue analysis
- Nonlinear time-series analysis
- Model-based analysis such as residual computation, state estimation, and parameter estimation

Predictive Maintenance Toolbox supplements functionality in other toolboxes such as Signal Processing Toolbox™ with functions for extracting signal-based or model-based condition indicators from measured or generated data. For more information, see “Identify Condition Indicators”.

Train Detection or Prediction Model

At the heart of the predictive maintenance algorithm is the detection or prediction model. This model analyzes extracted condition indicators to determine the current condition of the system (fault detection and diagnosis) or predict its future condition (remaining useful life prediction).

Fault Detection and Diagnosis

Fault detection and diagnosis relies on using one or more condition indicator values to distinguish between healthy and faulty operation, and between different types of faults. A simple fault-detection model is a threshold value for the condition indicator that is indicative of a fault condition when exceeded. Another model might compare the condition indicator to a statistical distribution of indicator values to determine the likelihood of a particular fault state. A more complex fault-diagnosis approach is to train a classifier that compares the current value of one or more condition indicators to values associated with fault states, and returns the likelihood that one or another fault state is present.

When designing your predictive maintenance algorithm, you might test different fault detection and diagnosis models using different condition indicators. Thus, this step in the design process is likely iterative with the step of extraction condition indicators, as you try different indicators, different combinations of indicators, and different decision models. Statistics and Machine Learning Toolbox™ and other toolboxes include functionality that you can use to train decision models such as classifiers and regression models. For more information, see “Decision Models for Fault Detection and Diagnosis”.

Remaining Useful Life Prediction

Examples of prediction models include:

- A model that fits the time evolution of a condition indicator and predicts how long it will be before the condition indicator crosses some threshold value indicative of a fault condition.
- A model that compares the time evolution of a condition indicator to measured or simulated time series from systems that ran to failure. Such a model can compute the most likely time-to-failure of the current system.

You can predict remaining useful life by forecasting with dynamic system models or state estimators. Additionally, Predictive Maintenance Toolbox includes specialized functionality for RUL prediction based on such techniques as similarity, threshold, and survival analysis. For more information, see “Models for Predicting Remaining Useful Life”.

Deploy and Integrate

When you have identified a working algorithm for handling your new system data, processing it appropriately, and generating a prediction, deploy the algorithm and integrate it into your system. Based the specifics of your system, you can deploy your algorithm on the cloud or on embedded devices.

A cloud implementation can be useful when you are gathering and storing large amounts of data on the cloud. Removing the need to transfer data between the cloud and local machines that are running the prognostics and health monitoring algorithm makes the maintenance process more effective. Results calculated on the cloud can be made available through tweets, email notifications, web apps, and dashboards.

Alternatively, the algorithm can run on embedded devices that are closer to the actual equipment. The main benefits of doing this are that the amount of information sent is reduced as data is transmitted only when needed, and updates and notifications about equipment health are immediately available without any delay. .

A third option is to use a combination of the two. The preprocessing and feature extraction parts of the algorithm can be run on embedded devices, while the predictive model can run on the cloud and generate notifications as needed. In systems such as oil drills and aircraft engines that are run continuously and generate huge amounts of data, storing all the data on board or transmitting it is not always viable because of cellular bandwidth and cost limitations. Using an algorithm that operates on streaming data or on batches of data lets you store and send data only when needed.

MathWorks® code generation and deployment products can help you with this step of the workflow. For more information, see “Deploy Predictive Maintenance Algorithms”.

References

[1] Isermann, R. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Berlin: Springer Verlag, 2006.

See Also

Related Examples

- “Using Simulink to Generate Fault Data”
- “Rolling Element Bearing Fault Diagnosis”
- “Similarity-Based Remaining Useful Life Estimation”
- “Explore Ensemble Data and Compare Features Using Diagnostic Feature Designer”

Getting Started with Diagnostic Feature Designer

Identify Condition Indicators for Predictive Maintenance Algorithm Design

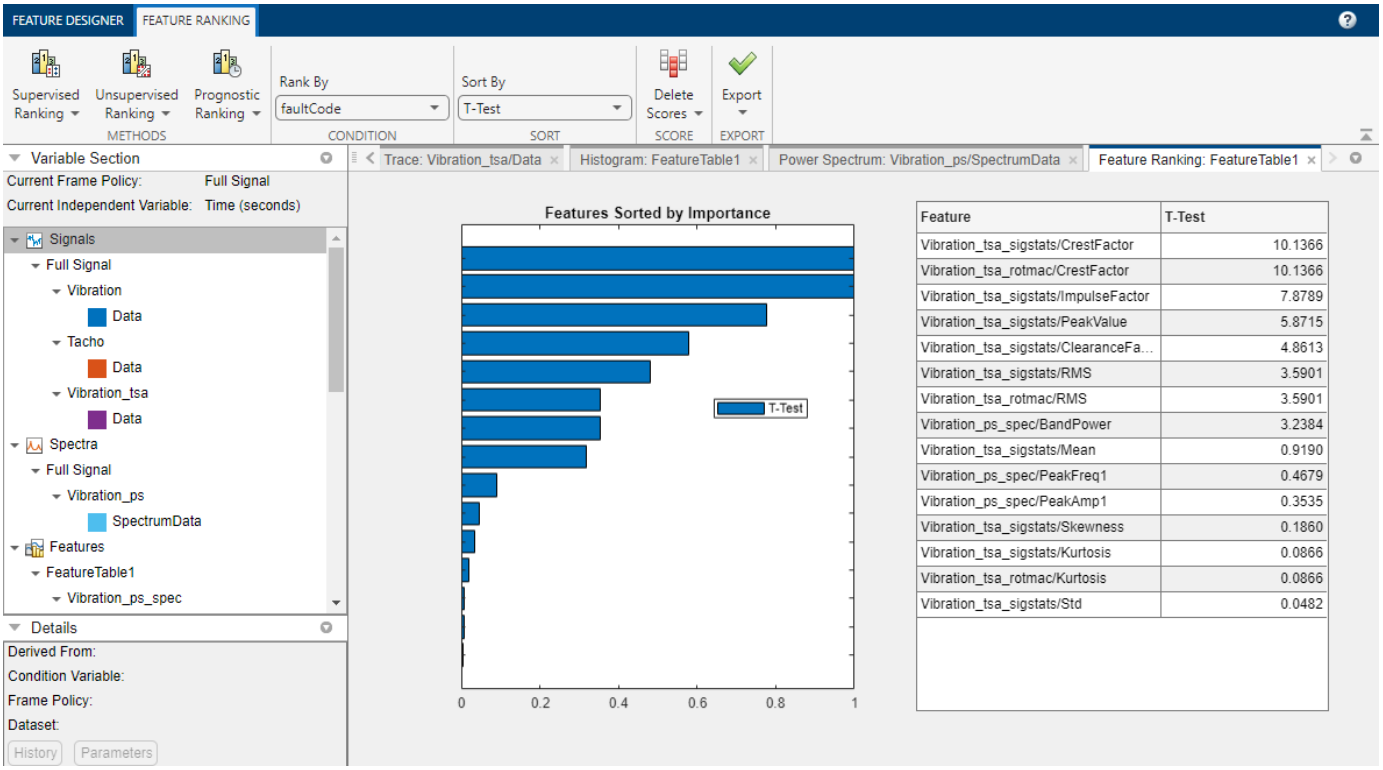
Predictive maintenance allows equipment users and manufacturers to assess the working condition of machinery, diagnose faults, or estimate when the next equipment failure is likely to occur. When you can diagnose or predict failures, you can plan maintenance in advance, better manage inventory, reduce downtime, and increase operational efficiency.

The full workflow for a predictive maintenance program includes multiple steps that begin with data acquisition and end with deployment and integration of a condition monitoring algorithm. For more information, see “Designing Algorithms for Condition Monitoring and Predictive Maintenance” on page 1-3.

A key step in predictive maintenance algorithm development is identifying condition indicators, which are features in your system data whose behavior changes in a predictable way as the system degrades. A condition indicator can be any feature that is useful for distinguishing normal from faulty operation or for predicting remaining useful life. A useful condition indicator clusters similar system statuses together and sets different statuses apart. Examples of condition indicators include quantities derived from:

- Simple analysis, such as the mean value of the data over time
- More complex signal analysis, such as the frequency of the peak magnitude in a signal spectrum or the time-synchronous average of a signal from a rotating source

In the **Diagnostic Feature Designer** app, you can develop features and evaluate potential condition indicators using a multifunction graphical interface.



The app operates on data ensembles. An *ensemble* is a collection of data sets, created by measuring or simulating a system under varying conditions. An individual data set representing one system under one set of conditions is a *member*. **Diagnostic Feature Designer** processes all ensemble members when executing one operation.

Within **Diagnostic Feature Designer**, you can interactively:

- Explore your data ensemble visually by plotting and interacting with your ensemble members together.
- Convert your data into different forms for further exploration. For example, you can create a power spectrum of your signal to evaluate its frequency-domain behavior. Or you can perform time-synchronous averaging, which filters out any noise or disturbance that is not associated with your machine rotation.
- Generate features of various types, and plot histograms that visualize the effectiveness of each feature in separating data from systems with different conditions.
- Rank the generated features by using ranking algorithms that use specific criteria to establish which features are most effective.
- Export the data set or the feature set in the app to your MATLAB® workspace, or export the feature set to **Classification Learner** for model development and additional feature assessment.
- Generate MATLAB code for your features so that you can perform the feature calculations on other or larger data sets.

The following three-part tutorial takes you through the **Diagnostic Feature Designer** workflow for a transmission system model, from your initial data import to the export of your chosen features.

1. “Import and Visualize Ensemble Data in Diagnostic Feature Designer” on page 2-4
2. “Process Data and Explore Features in Diagnostic Feature Designer” on page 2-16
3. “Rank and Export Features in Diagnostic Feature Designer” on page 2-37

Import and Visualize Ensemble Data in Diagnostic Feature Designer

Diagnostic Feature Designer is an app that allows you to develop features and evaluate potential condition indicators using a multifunction graphical interface.

The app operates on data ensembles. An ensemble is a collection of data sets, created by measuring or simulating a system under varying conditions. An individual data set representing one system under one set of conditions is a member. **Diagnostic Feature Designer** processes all ensemble members when executing one operation.

This tutorial shows how to import data into **Diagnostic Feature Designer** and visualize your imported data.

Load Transmission Model Data

This example uses data generated from a transmission system model in “Using Simulink to Generate Fault Data”. Outputs of the model include:

- Vibration measurements from a sensor monitoring casing vibrations
- Data from the tachometer, which issues a pulse every time the shaft completes a rotation
- Fault codes indicating the presence of a modeled fault

Load the data. The data is a table containing variables logged during multiple simulations of the model under varying conditions. Sixteen members have been extracted from the transmission model logs to form an ensemble. Four of these members represent healthy data, and the remaining 12 members exhibit varying levels of sensor drift.

```
load dfd_Tutorial dataTable
```

View this table in your MATLAB command window.

```
dataTable =

  16×3 table

       Vibration              Tacho           faultCode
    ___________________    ___________________    _________

    {6000×1 timetable}    {6000×1 timetable}        0
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        0
    {6000×1 timetable}    {6000×1 timetable}        0
    {6000×1 timetable}    {6000×1 timetable}        0
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
```

```
    {6000×1 timetable}    {6000×1 timetable}        1
    {6000×1 timetable}    {6000×1 timetable}        1
```

The table contains 16 rows, each representing one member. Each column has a variable name. The data variables `Vibration` and `Tacho` are each represented by a `timetable`, and all timetables have the same length. The third variable, `faultCode`, is the condition variable. `faultCode` has a value of `0` for healthy and `1` for degraded.

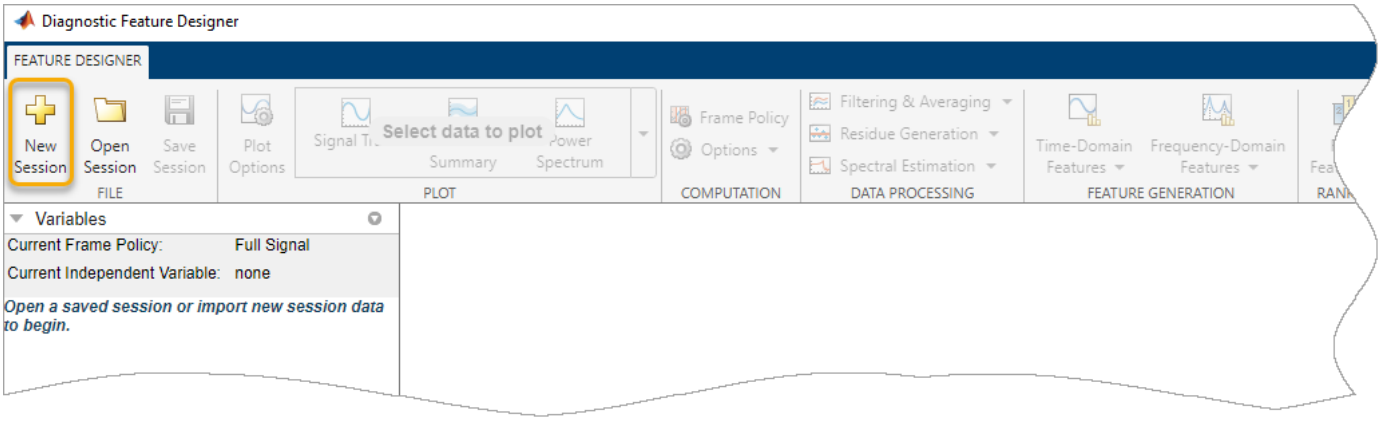
Open Diagnostic Feature Designer

To open **Diagnostic Feature Designer**, enter the following command in your command window.

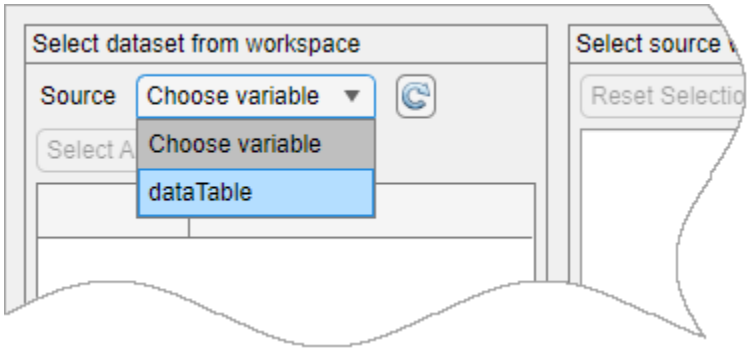
```
diagnosticFeatureDesigner
```

Import Data

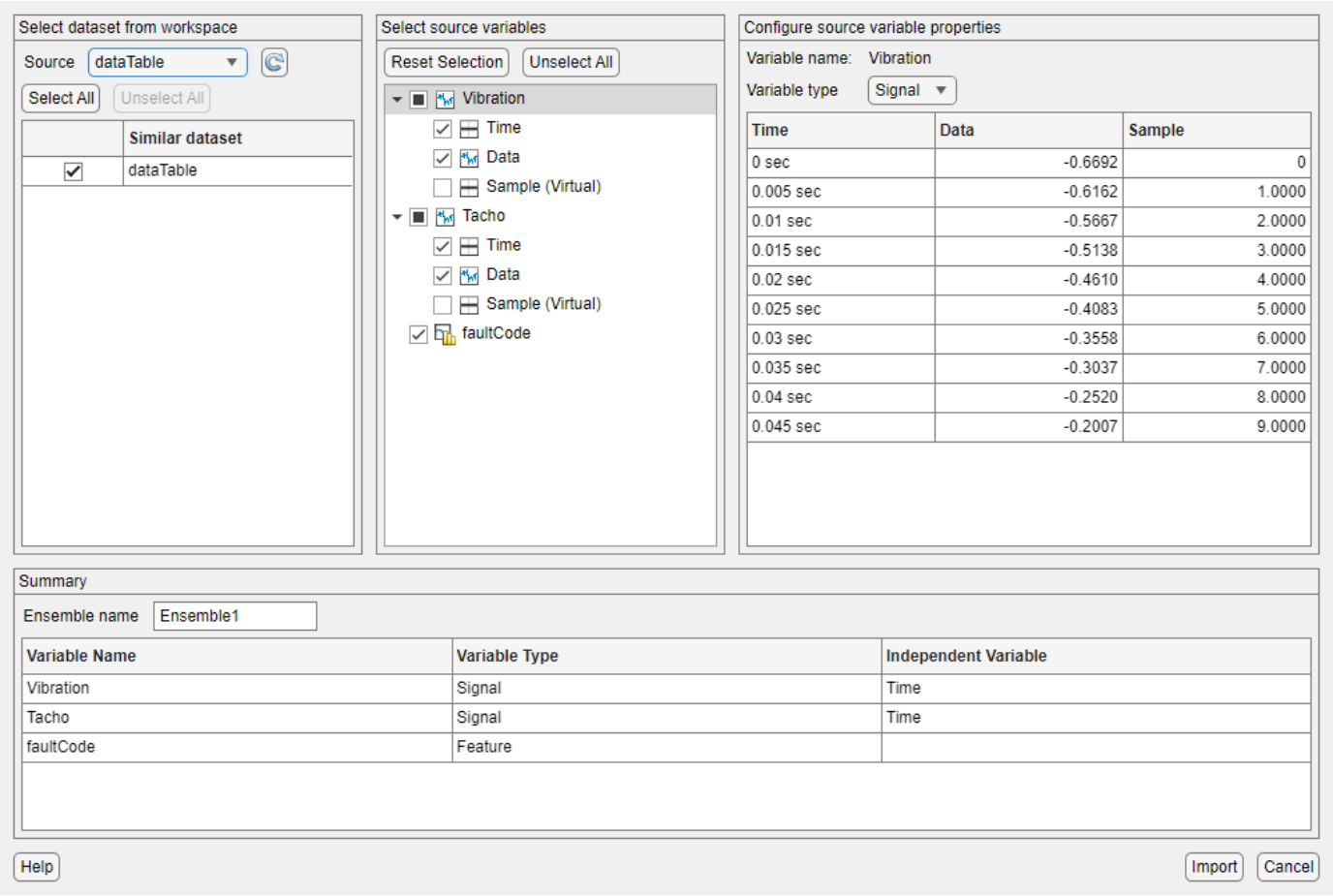
Import the data set that you previously loaded into your MATLAB workspace. To initiate the import process, in the **Feature Designer** tab, click **New Session**.



The **New Session** dialog box opens. From the **Source** list in the **Select dataset from workspace** pane, select `dataTable`.



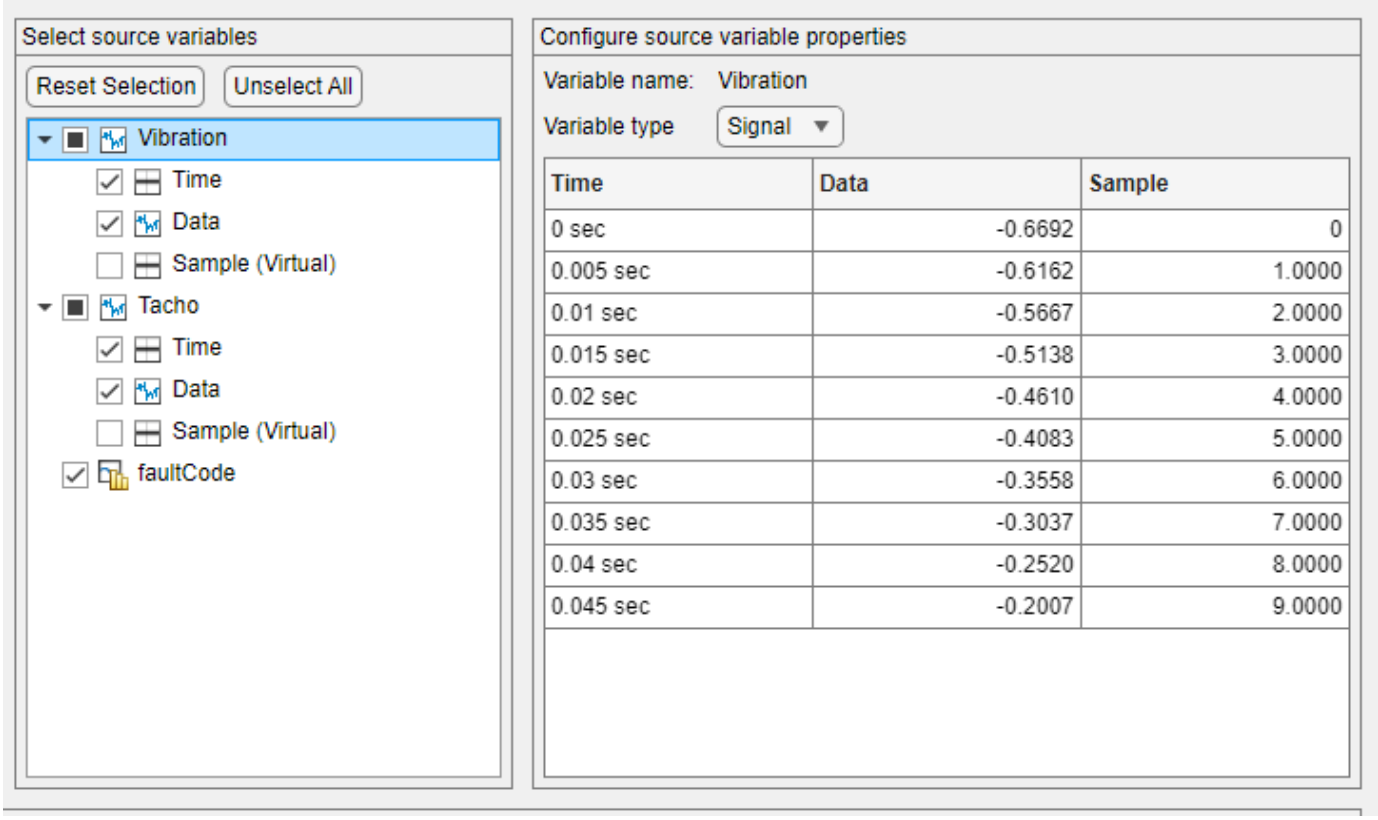
The **Select source variables** pane of the dialog box now displays the variables within `dataTable`. By default, the app initially selects all source variables for import.



The app extracts the variable names from your member tables and embedded timetables. The icon next to the `Vibration` and `Tacho` variable names indicates that the app interprets the variables as time-based signals that each contain `Time` and `Data` variables. You can verify this interpretation in the **Summary** pane at the bottom, which displays the variable name, type, and independent variable for each source-level variable.

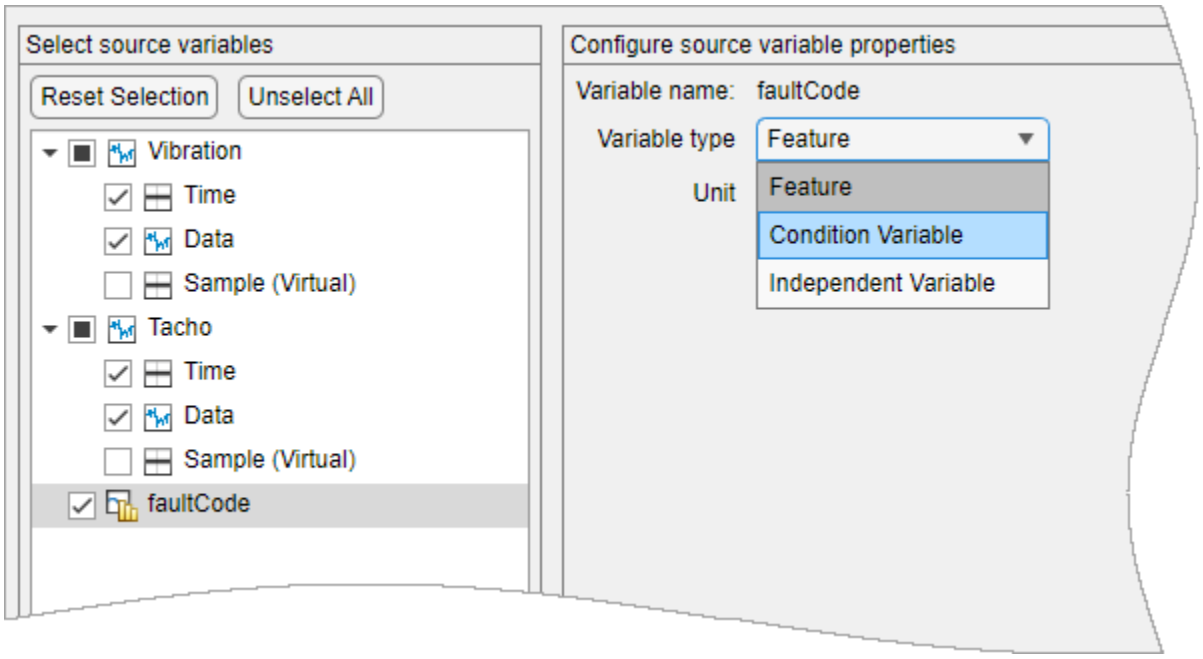
A third variable, `Sample (Virtual)`, also appears in the list, but is not selected and does not appear in **Summary**. The import dialog always includes this variable as an option to allow you to generate virtual independent variables within the app.

View the properties of `Vibration` by selecting the `Vibration` row.

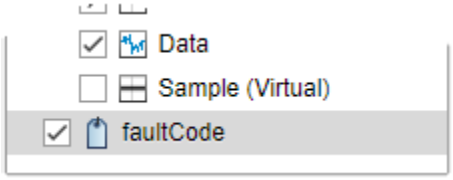


Because the app automatically selects the first variable row for property configuration, the **Configure source variable properties** pane displays the `Vibration` variable name and the **Signal** variable type. For `Vibration`, **Signal** is the only **Variable type** option because the vibration data is packaged in a timetable. Source variable properties also displays a preview of `Vibration` data.

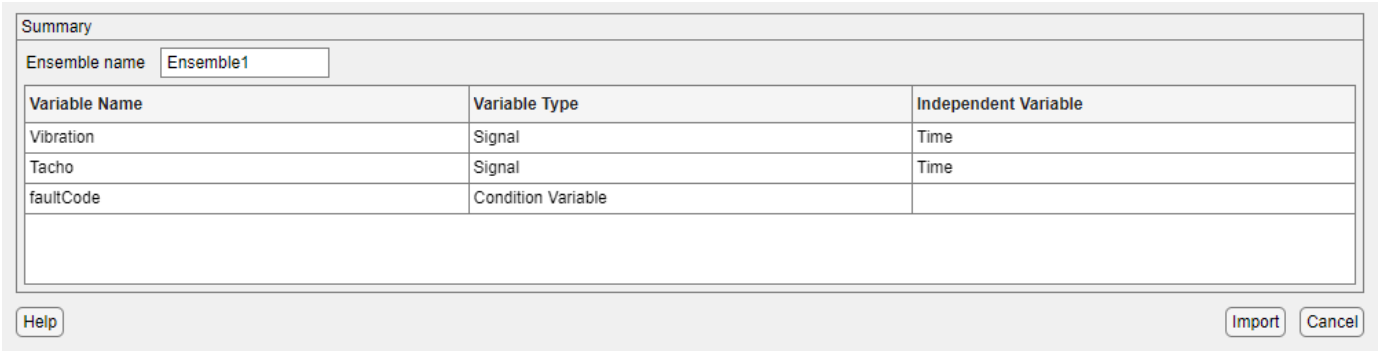
Now examine the variable type of `faultCode`. The icon next to `faultCode`, which illustrates a histogram, represents a feature. Features and condition variables can both be represented by scalars, and the app cannot distinguish between the two unless the condition variable is a categorical. To change the variable type, click on the `faultCode` to open the its properties, and, in **Variable type**, change **Feature** to **Condition Variable**.



The icon for `faultCode` now illustrates a paper tag, which represents a condition variable.

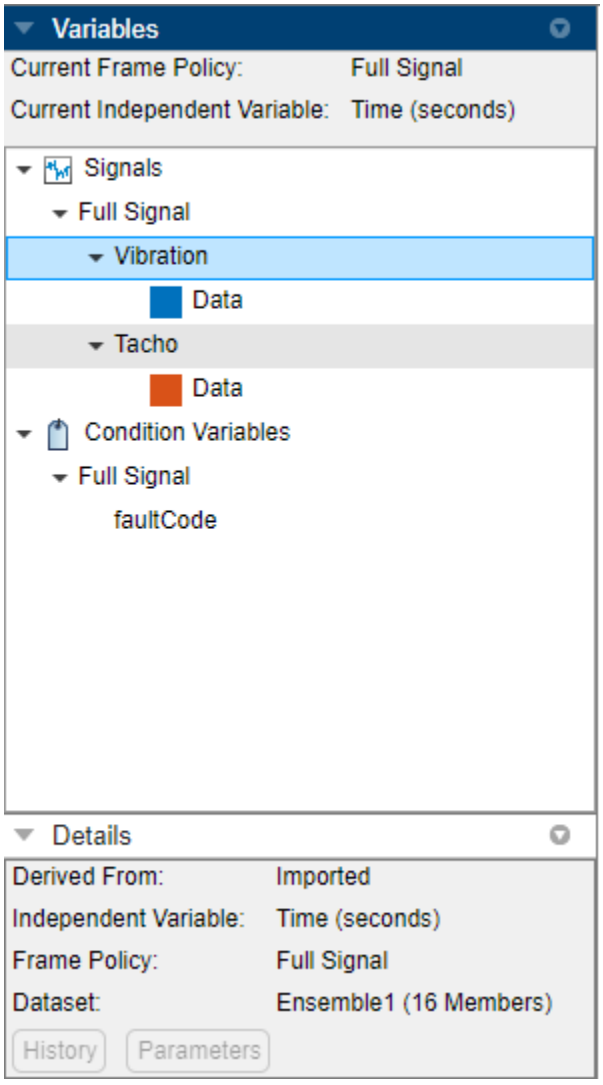


Confirm the ensemble specification in **Summary** and click **Import**.



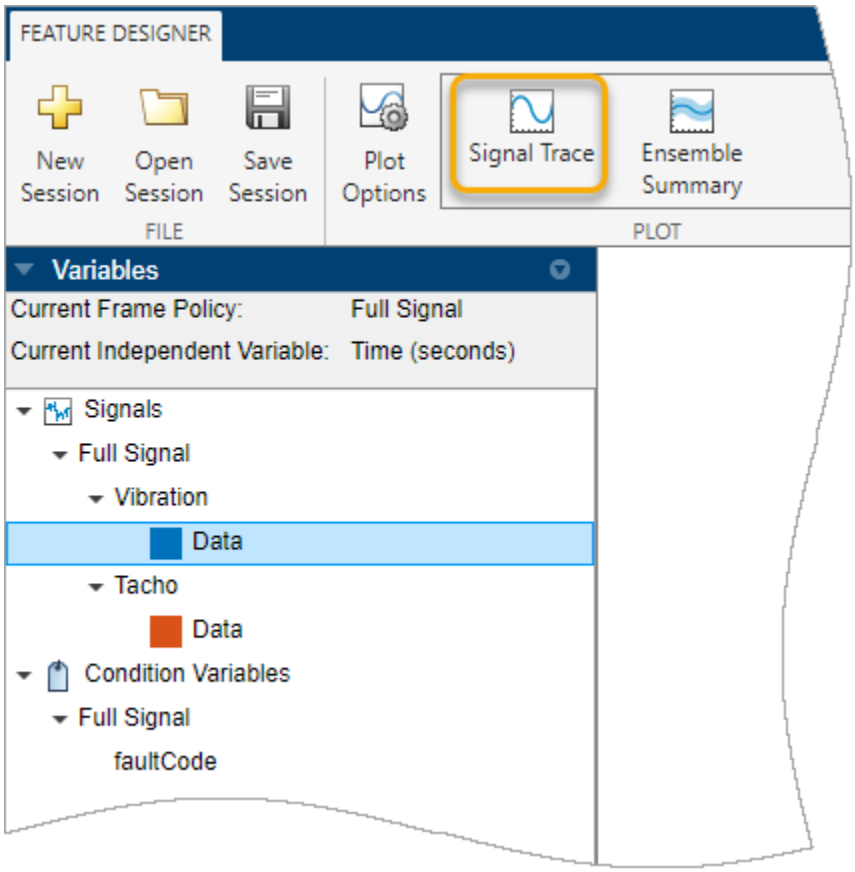
Your imported variables are now in the **Variables** pane, organized into **Signal** and **Condition Variable** types. The color box next to each signal represents that signal in plots. Since the `Vibration` and `Tacho` signals are both timetables that contain the signal data in a column named `Data`, the name `Data` appears below both signal names.

Because the `Vibration` signal is selected, the **Details** pane provides additional information about the signal, including the signal derivation as a direct import and its independent variable (IV). The **Details** pane also shows that it is a full signal rather than a signal that has been processed in frame segments, and that the signal belongs to a dataset that contains 16 members.

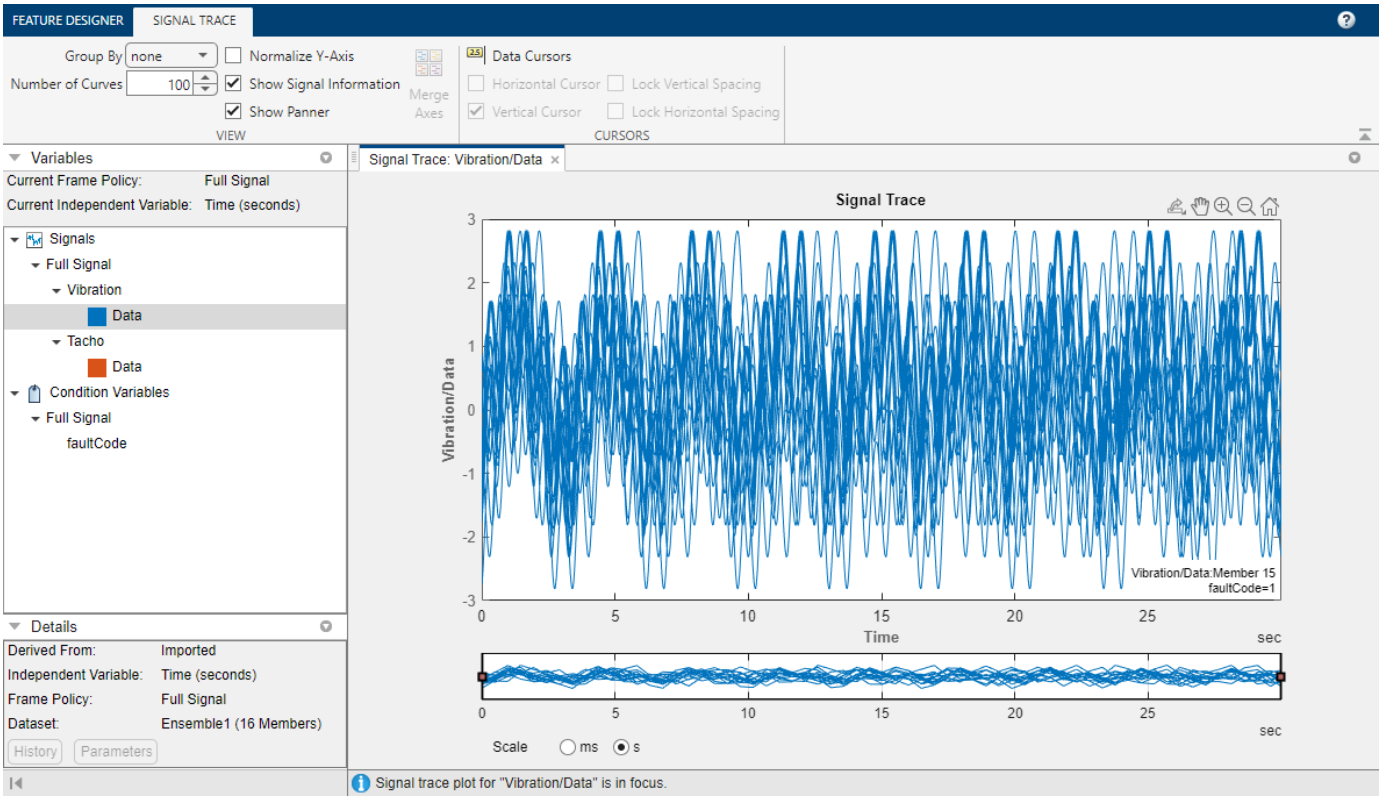


Visualize Data

After you load your signals, plot them and view all your ensemble members together. To view your vibration signal, in the `Vibration` signal in the **Variables** pane, select `Data`. Selecting a signal variable enables the **Signal Trace** option in the plot gallery. Click **Signal Trace**.



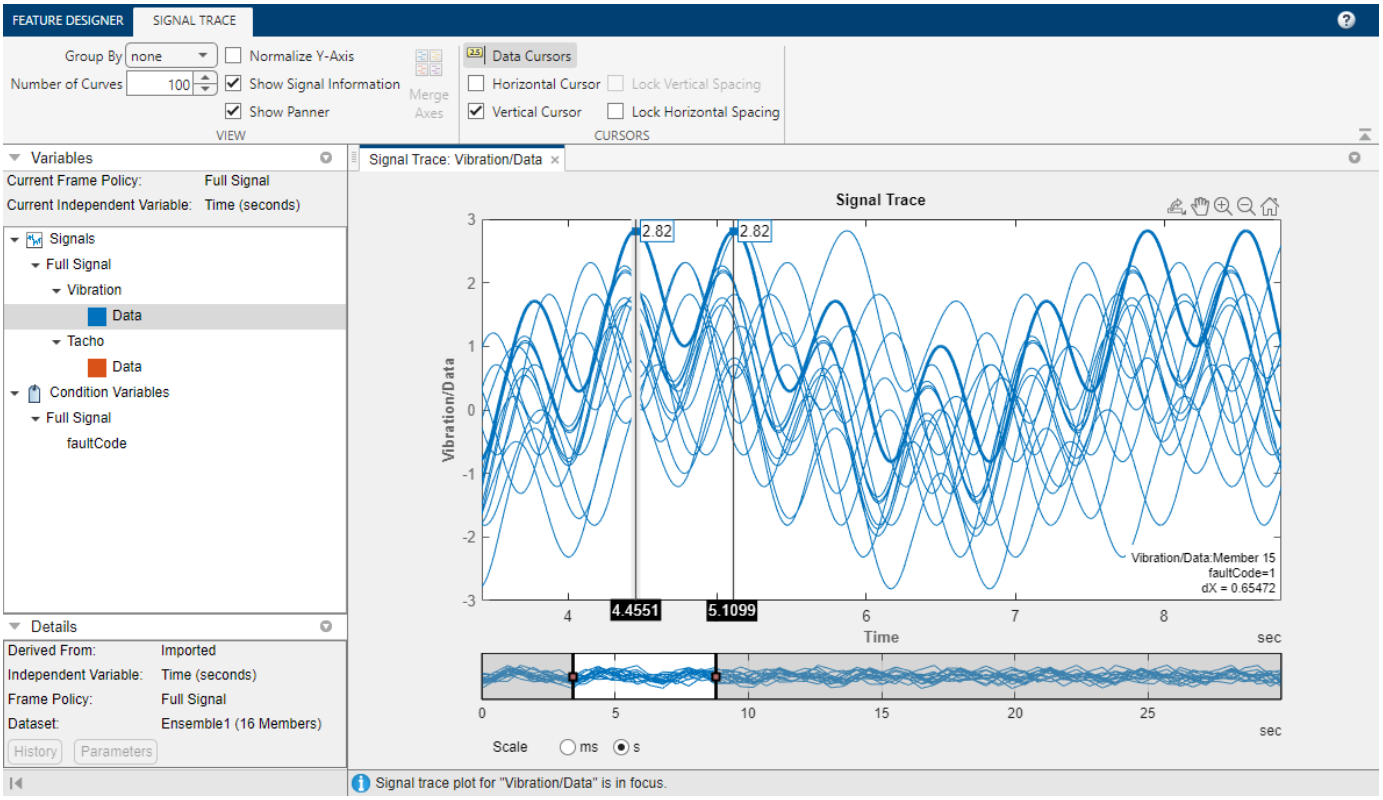
The plotting area displays a signal trace plot of all 16 members. As you move the cursor over the data, an indicator in the lower right corner identifies the member your cursor is on. A second indicator provides the fault code value for that member.



Interact with the trace plot using standard MATLAB plot tools, such as zoom and pan. Access these tools by pointing to the top right edge of the plot. You can also use the specialized options on the **Signal Trace** tab, which appears when you select the **Signal Trace** plot.

Explore Your Data Using Signal Trace Options

Explore the data in your plot using options in the **Signal Trace** tab.

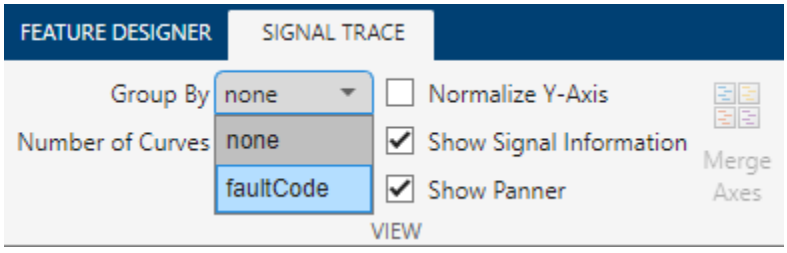


Measure the distance between peaks for the one of the members with high peaks.

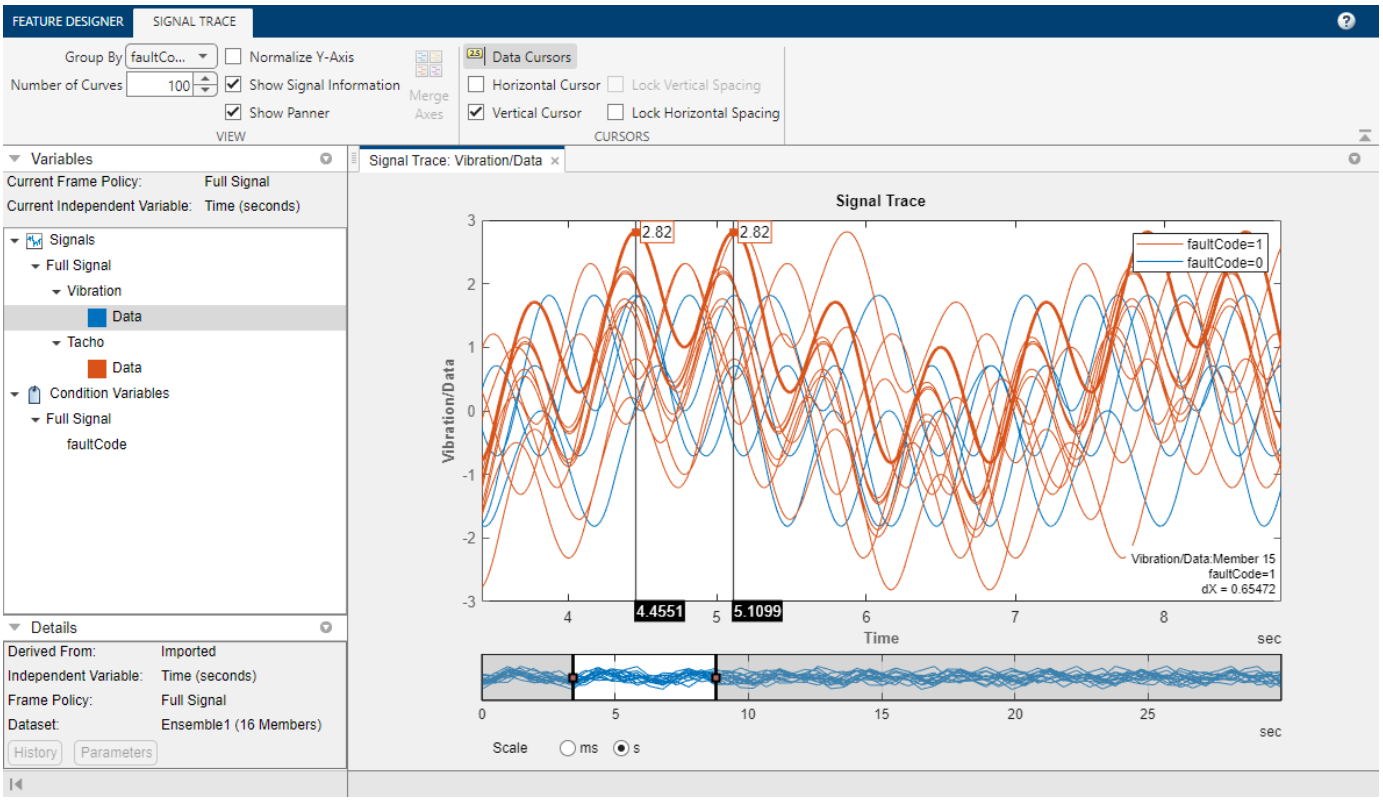
1. Zoom in on the second clusters of peaks. In the panner strip, move the right handle to about 8. Then, move the panner window so that the left handle is at about 4. You now have the second set of peaks within the window.
2. Pause on the first high peak, and note the member number. The second high peak is a continuation of the same member trace.
3. Click **Data Cursors**, and select **Vertical Cursor**. Place the left cursor over the first high peak and the right cursor over the second peak for that member. The lower right corner of the plot displays the separation dX.
4. Select **Lock Horizontal Spacing**. Shift the cursor pair to the right by one peak for the same member. Note that the right cursor is now aligned with the third member peak.

Display Signals with Healthy and Faulty Labels in Different Colors

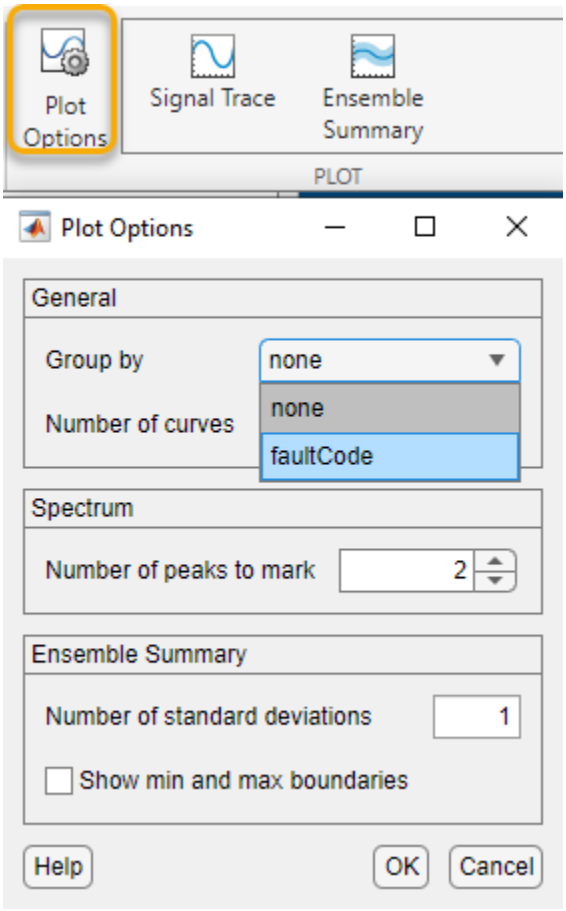
Show which members have matching `faultCode` values by using color coding. In **Group By**, select `faultCode`.



The resulting signal trace shows you that all the highest vibration peaks are associated with data from degraded systems. However, not all the degraded systems have higher peaks.

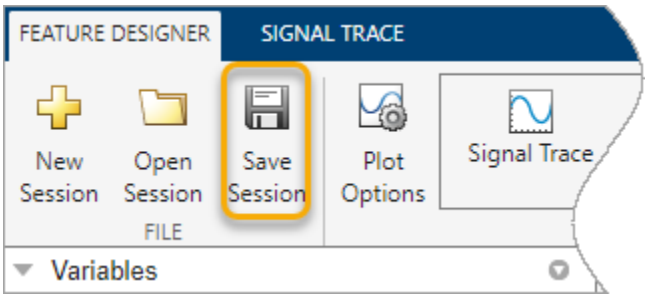


When you use options like **Group By** in the plot tab, your selection is limited to the current plot. To set default plot options for all your plots, in the **Feature Designer** tab, click **Plot Options**. Then, in **Group By**, select `faultCode`.



All the plots that you create in this tutorial now always group by color. You can override the default settings for a specific plot in the plot tab for that plot.

Save your session data. You need this data to run the “Process Data and Explore Features in Diagnostic Feature Designer” on page 2-16 example.



Next Steps

The next step is to explore different ways to characterize your data through features. The example “Process Data and Explore Features in Diagnostic Feature Designer” on page 2-16 guides you through the feature exploration process.

See Also

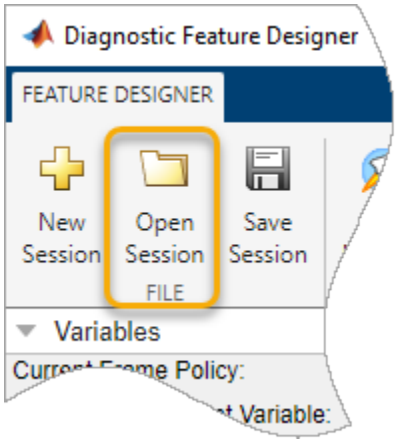
Diagnostic Feature Designer

More About

- “Process Data and Explore Features in Diagnostic Feature Designer” on page 2-16
- “Rank and Export Features in Diagnostic Feature Designer” on page 2-37
- “Import Data into Diagnostic Feature Designer”
- “Explore Ensemble Data and Compare Features Using Diagnostic Feature Designer”
- “Data Ensembles for Condition Monitoring and Predictive Maintenance”
- “Condition Indicators for Monitoring, Fault Detection, and Prediction”

Process Data and Explore Features in Diagnostic Feature Designer

This example shows how to process your data in the app in preparation for feature extraction. If you want to follow along with the steps interactively, use the data you imported in “Import and Visualize Ensemble Data in Diagnostic Feature Designer” on page 2-4. Use **Open Session** to reload your session data using the file name you provided.



If you have no session data, execute the steps for loading and importing data in “Import and Visualize Ensemble Data in Diagnostic Feature Designer” on page 2-4.

A key step in predictive maintenance algorithm development is identifying condition indicators. Condition indicators are features in your system data whose behavior changes in a predictable way as the system degrades. A condition indicator can be any feature that is useful for distinguishing normal from faulty operation or for predicting remaining useful life. A useful feature clusters similar system statuses together and sets different statuses apart.

Diagnostic Feature Designer lets you design features that provide these diagnostics.

- For some features, you can generate features directly using signals you imported.
- For other features, you must perform additional signal processing, such as filtering and averaging, to obtain meaningful results.

The processing you perform depends both on the computational requirements of the feature and the characteristics of your systems and your system data. This example shows how to:

- Process your data in preparation for feature extraction
- Generate features of various types
- Interpret the effectiveness of your features in histograms

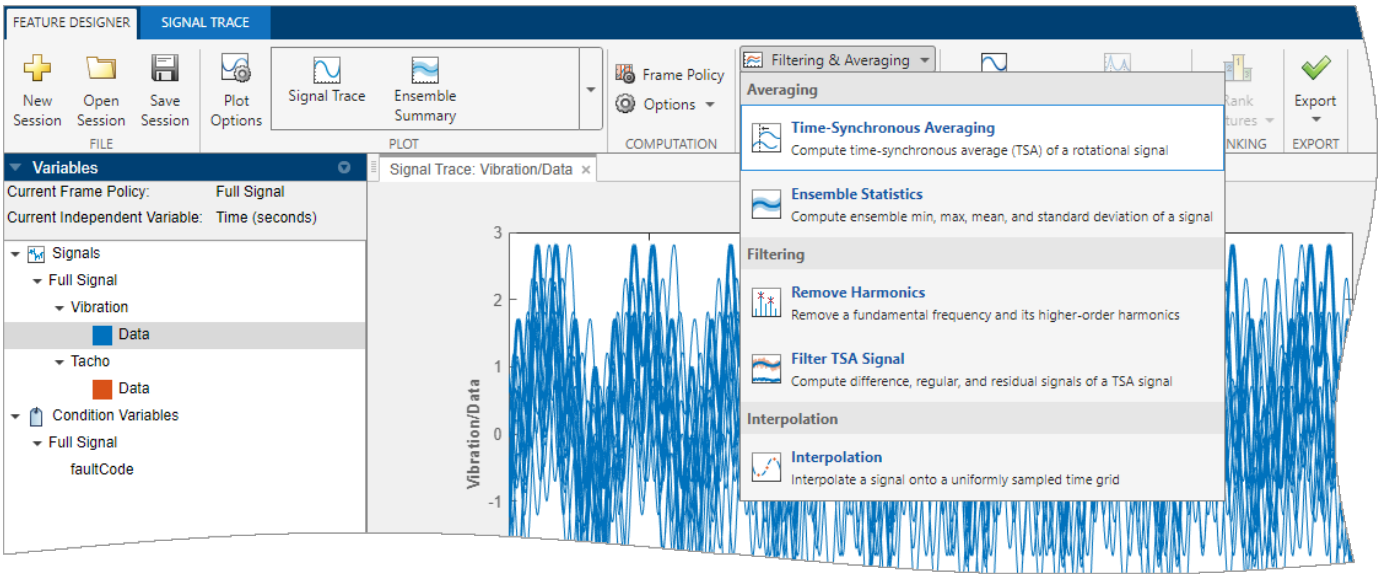
Perform Time-Synchronous Averaging

The data for this system represents a transmission system with rotating parts. The variables include tachometer outputs that precisely mark the completion of each shaft revolution. The data, therefore, is an ideal candidate for time-synchronous averaging.

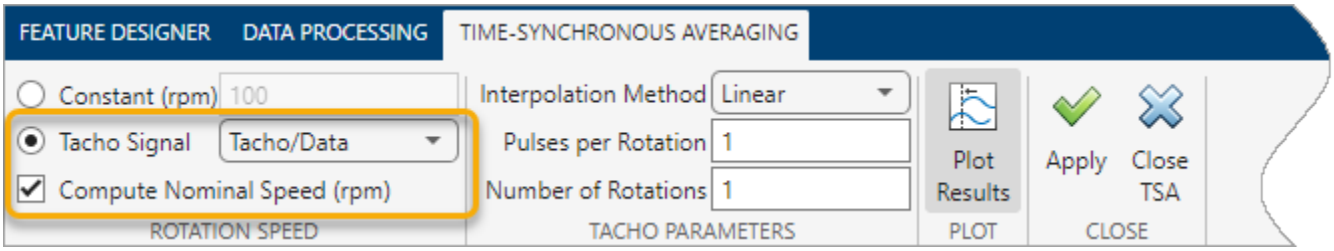
Time-synchronous averaging (TSA) is a common technique for analyzing data from rotating machinery. TSA averages rotation by rotation, and filters out any disturbances or noise that is not coherent with the rotation.

TSA is useful for isolating fault signatures that repeat each rotation, such as perturbations from gear-tooth defects. Features generated from a TSA signal rather than the original vibration signal provide clearer differentiation for rotational fault conditions. This advantage holds true even for features that are not specifically for rotating machinery.

To compute the TSA of the vibration data, first select the signal to average, `Vibration/Data`, in the variables pane. Then, select **Filtering & Averaging > Time-Synchronous Averaging**.

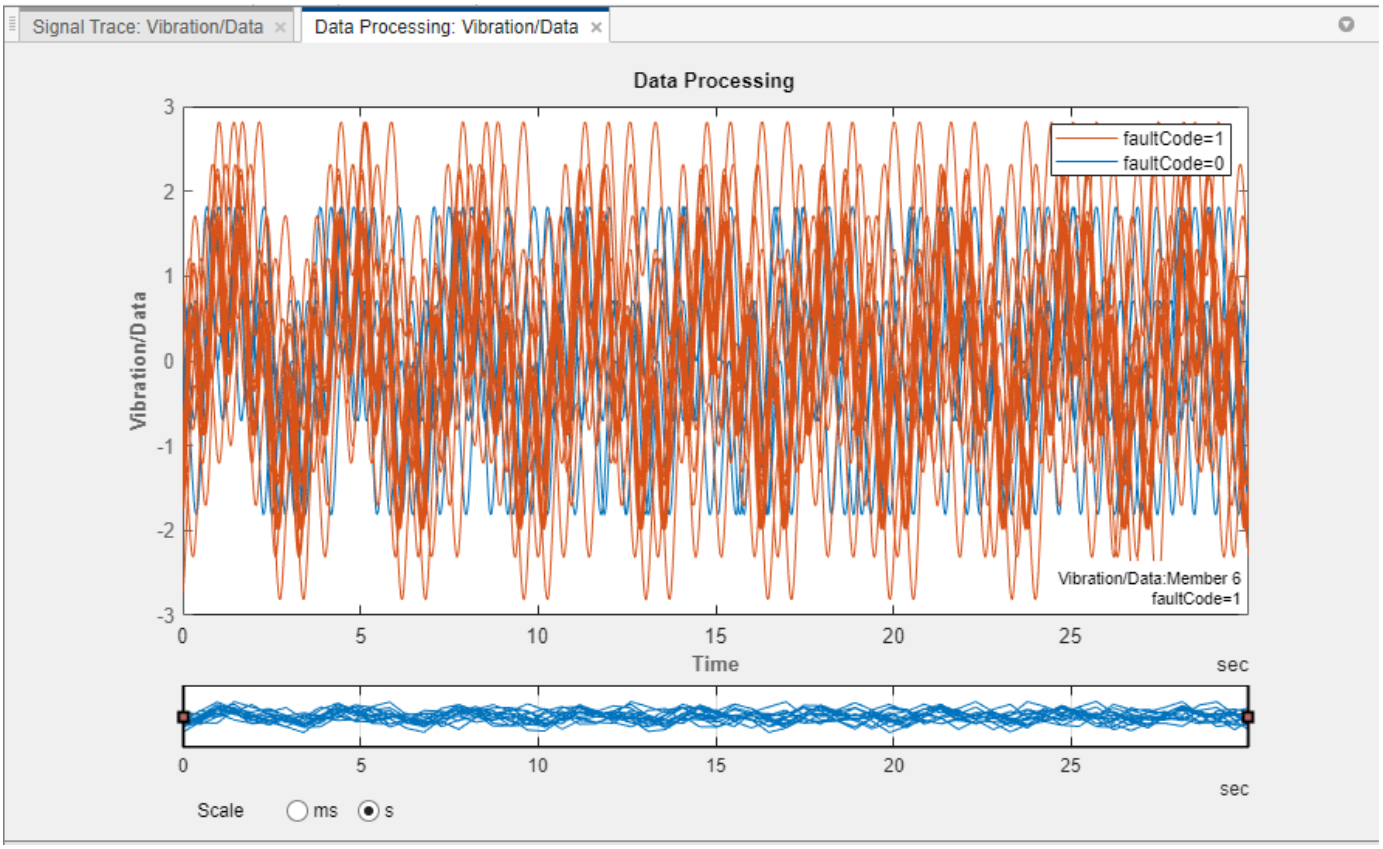


A new **Time-Synchronous Averaging** tab appears.

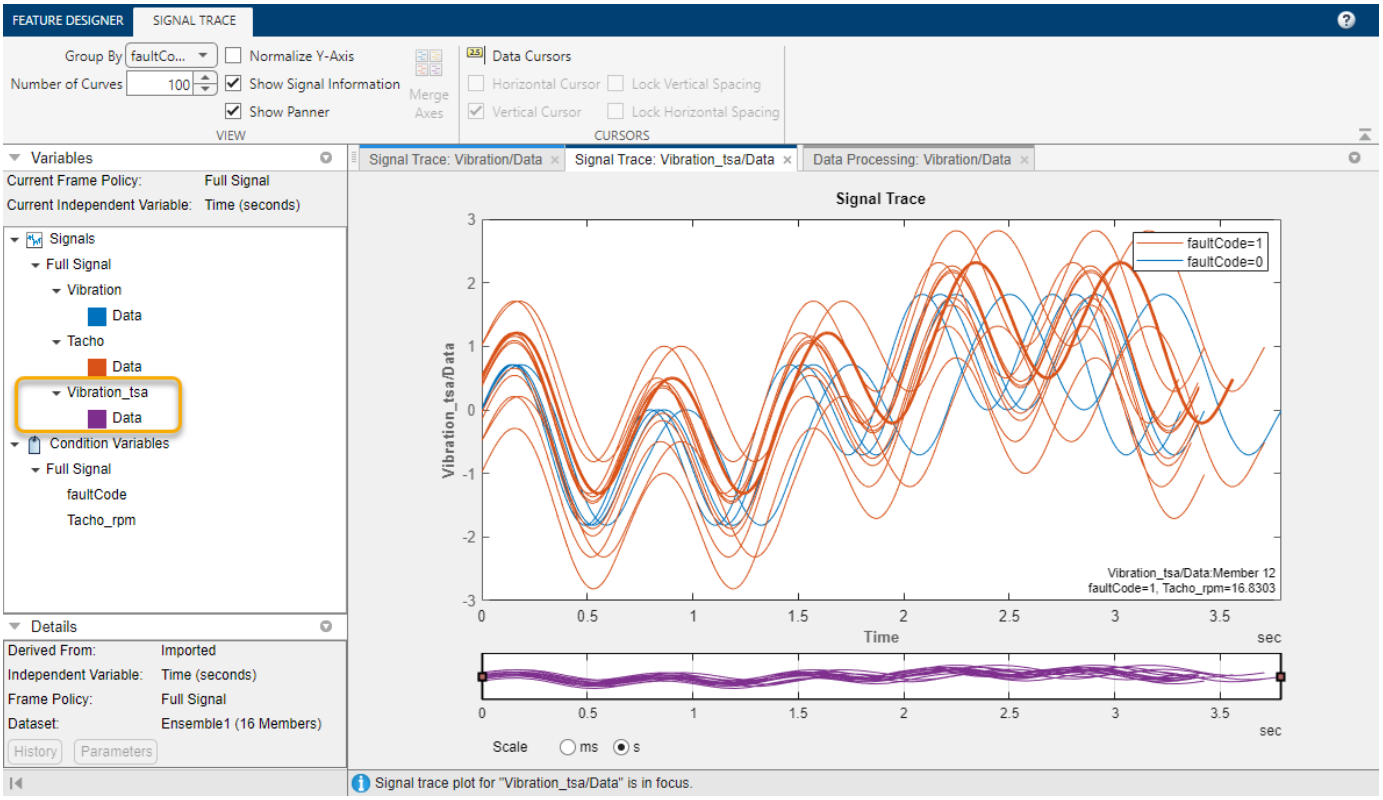


Since you have a tacho signal, select **Tacho signal** and `Tacho/Data`. You can leave **Compute Nominal Speed (rpm)** selected, but this tutorial does not use the nominal rpm information.

Beneath the tab, the plot tab **Data Processing: Vibration/Data** displays the source signal for the TSA signal.



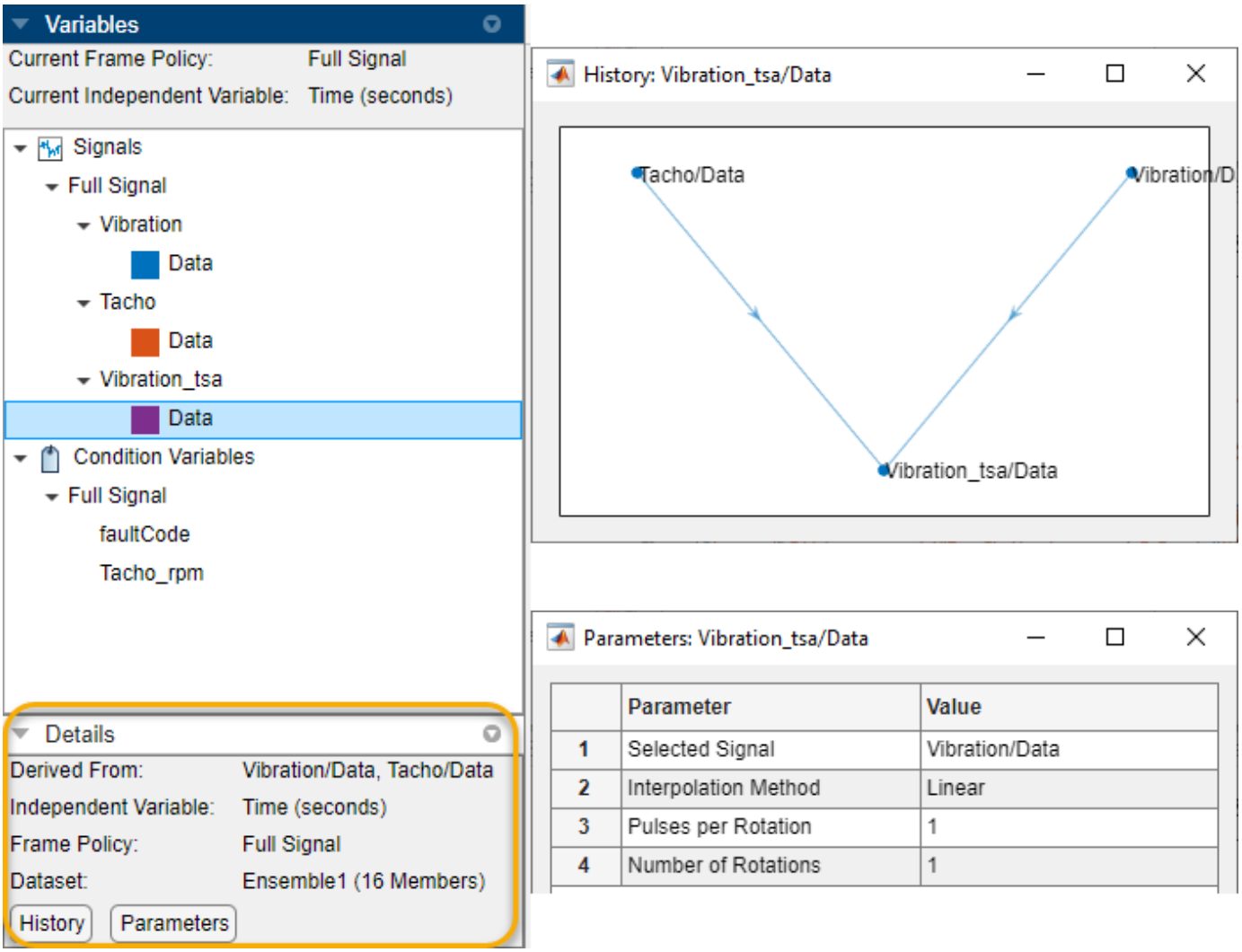
Click **Apply** to start the TSA computation for each of the 16 members of the ensemble. A progress bar shows the status while the computation progresses. When the computation concludes, the app adds a new variable `Vibration_tsa` to the signal list and plots the signal.



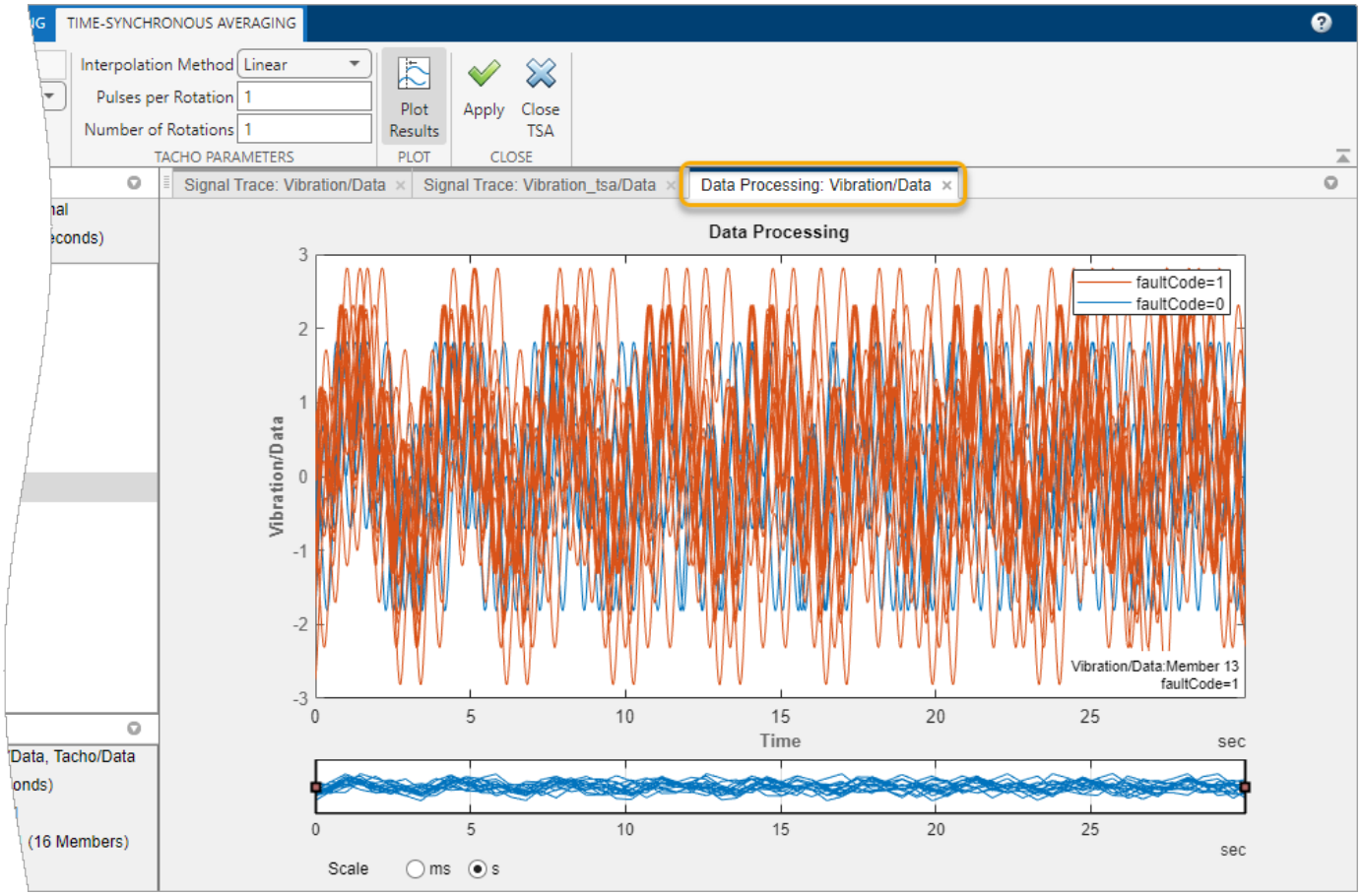
Note that the time axis of the TSA signal plot is less than four seconds long. The original vibration data was 30 seconds long. The shorter timespan reflects the duration of a single rotation for each member.

The member shaft rates for these signals diverge. This divergence is evident in the increasing misalignment of the peaks during the rotation, and the fact that the member traces stop at different times.

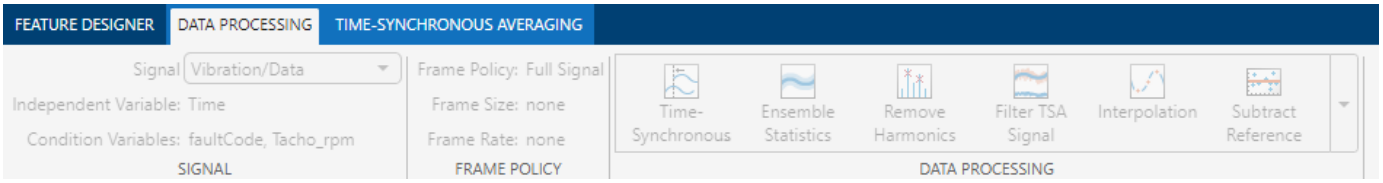
Use the **Details** pane to find out more about the TSA signal. In this pane, you can see that the TSA signal is computed from the vibration and tacho signals. Click **History** to see a plot of the TSA signal processing history. Click **Parameters** to see a list of the processing parameters that you used.



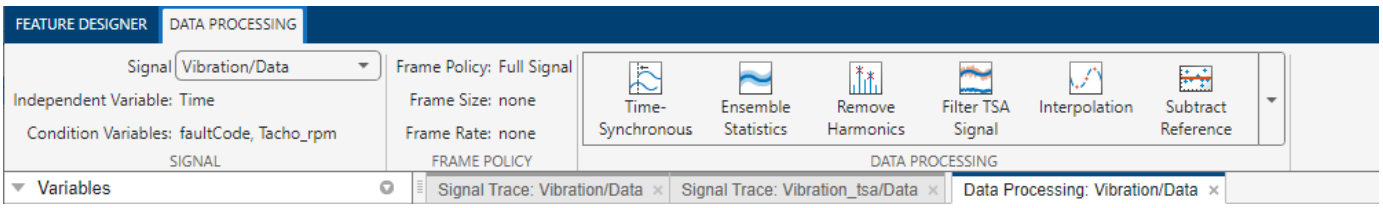
When the TSA computation completes, the **Signal Trace** tab for the TSA signal replaces the **Time-Synchronous Averaging** tab and the **Data Processing** tab. If you want to return to the **Time-Synchronous Averaging** tab, click the plot tab **Data Processing: Vibration/Data**.



The app restores both toolstrip tabs, with the TSA tab active and the data processing tab inactive.

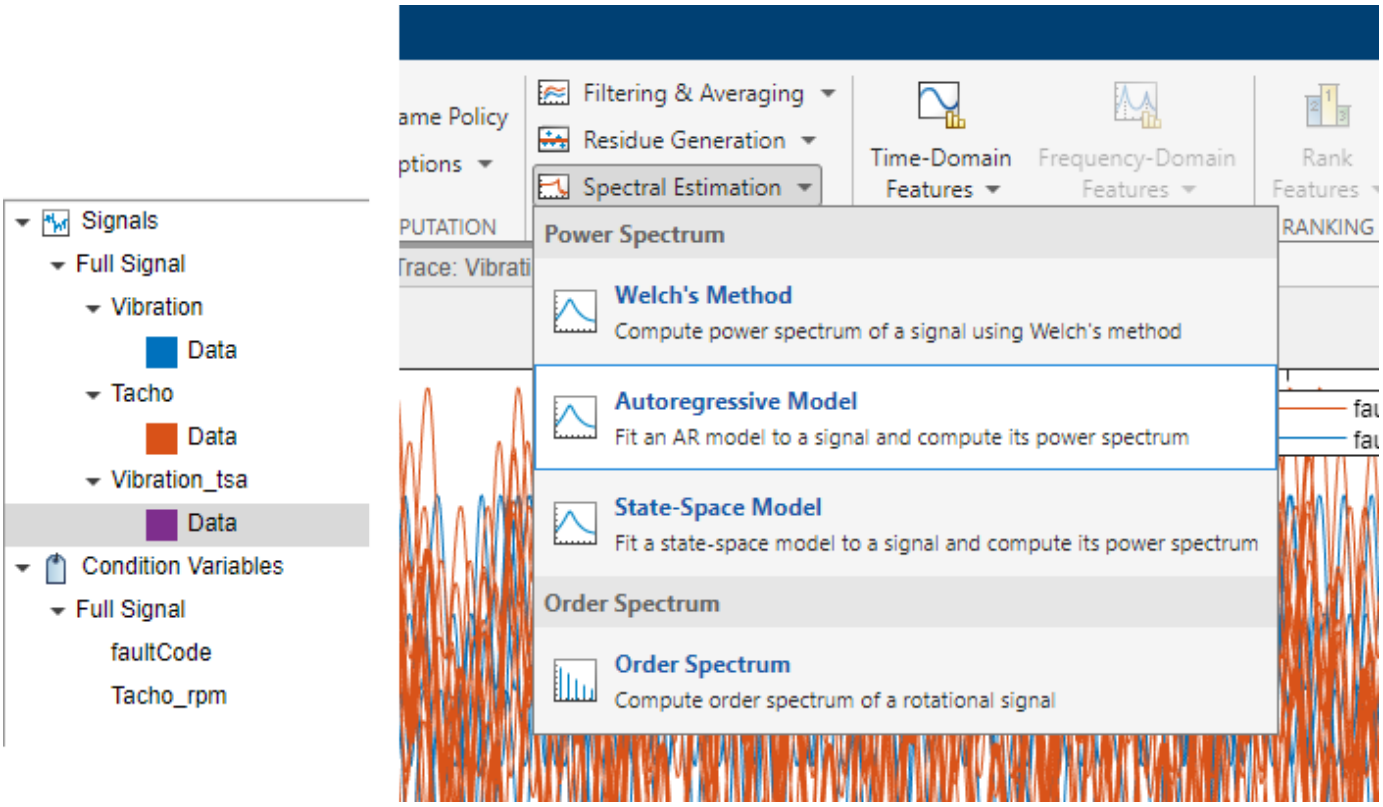


If you want to perform similar processing on another variable, click **Close TSA**. The **Data Processing** tab activates. From that tab, you can change the signal to process. Then, from the data processing gallery, you can select TSA processing or any other processing that is compatible with your signal selection. The processing tab that you select retains any settings that you specified previously in the session.

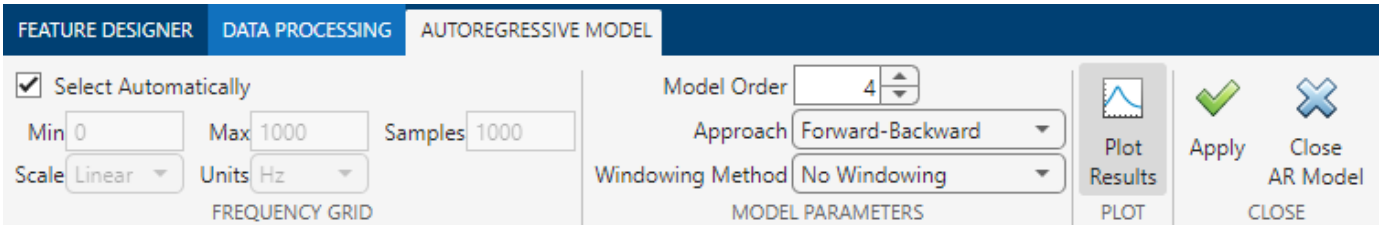


Compute Power Spectrum

The TSA signal gives you enough information to start generating time-domain features, but you must provide a spectrum to explore spectral features. To generate a power spectrum, select the new TSA signal `Vibration_tsa/Data` in the variables pane. Then, click **Spectral Estimation** to bring up the spectrum options. From these options, select `Autoregressive Model`.

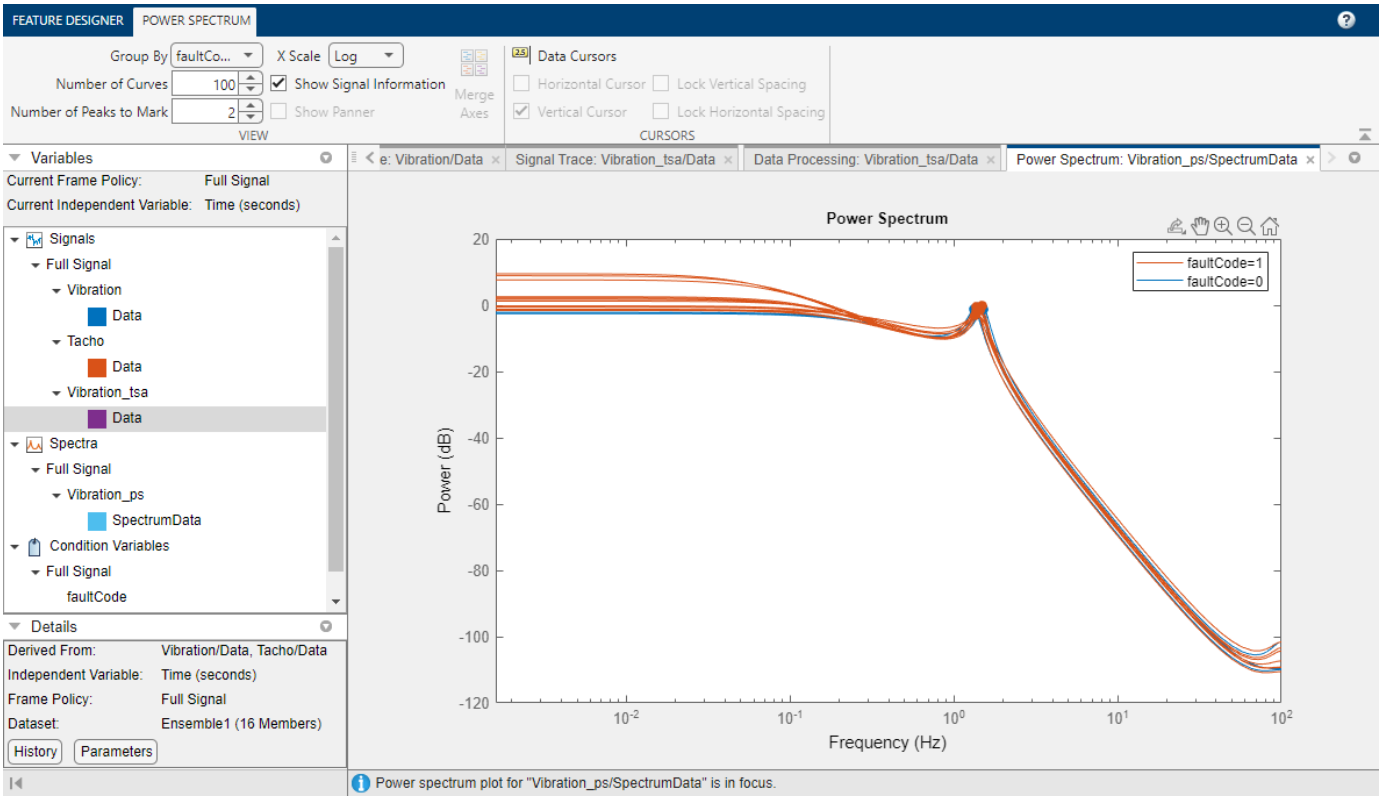


The **Autoregressive Model** tab provides parameters that you can modify. Accept the default values by clicking **Apply**.

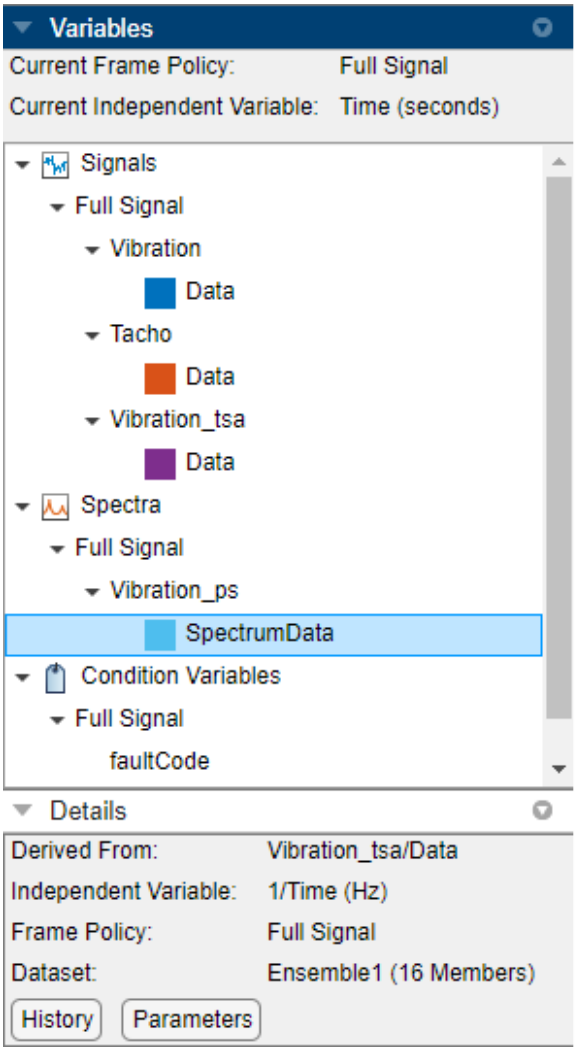


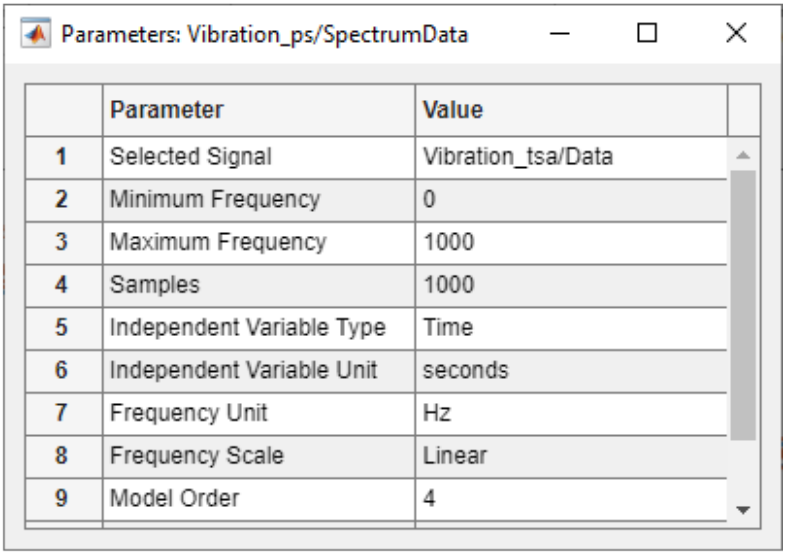
The power spectrum processing results in a new variable, `Vibration_ps/SpectrumData`. The associated icon represents a frequency response.

A plot of the spectrum appears in the plot area. As with **Signal Trace**, a **Power Spectrum** tab provides options for plotting. These options are similar to **Signal Trace**. The plot has no **Panner** option because **Panner** does not work for spectral plots unless the spectra are frame-based (segmented).



Select `Vibration_ps/SpectrumData`. The details pane shows that this signal is derived from the TSA signal. The processing parameters list is more extensive than that of the TSA processing parameters.

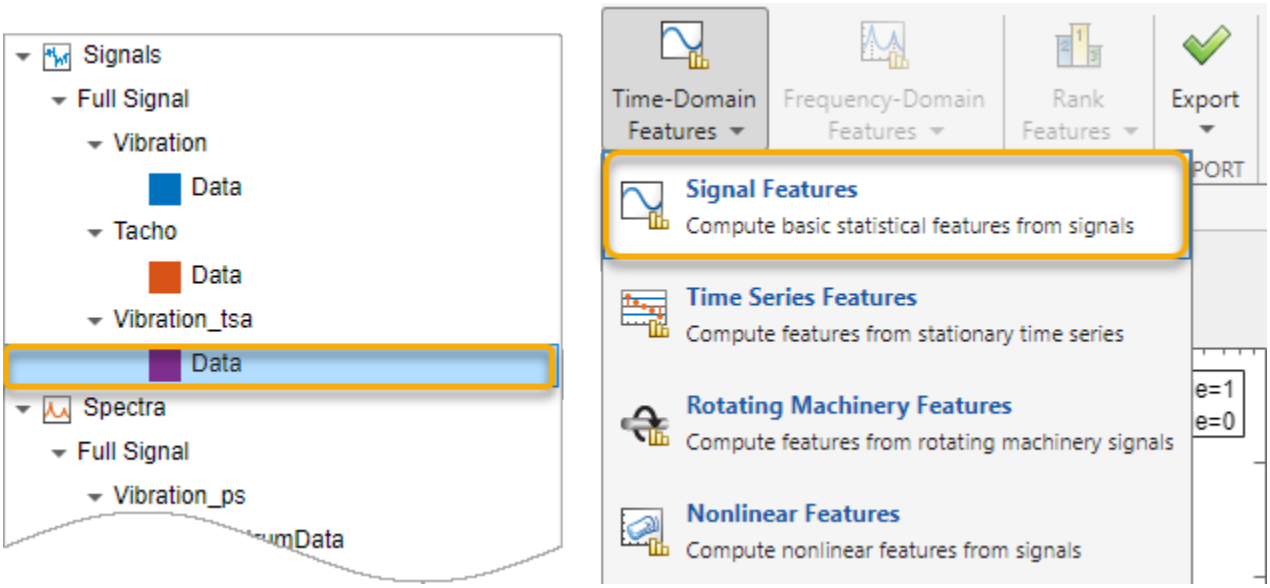




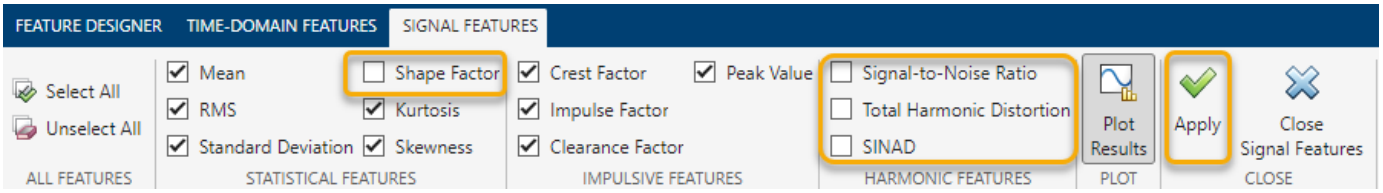
Generate Features

Signal Features

Generate features based on general data statistics, using the TSA signal as your source. Select **Time-Domain Features > Signal Features**.



As with data processing, preselect your source signal before choosing a feature option. Select `Vibration_tsa/Data` and then click **Signal Features** to bring up the **Signal Features** tab. By default, all features are selected. Clear the selections for **Shape Factor** and all options in **Harmonic Features**.

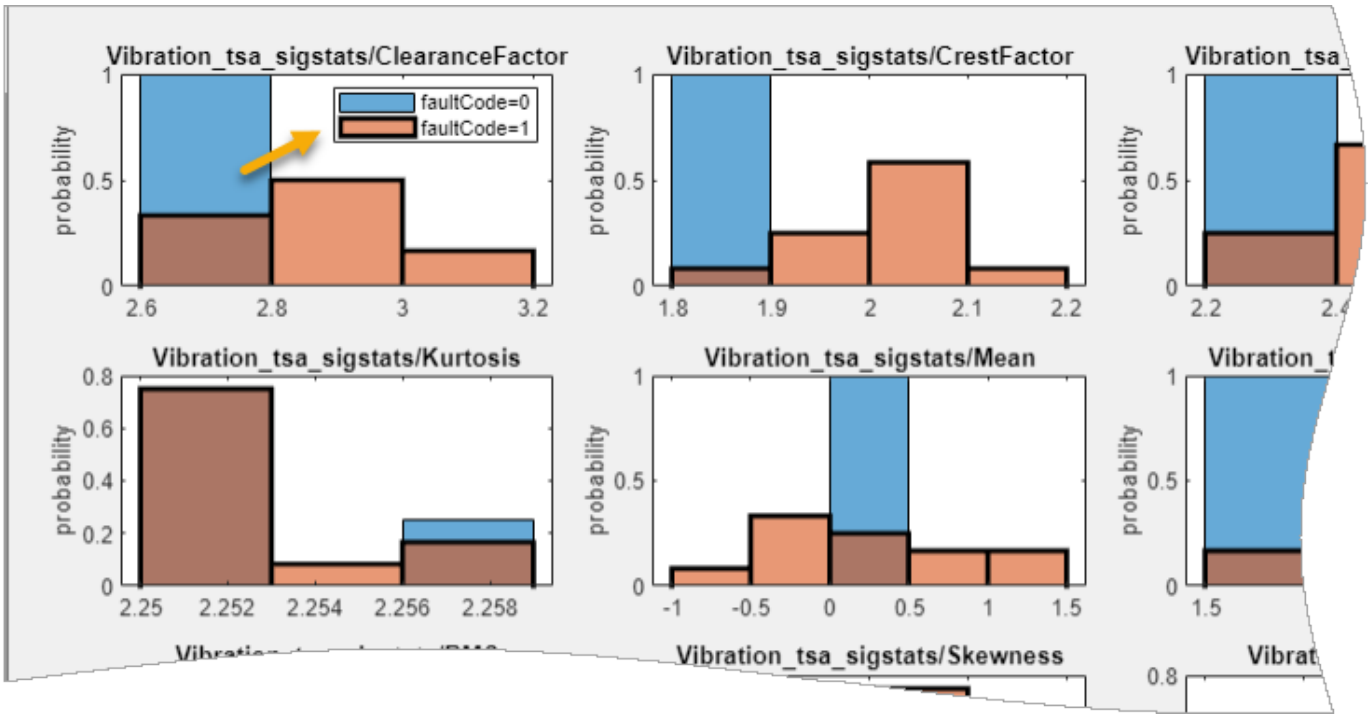


For every selected feature, the app computes a value for each ensemble member and displays the results in a histogram. Each histogram contains bins containing the number of feature values that fall within the bin range. The **Histogram** tab displays parameters that determine the content and resolution of the histograms.

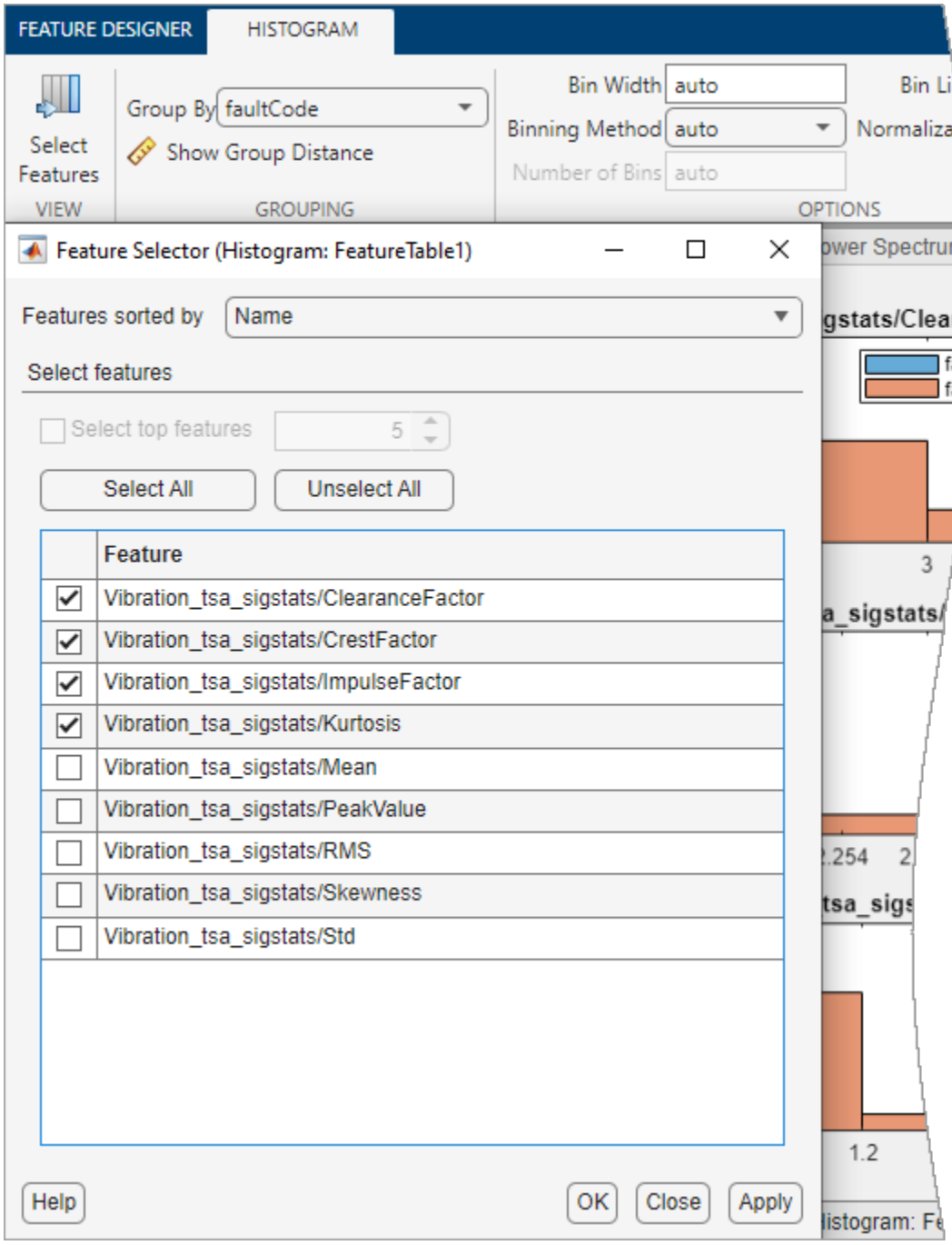
The histogram groups, or color codes, the data according to the condition variable `faultCode` in **Group By**. Blue data is healthy (`faultCode = 0`) and orange data is degraded (`faultCode = 1`), as indicated by the legend (color coding might appear different in your session). For feature values where the healthy and degraded labels overlap, the color appears brown because of the overlap between blue and orange.

You can get a rough idea of which features are effective by assessing which ones clearly segregate blue data from orange data. `CrestFactor` (top center) and `RMS` (bottom left) appear effective, as they have only small areas of overlap. Conversely, `Skewness` (bottom center) and `Kurtosis` (middle left) have large amounts of overlap. These features appear ineffective for this data and this condition variable.

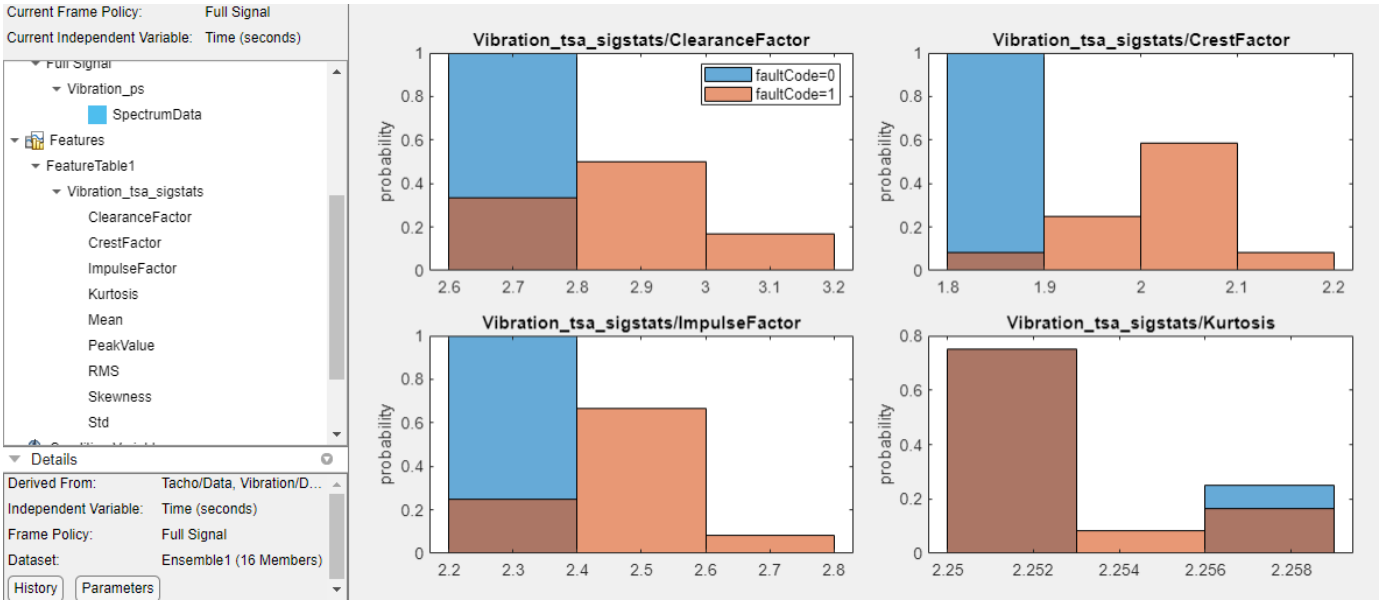
If you want to see how the histogram represents a specific condition, click on the color box that represents that condition in the legend. The app outlines the parts of the histogram that represent where that condition is present. This condition outline is particularly useful when you are evaluating histograms for more than two conditions.



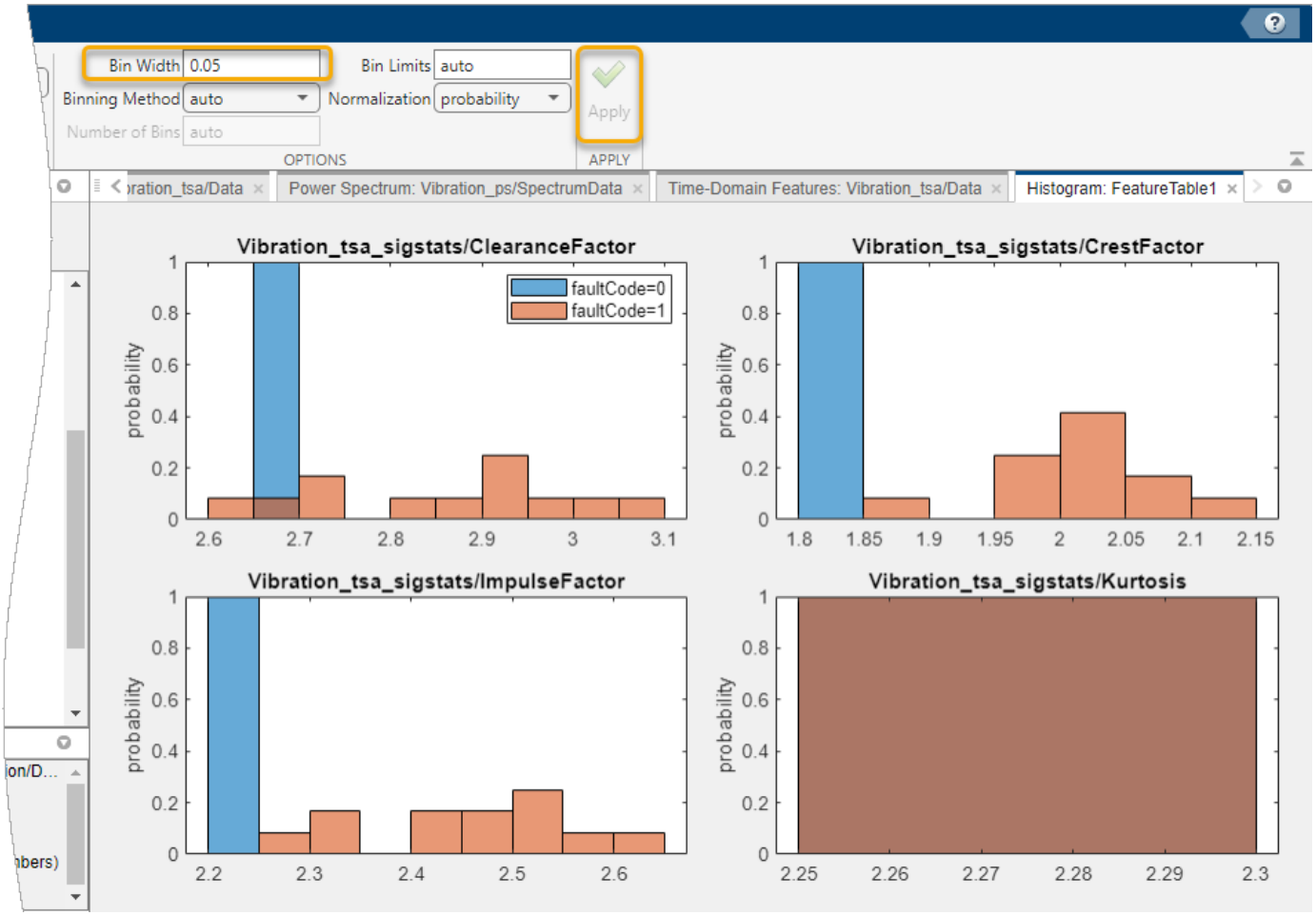
By default, the app plots the histograms for all the features in the feature table. You can focus on a subset of histograms by using **Select Features**. Use **Select Features** to limit the histogram plots to the first four in the feature table. Because the features are not yet ranked, the app sorts the features by name. When you use **Select Features** after ranking, you can select ranking order (the default) or name order.



The histogram view now includes only the features you select.

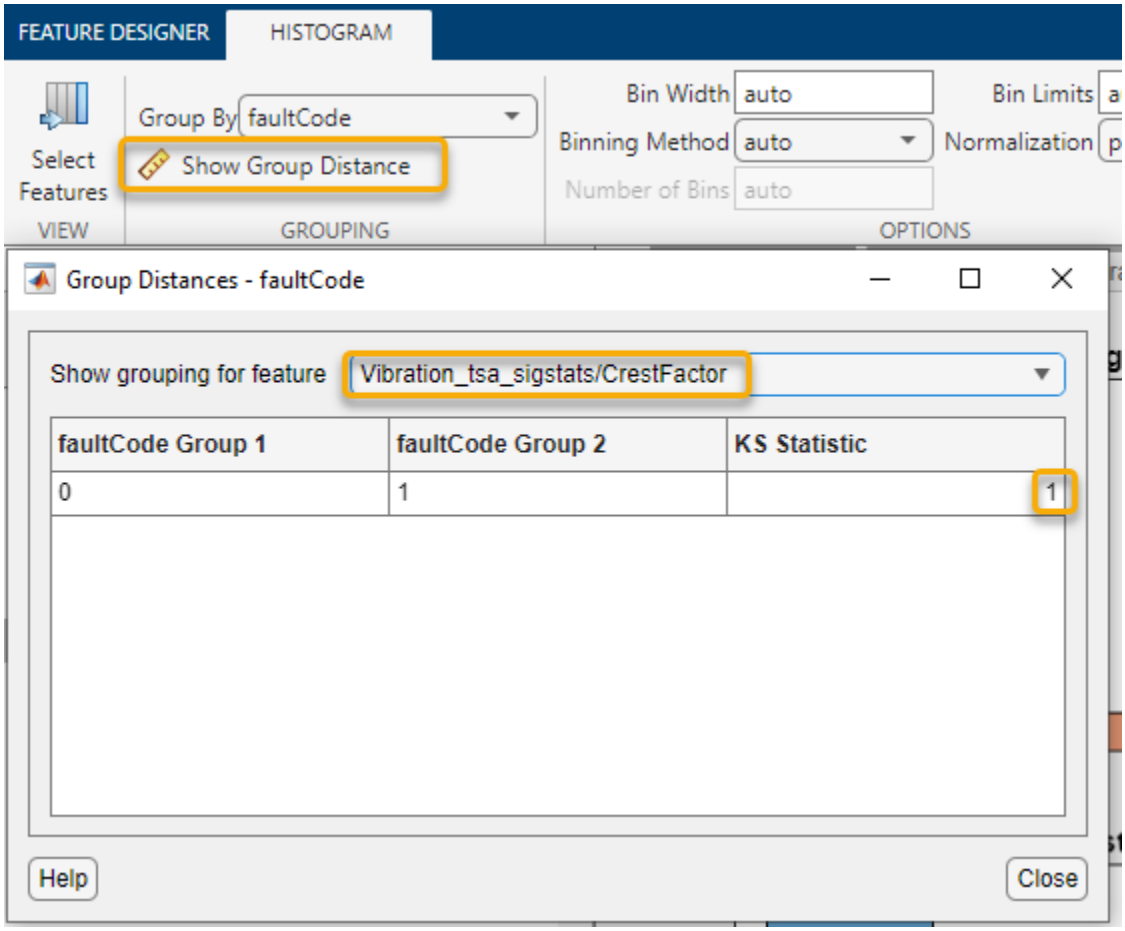


Control the appearance of the histograms using the parameters in the **Histogram** tab, which activates when you generate the histograms. The `CrestFactor` feature appears to separate healthy and unhealthy data almost completely. Investigate whether this result is sensitive to resolution. In the **Histogram** tab, the `auto` setting of bin width results in a resolution of 0.1 for `CrestFactor`. Enter a bin width 0.05, and click **Apply**.



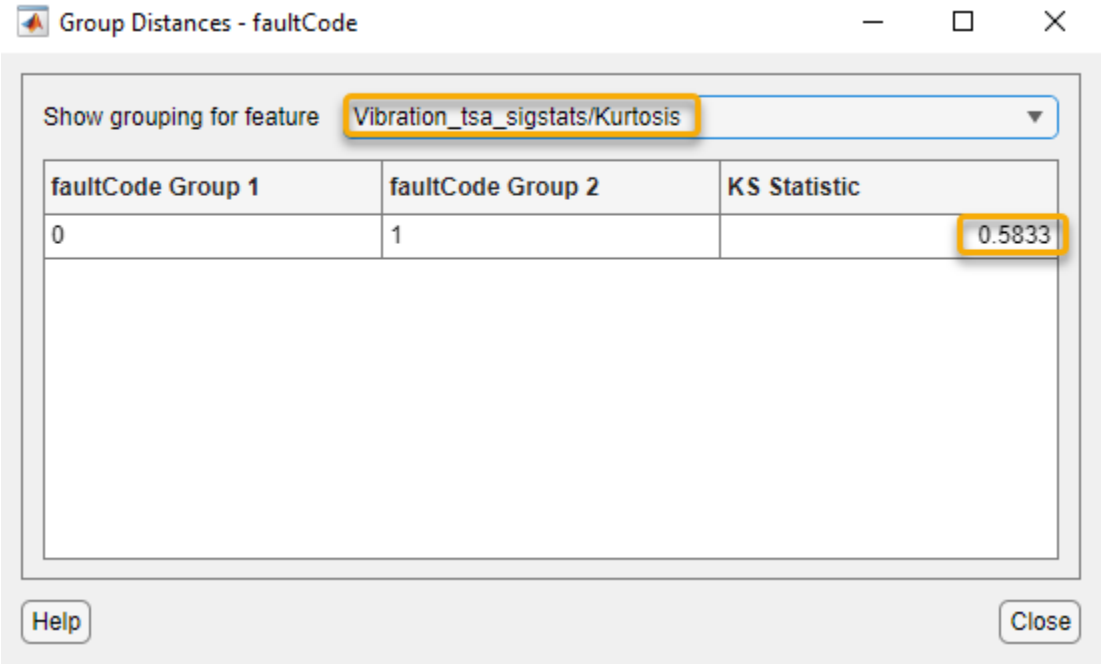
At this resolution, both `CrestFactor` and `ImpulseFactor` appear to completely separate healthy from degraded data. `ClearanceFactor` still has some mixed data, but to a lesser degree than with the larger bin width. `Kurtosis` has a smaller bin width of 0.002 with the `auto` bin width setting. Changing the bin width to 0.05 results in a single bin that contains all the `Kurtosis` data.

Histograms visualize the ability of features to separate healthy from unhealthy data. You can also get a numerical assessment using group distance. The group distance represents the separation between the healthy and unhealthy data distributions. Click **Show Group Distance**. In the dialog box, select `CrestFactor` in **Show grouping for feature**.



The group distance, represented by the KS Statistic, is 1. This value represents complete separation.

Next, select `Kurtosis`. The `Kurtosis` histogram shows substantial intermixing.

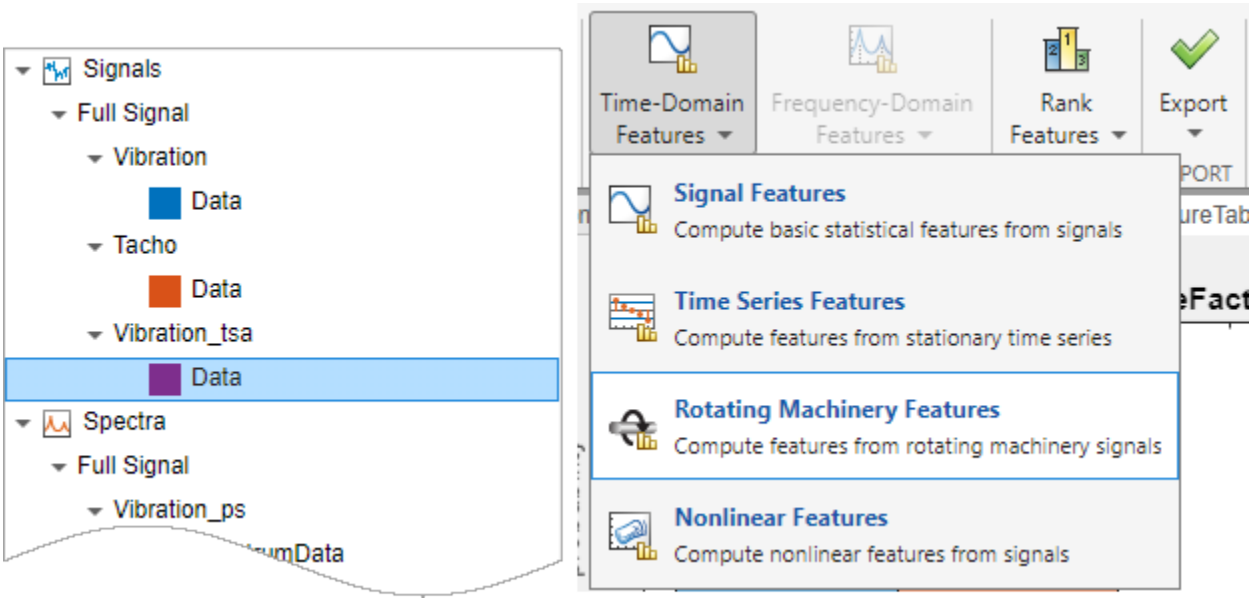


The KS Statistic in this case is about 0.6, reflecting the intermixing in the histogram.

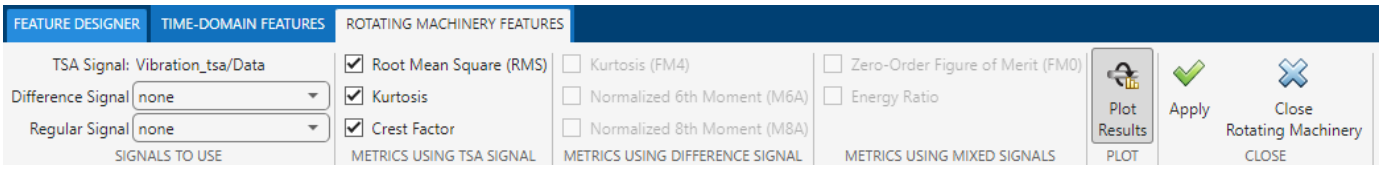
Restore **Bin Width** to `auto`.

Rotating Machinery Features

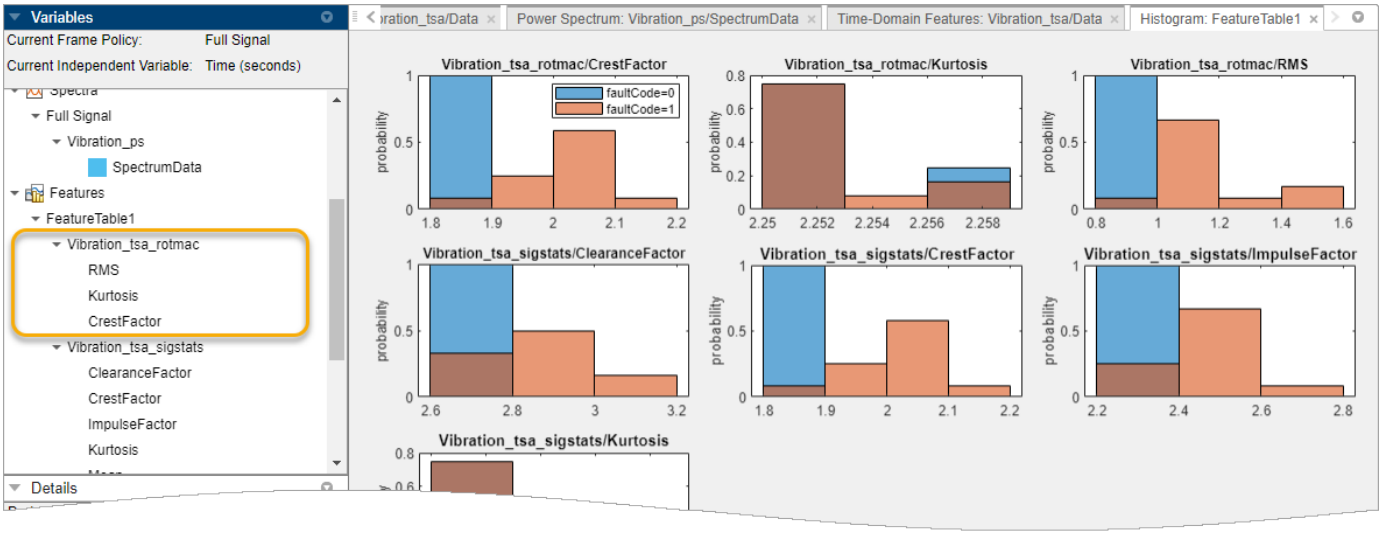
Since you have rotating machinery, compute rotating machinery features. In the variables pane, select your TSA signal. Then, select **Time-Domain Features** > **Rotating Machinery Features**.



The **Rotating Machinery Features** tab can create features from TSA signals as well as from TSA difference signals and TSA regular signals. Since you have only a TSA signal, the app disables the selections that do require a different signal type.



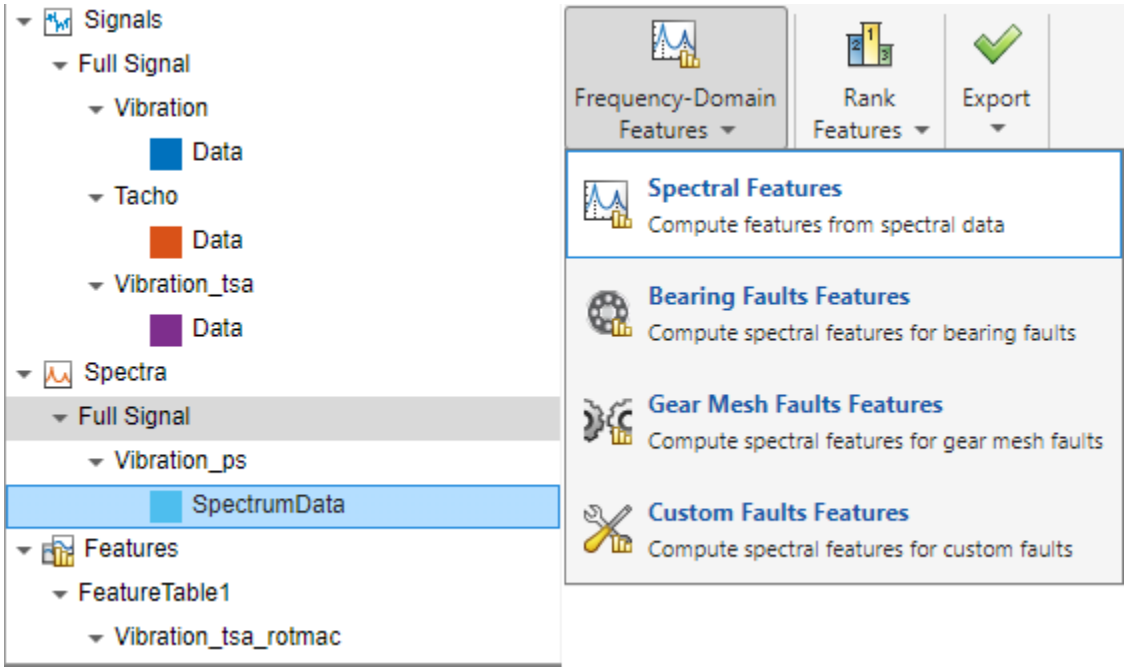
Accept the default selections by clicking **Apply**.



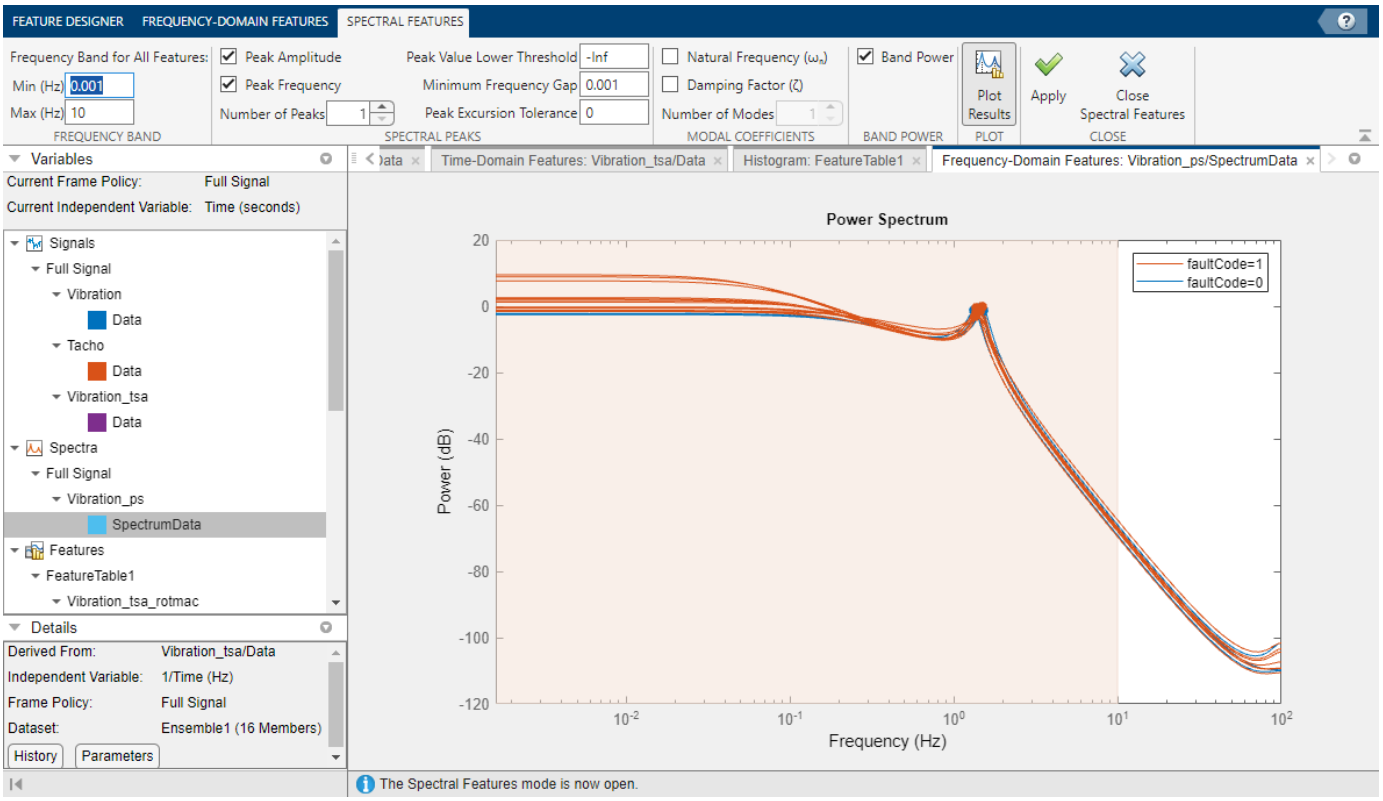
The app automatically adds the new features to the feature table and the **Select Features** list, and plots the new histograms at the top of the histogram display. `CrestFactor` and `Kurtosis` histograms are essentially the same whether they are computed as signal features or rotating machinery features, since both computations use the TSA signal as a source.

Spectral Features

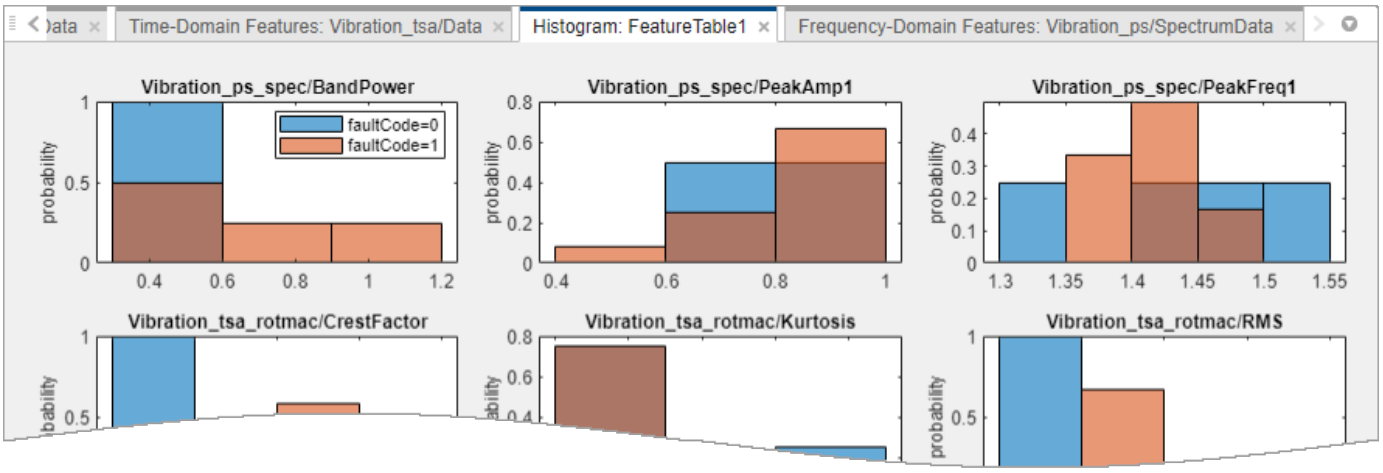
Compute spectral features from the power spectrum you generated earlier. Select `Vibration_ps/SpectrumData`. Then, select **Frequency-Domain Features** > **Spectral Features**.



Specify the frequency band to use by setting the minimum and maximum band values. To capture the power spectrum peaks efficiently, limit the frequency range to a maximum of `10` Hz, starting at `0.001` Hz. The plot displays this band as an orange rectangle underlaying the frequency plot.

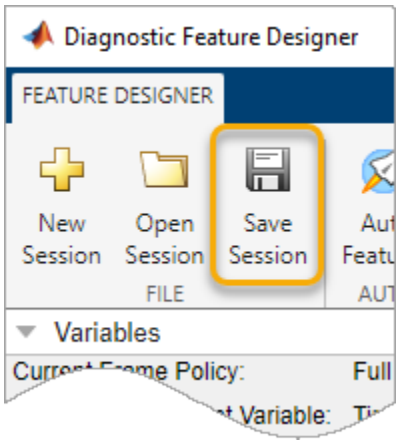


The histograms show substantial intermixing of healthy and unhealthy data in one or more of the bins for all three features.



You now have a diverse set of features.

Save your session data. You need this data to run the “Rank and Export Features in Diagnostic Feature Designer” on page 2-37 example.



Next Steps

The next step is to rank these features to determine which ones provide the best indication of system condition. For more information, see “Rank and Export Features in Diagnostic Feature Designer” on page 2-37.

See Also

Diagnostic Feature Designer

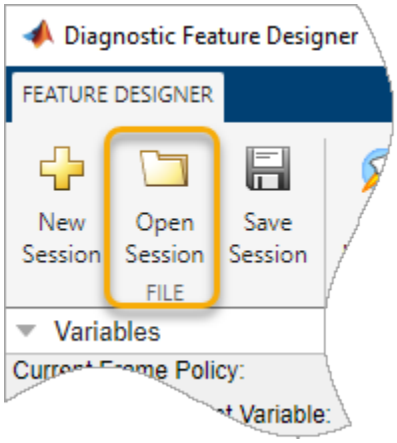
More About

- “Open Diagnostic Feature Designer” on page 2-5

- “Rank and Export Features in Diagnostic Feature Designer” on page 2-37
- “Explore Ensemble Data and Compare Features Using Diagnostic Feature Designer”
- “Interpret Feature Histograms in Diagnostic Feature Designer”
- “Data Preprocessing for Condition Monitoring and Predictive Maintenance”
- “Condition Indicators for Monitoring, Fault Detection, and Prediction”
- “Condition Indicators for Gear Condition Monitoring”

Rank and Export Features in Diagnostic Feature Designer

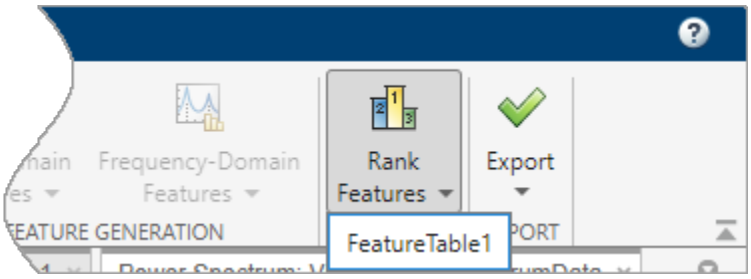
This example shows how to rank features using several classification ranking techniques, how to compare the results, and how to export features from the app. If you want to follow along with the steps interactively, use the data you imported in “Process Data and Explore Features in Diagnostic Feature Designer” on page 2-16, and use the feature set from that example. Use **Open Session** to reload your session data using the file name you provided.



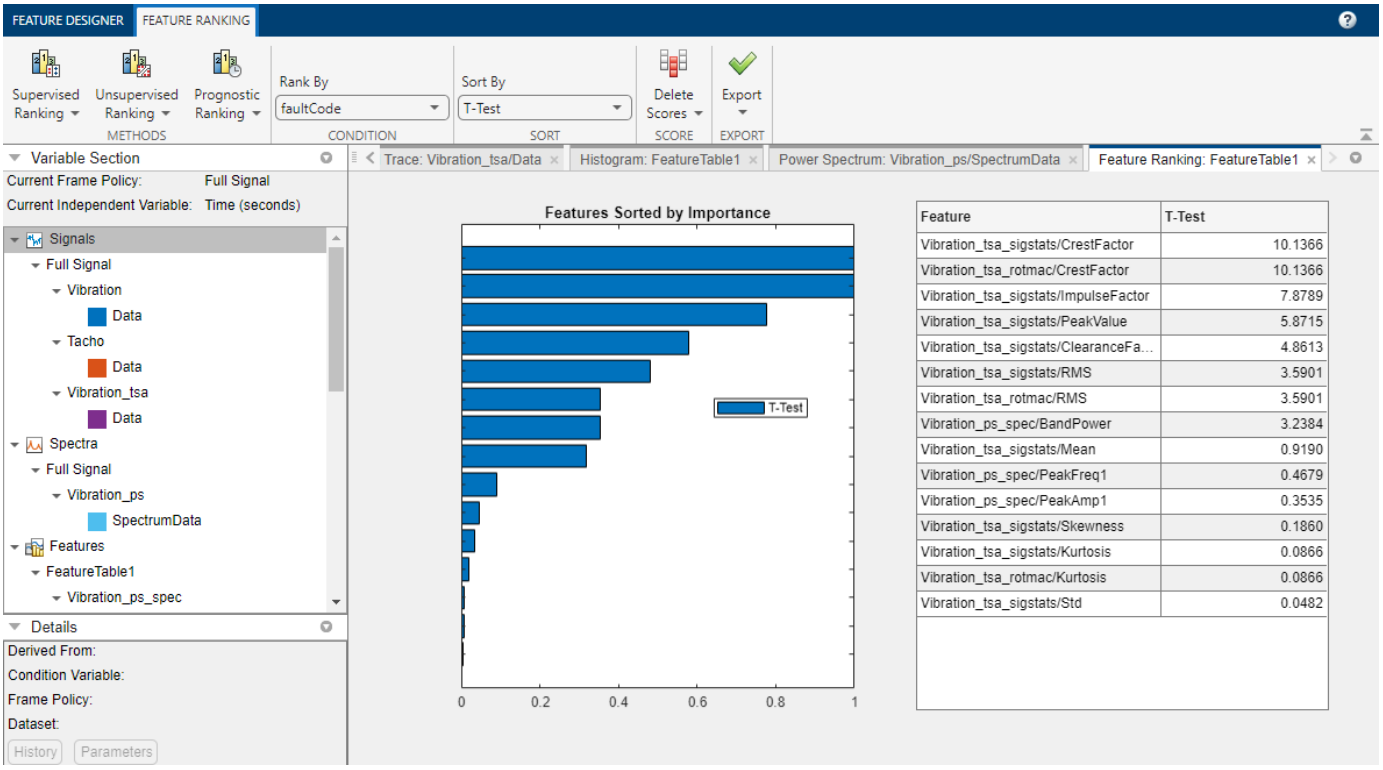
When you generate features for classification, you seek the features that perform best at distinguishing among different conditions. When you view the histograms, you can get an idea of the relative effectiveness of your features. In this example, you use ranking algorithms to perform this feature comparison more rigorously. Once you choose the features you want to retain, you export these features into the MATLAB workspace.

Rank Features

Rank your features using the default `T-Test` method. Click **Rank Features**. Select `FeatureTable1`.



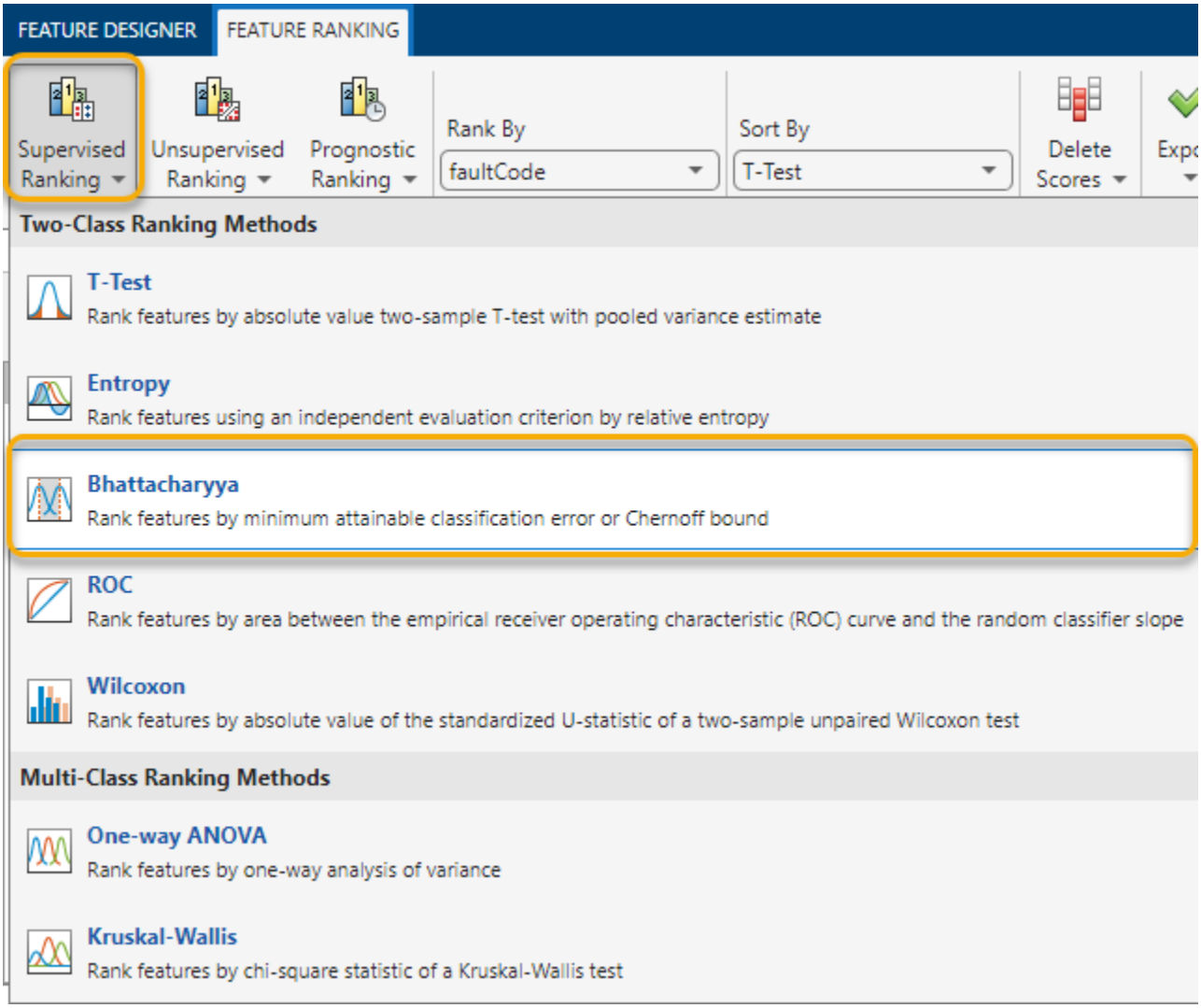
Your selection brings up a ranked list of features, displayed as both a bar chart and a numerical table.



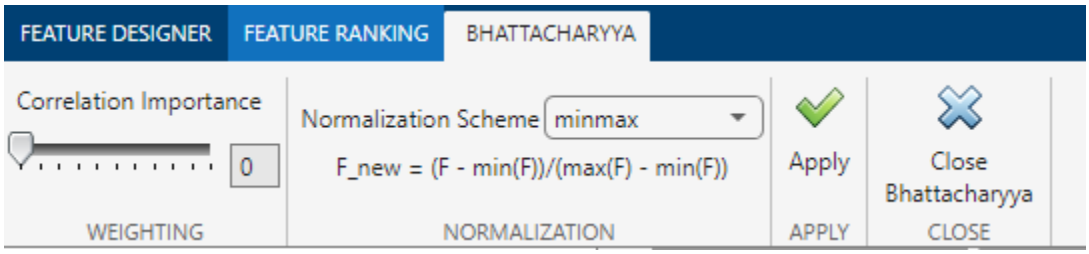
The bar chart legend shows that the initial ranking is performed using the `T-Test` method. The chart is normalized to 1 to facilitate visual comparison, while the table displays unnormalized ranking scores. The highest ranking feature is `CrestFactor`, which has the same value whether it was computed as a signal feature or a rotating machinery feature.

Choose Alternative Ranking Method

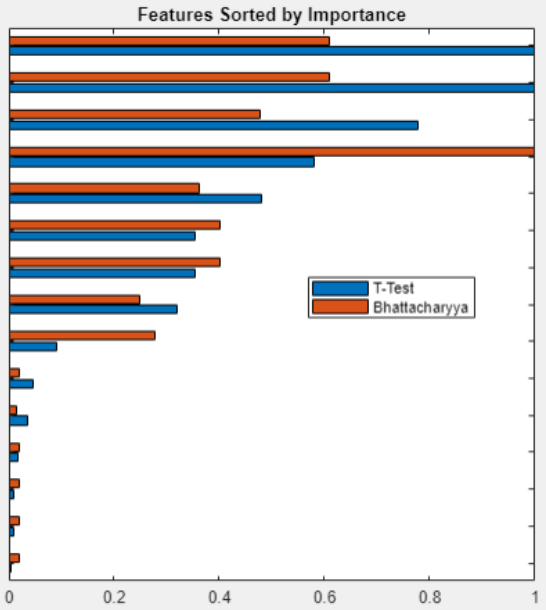
Each ranking method uses different criteria to perform the ranking. In the **Feature Ranking** tab, click **Supervised Ranking** to bring up a menu that summarizes each method. From that menu, select `Bhattacharyya`.



A **Bhattacharyya** tab opens with ranking specifications that are standard for all of the methods. Click **Apply**.



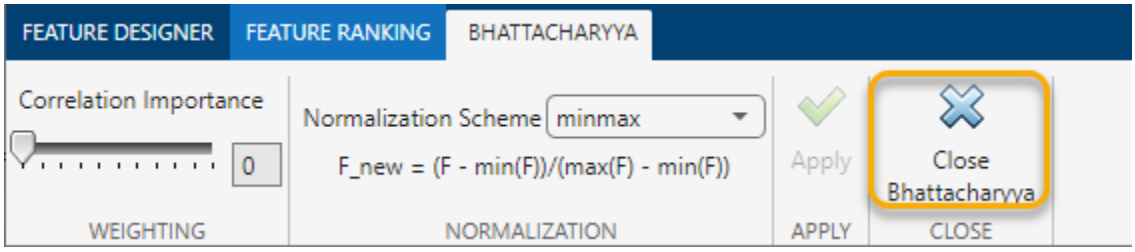
Apply updates the ranking display with the new results, displayed along with the original `T-Test` results.

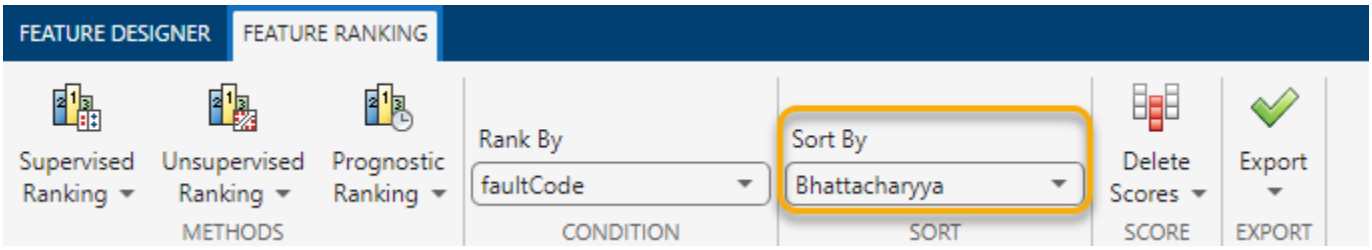


Feature	T-Test	Bhattacharyya
Vibration_tsa_sigstats/CrestFactor	10.1366	3.3698
Vibration_tsa_rotmac/CrestFactor	10.1366	3.3698
Vibration_tsa_sigstats/ImpulseFactor	7.8789	2.6337
Vibration_tsa_sigstats/PeakValue	5.8715	5.5265
Vibration_tsa_sigstats/ClearanceFa...	4.8613	2.0050
Vibration_tsa_sigstats/RMS	3.5901	2.2244
Vibration_tsa_rotmac/RMS	3.5901	2.2244
Vibration_ps_spec/BandPower	3.2384	1.3687
Vibration_tsa_sigstats/Mean	0.9190	1.5317
Vibration_ps_spec/PeakFreq1	0.4679	0.1042
Vibration_ps_spec/PeakAmp1	0.3535	0.0842
Vibration_tsa_sigstats/Skewness	0.1860	0.1078
Vibration_tsa_sigstats/Kurtosis	0.0866	0.1085
Vibration_tsa_rotmac/Kurtosis	0.0866	0.1085
Vibration_tsa_sigstats/Std	0.0482	0.1038

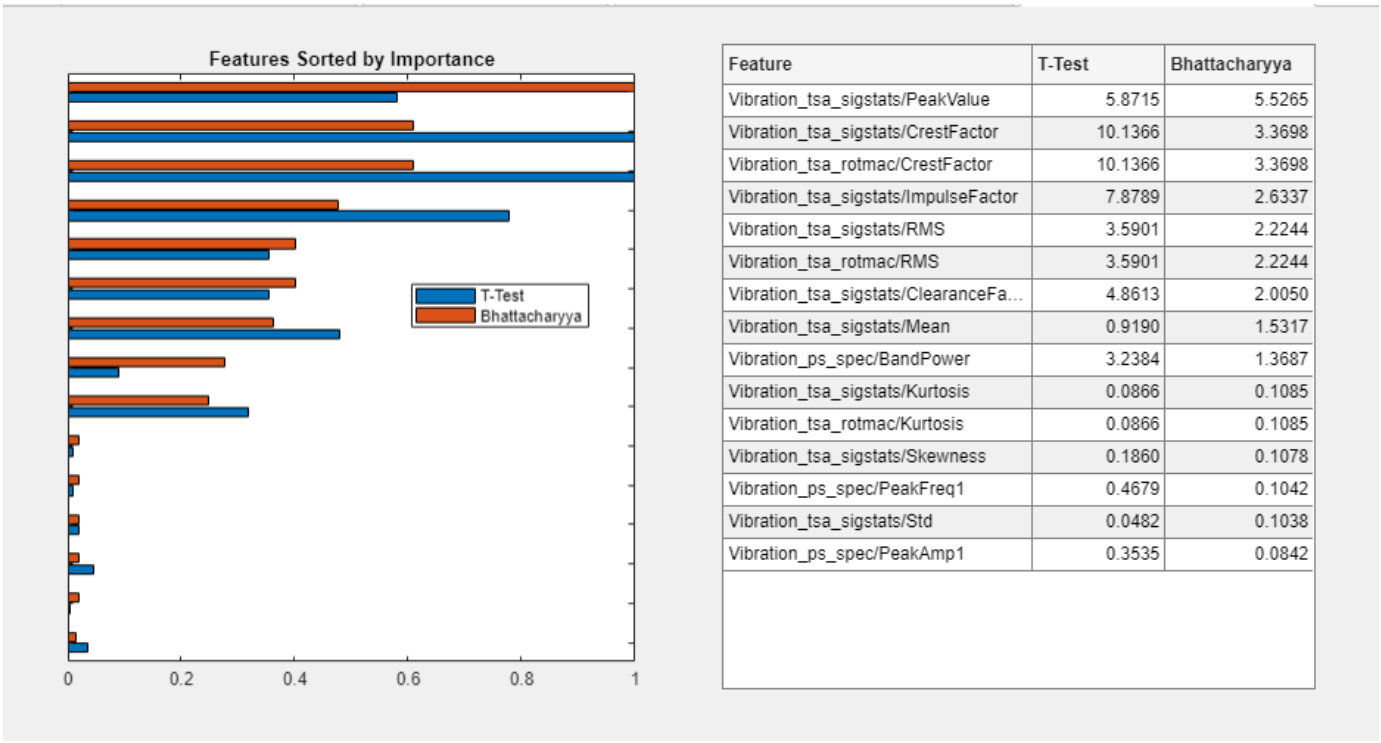
The `Bhattacharyya` method yields results that are similar to, but not identical to, the `T-Test` results. The highest ranking feature is `PeakValue` from the `Signal Statistics` set. This feature is fourth in the T-test ranking. The crest factor features are still in the top three.

The ranking is still sorted by `T-Test`. Sort instead by `Bhattacharyya`. Close the **Bhattacharyya** tab and return to the **Feature Ranking** tab. Then, select `Bhattacharyya` in the **Sort by** list.





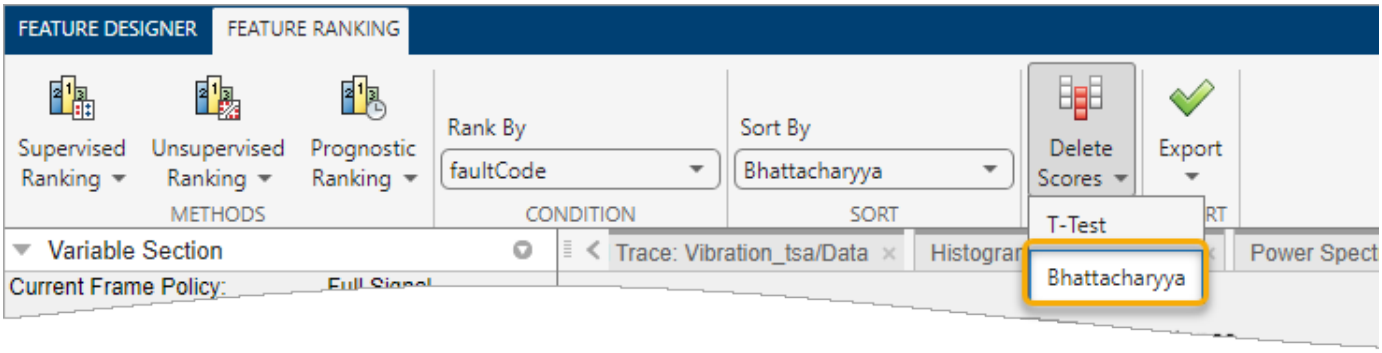
The ranking table now shows `PeakValue` at the top.



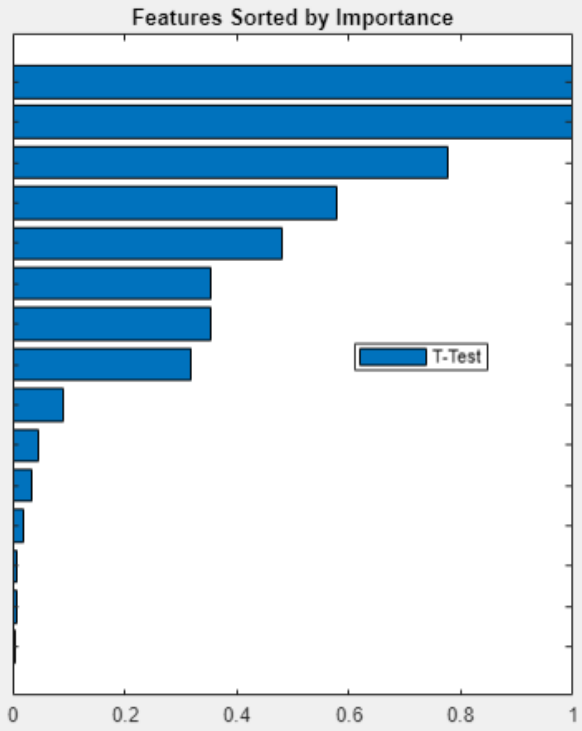
Feature	T-Test	Bhattacharyya
Vibration_tsa_sigstats/PeakValue	5.8715	5.5265
Vibration_tsa_sigstats/CrestFactor	10.1366	3.3698
Vibration_tsa_rotmac/CrestFactor	10.1366	3.3698
Vibration_tsa_sigstats/ImpulseFactor	7.8789	2.6337
Vibration_tsa_sigstats/RMS	3.5901	2.2244
Vibration_tsa_rotmac/RMS	3.5901	2.2244
Vibration_tsa_sigstats/ClearanceFa...	4.8613	2.0050
Vibration_tsa_sigstats/Mean	0.9190	1.5317
Vibration_ps_spec/BandPower	3.2384	1.3687
Vibration_tsa_sigstats/Kurtosis	0.0866	0.1085
Vibration_tsa_rotmac/Kurtosis	0.0866	0.1085
Vibration_tsa_sigstats/Skewness	0.1860	0.1078
Vibration_ps_spec/PeakFreq1	0.4679	0.1042
Vibration_tsa_sigstats/Std	0.0482	0.1038
Vibration_ps_spec/PeakAmp1	0.3535	0.0842

Delete Set of Rankings

You have two sets of rankings. Now, delete the `Bhattacharyya` results. In the **Feature Ranking** tab, select **Delete Scores > Bhattacharyya**.



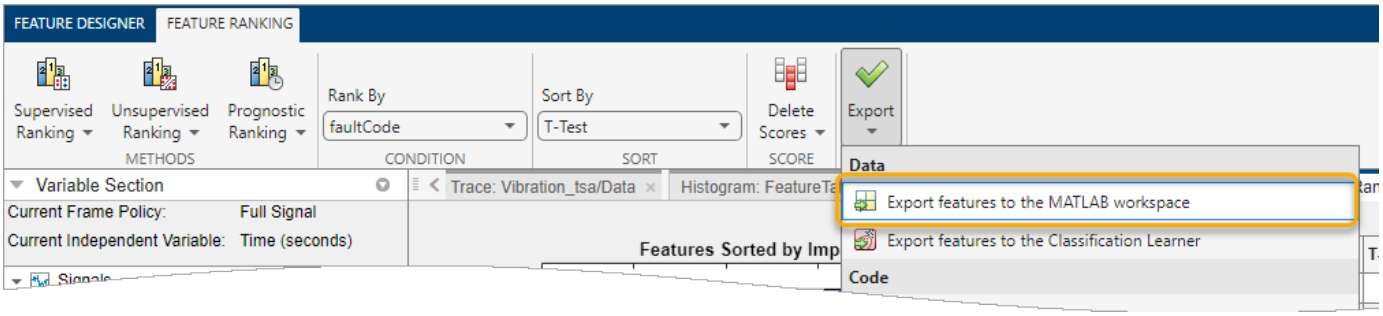
`Bhattacharyya` disappears from the ranking results.



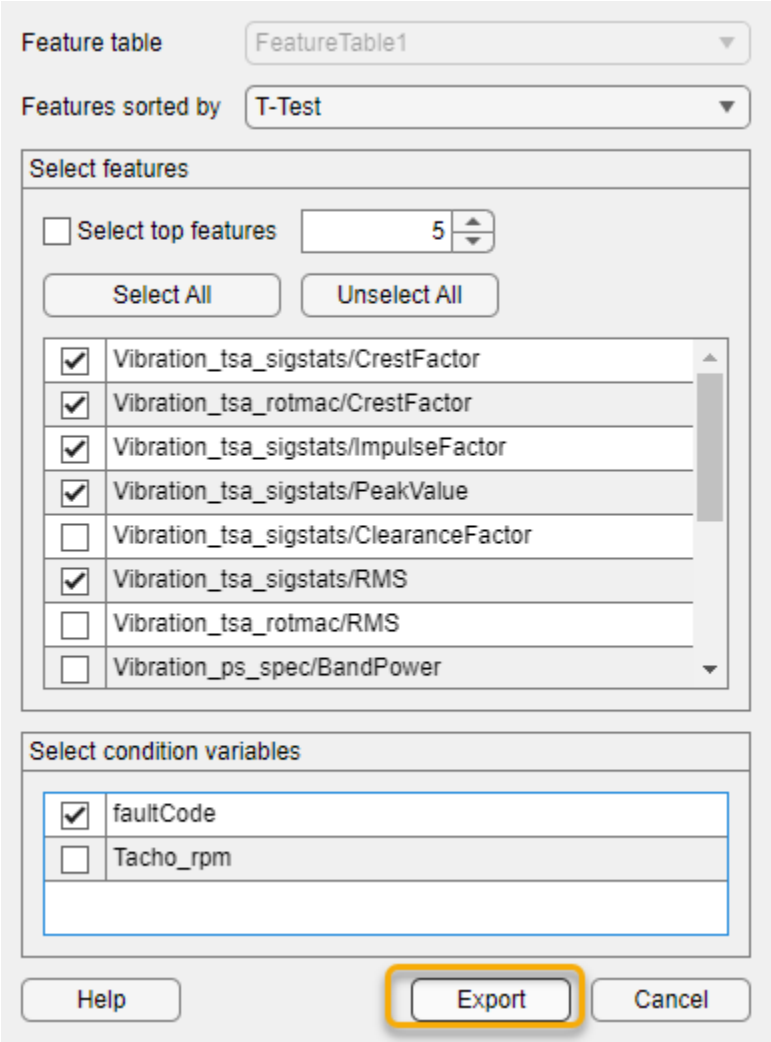
Feature	T-Test
Vibration_tsa_sigstats/CrestFactor	10.1366
Vibration_tsa_rotmac/CrestFactor	10.1366
Vibration_tsa_sigstats/ImpulseFactor	7.8789
Vibration_tsa_sigstats/PeakValue	5.8715
Vibration_tsa_sigstats/ClearanceFa...	4.8613
Vibration_tsa_sigstats/RMS	3.5901
Vibration_tsa_rotmac/RMS	3.5901
Vibration_ps_spec/BandPower	3.2384
Vibration_tsa_sigstats/Mean	0.9190
Vibration_ps_spec/PeakFreq1	0.4679
Vibration_ps_spec/PeakAmp1	0.3535
Vibration_tsa_sigstats/Skewness	0.1860
Vibration_tsa_sigstats/Kurtosis	0.0866
Vibration_tsa_rotmac/Kurtosis	0.0866
Vibration_tsa_sigstats/Std	0.0482

Export Features to MATLAB Workspace

The final step in the **Diagnostic Features Designer** workflow is to export your features. In the **Feature Ranking** tab, select **Export > Export features to the MATLAB workspace**.



Select the features to export. You can sort the features by any of the rankings you have computed. In this case, only one ranking, `T-Test`, is available. The app preselects the top five features. Modify this selection. Clear the fifth selection and select the sixth feature using Ctrl-click.



Your reduced feature table appears in your MATLAB workspace.

See Also

Diagnostic Feature Designer | `anova1` | `bhattacharyyaDistance` | `kruskalwallis` | `perfcurve` | `ranksum` | `relativeEntropy` | `ttest2`

More About

- “Process Data and Explore Features in Diagnostic Feature Designer” on page 2-16
- “Explore Ensemble Data and Compare Features Using Diagnostic Feature Designer”
- “Perform Prognostic Feature Ranking for a Degrading System Using Diagnostic Feature Designer”
- “Data Preprocessing for Condition Monitoring and Predictive Maintenance”