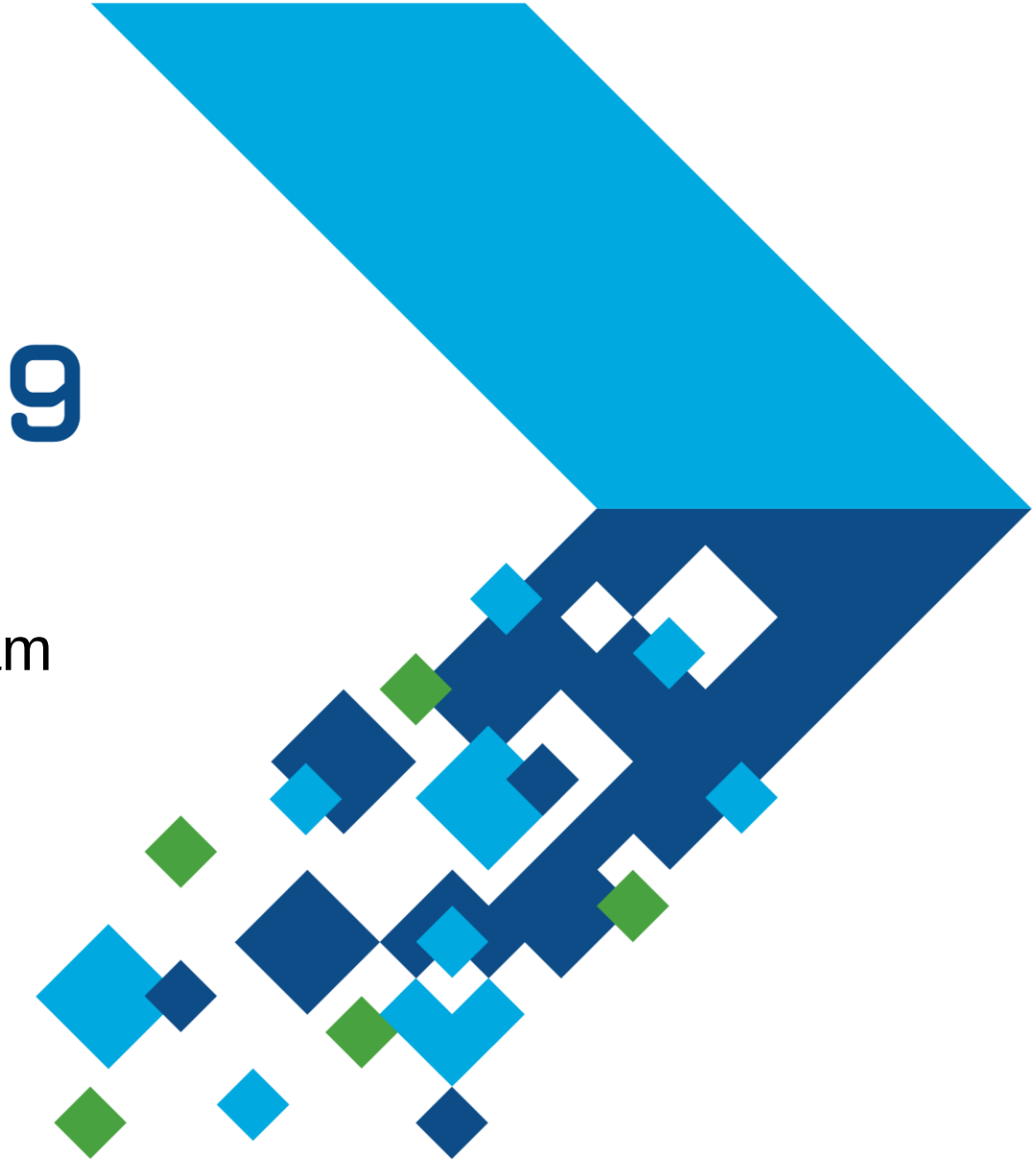
MATLAB EXPO 2019

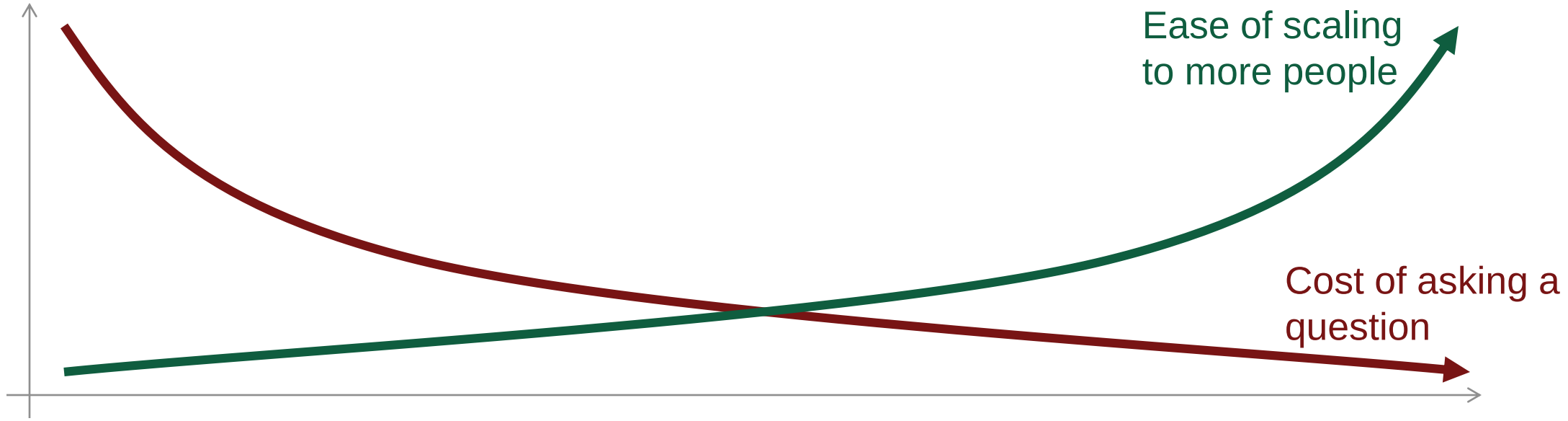
Becoming a Data-Centric Engineering Team

Paul Peeling



A **path** for how your team can better **explore**, **understand** and **analyze data**.

Data Science Maturity Levels



Data Science Maturity Levels

- **Goal is to be fast: reduce time to insight**

Overview of Flight Data

- **35** unique aircraft
- **180,000** unique flights

- **300 GB** of data

- Source:
  - NASA Dash Link: Sample Flight Data
  - https://c3.nasa.gov/dashlink/projects/85/

Big Data Creates Opportunities

Find rare events, then dive deeper

Build and validate test scenarios that match real-world conditions

Perform fleet-wide calculations

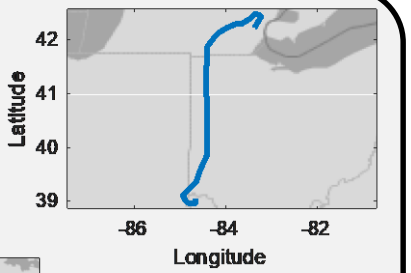
Exploring a New Dataset

1. Control data import with ImportOptions
2. Working with table and timetable datatypes
3. Interacting with data in the Live Editor
4. Filling in outliers with a Live Task
5. Synchronizing time-based data
6. New geographic visualizations

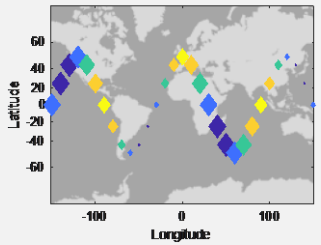
```
t = synchronize(t1hz,t4hz)
t = fillmissing(t,'previous')
inFlight = t.WOW==1;
geoplot(t.LATP(inFlight),t.LONP(inFlight),'LineWidth',3)
```

Missing Data

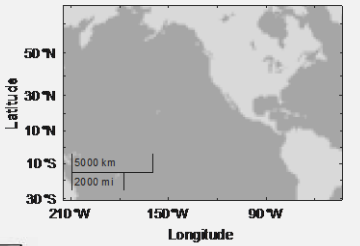
geoplot



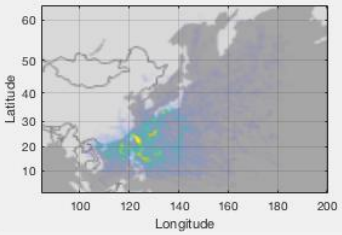
geoscatter



geobubble



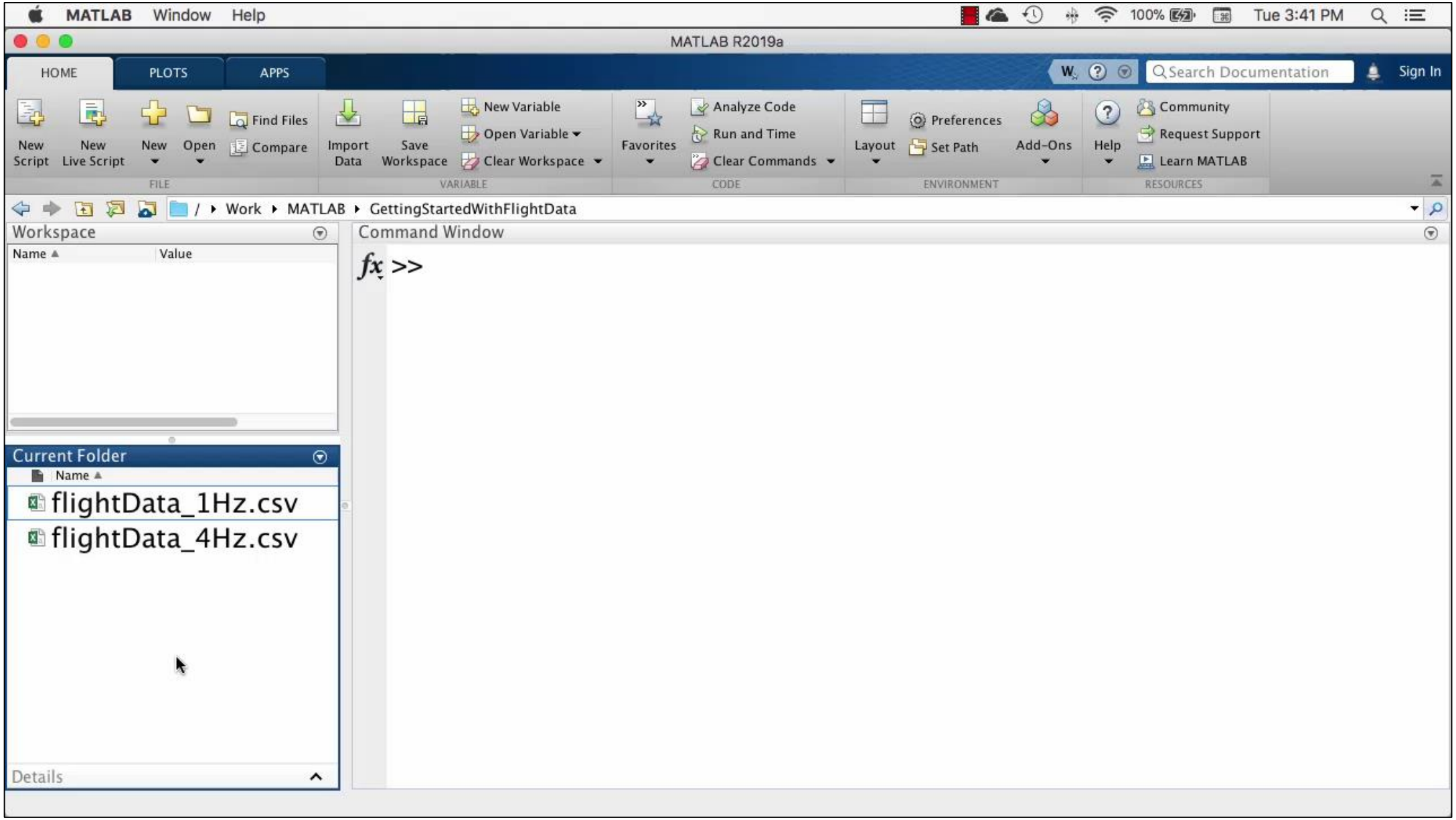
geodensityplot



https:/
matla

https://www.mathworks.com/help/matlab/geographic-plots.html

Getting Started: **Exploring a New Dataset**



Getting Started: **Exploring a New Dataset**

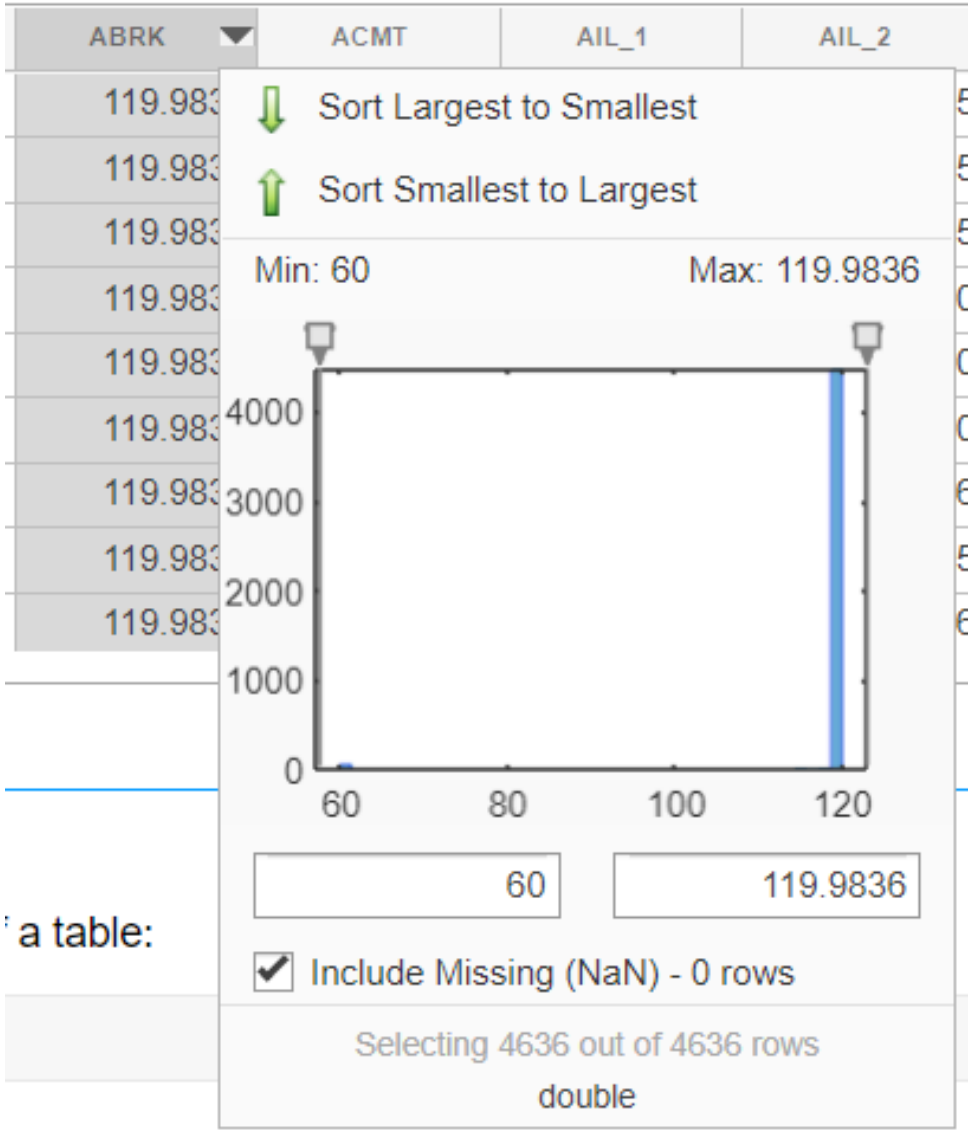
Import Data

```
Import1HzData
```

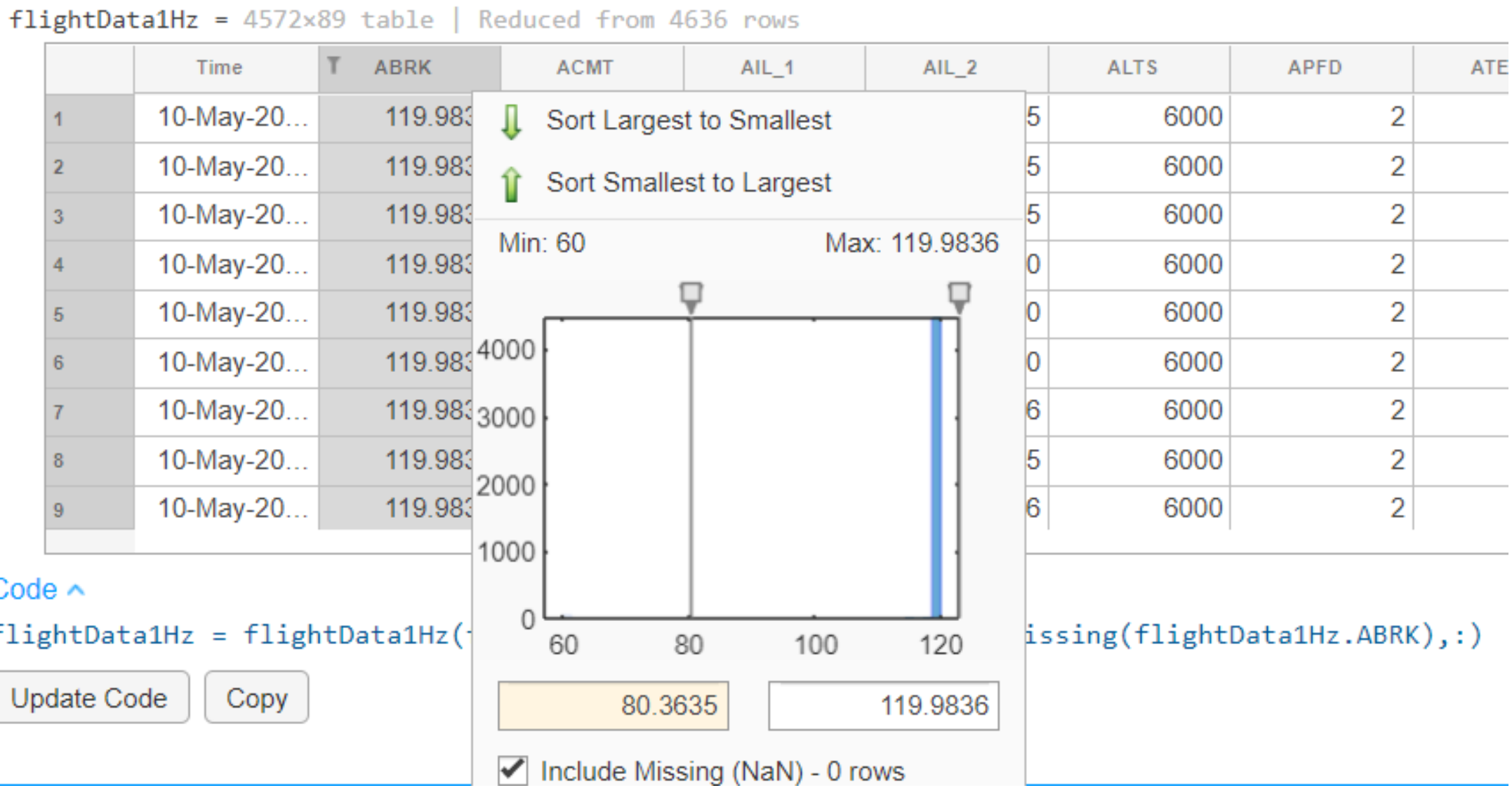
flightData1Hz = 4636×89 table

	Time	ABRK	ACMT	AIL_1	AIL_2	ALTS
1	10-May-20...	119.9836	59	91.8789	91.5925	6000
2	10-May-20...	119.9836	59	91.8994	91.5925	6000
3	10-May-20...	119.9836	60	91.8994	91.5925	6000
4	10-May-20...	119.9836	59	91.8994	91.5720	6000
5	10-May-20...	119.9836	60	91.8789	91.5720	6000
6	10-May-20...	119.9836	60	91.9607	91.5720	6000
7	10-May-20...	119.9836	59	91.9198	91.5516	6000
8	10-May-20...	119.9836	60	91.8789	91.5925	6000
9	10-May-20...	119.9836	59	91.8175	91.5516	6000

Getting Started: **Exploring a New Dataset**



Getting Started: **Exploring a New Dataset**



Getting Started: **Exploring a New Dataset**

Slicing the data

Access rows or columns of a table:

```
flightData1Hz(1,:)
```

ans = 1×89 table

	Time	ABRK	ACMT	AIL_1
1	10-May-20…	119.9836	59	91.8789

```
flightData1Hz(:,1)
```

ans = 4636×1 table

	Time
1	10-May-20…
2	10-May-20…
3	10-May-20…
4	10-May-20…
5	10-May-20…
6	10-May-20…
7	10-May-20…
8	10-May-20…
9	10-May-20…

Accessing a variable:

```
flightData1Hz.ABRK
```

```
ans = 4636×1
    119.9836
    119.9836
    119.9836
    119.9836
    119.9836
    119.9836
    119.9836
    119.9836
    119.9836
    119.9836
       ⋮
```

Access variables by type:

```
flightData1Hz{:,vartype("numeric")}
```

```
ans = 4636×88
10^6 ×
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
    0.0001    0.0001    0.0001    0.0001
       ⋮
```

```
flightData1Hz{:,vartype("datetime")}
```

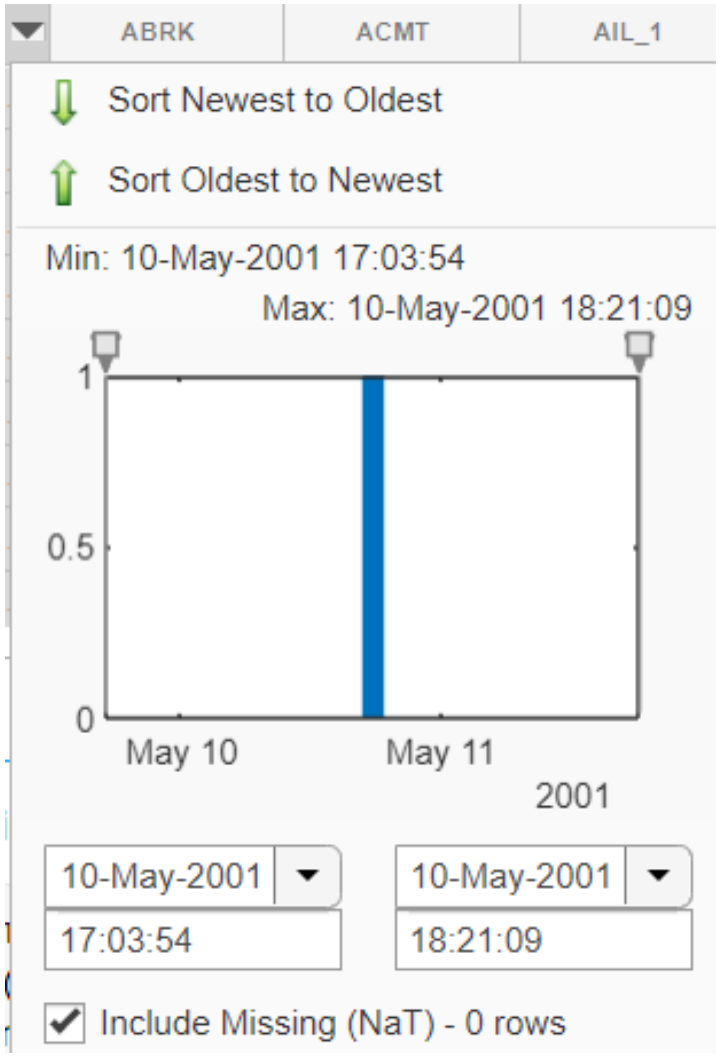
```
ans = 4636×1 datetime array
   10-May-2001 17:03:54
   10-May-2001 17:03:55
   10-May-2001 17:03:56
   10-May-2001 17:03:57
   10-May-2001 17:03:58
   10-May-2001 17:03:59
   10-May-2001 17:04:00
   10-May-2001 17:04:01
   10-May-2001 17:04:02
   10-May-2001 17:04:03
```

Getting Started: **Exploring a New Dataset**

```
t1hz = table2timetable(flightData1Hz)
```

t1hz = 4636×88 timetable

	Time	ABRK	ACMT	AIL_1
1	10-May-20…	119.9836	59	91.8789
2	10-May-20…	119.9836	59	91.8994
3	10-May-20…	119.9836	60	91.8994
4	10-May-20…	119.9836	59	91.8994
5	10-May-20…	119.9836	60	91.8789
6	10-May-20…	119.9836	60	91.9607
7	10-May-20…	119.9836	59	91.9198
8	10-May-20…	119.9836	60	91.8789
9	10-May-20…	119.9836	59	91.8175



Getting Started: **Exploring a New Dataset**

Index a timetable using time.

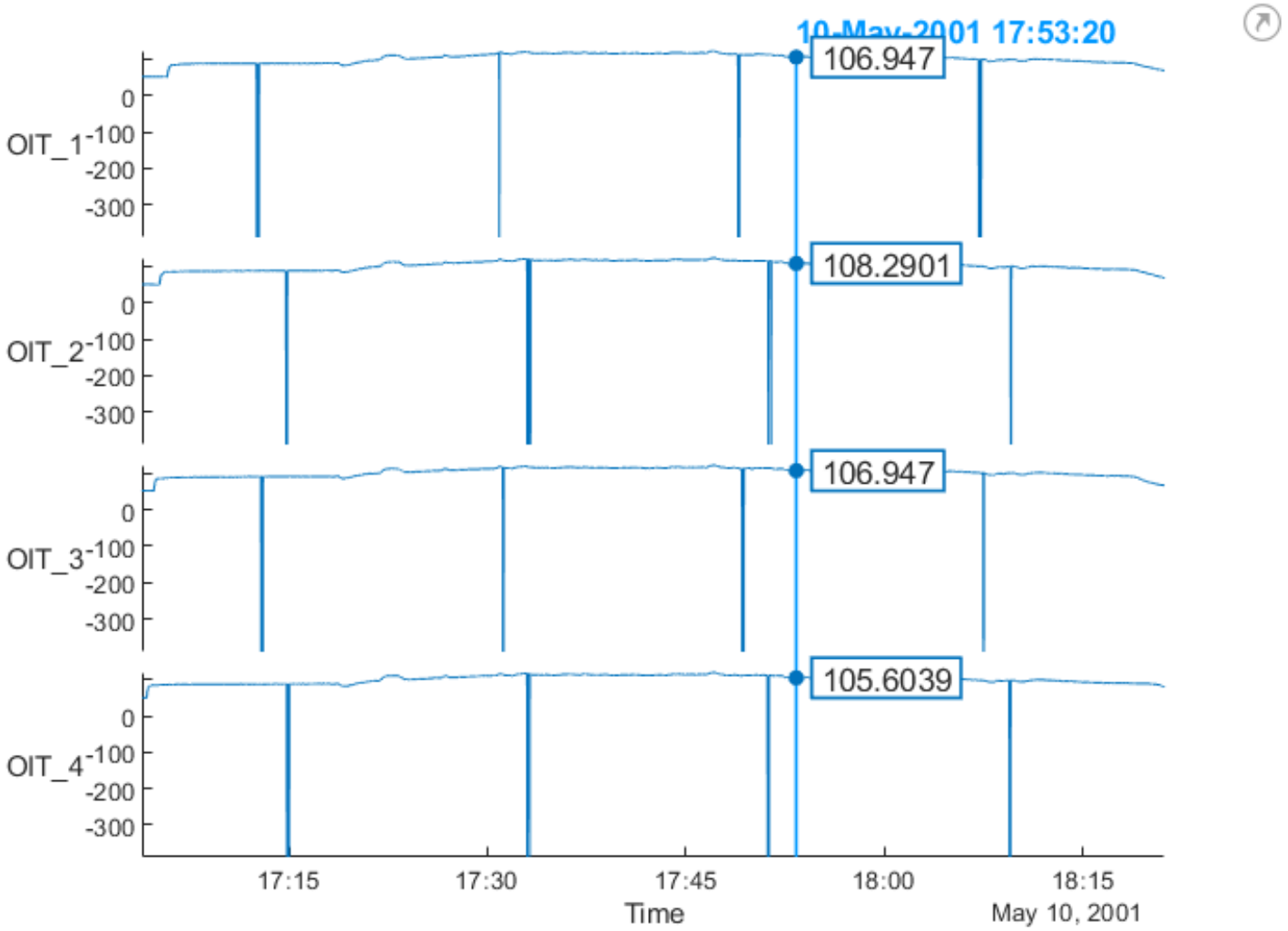
```
starttime = datetime(2001,5,10,17,30,0);
endtime = datetime(2001,5,10,17,30,5);
t1hz(timerange(starttime,endtime),:)
```

ans = 5×88 timetable

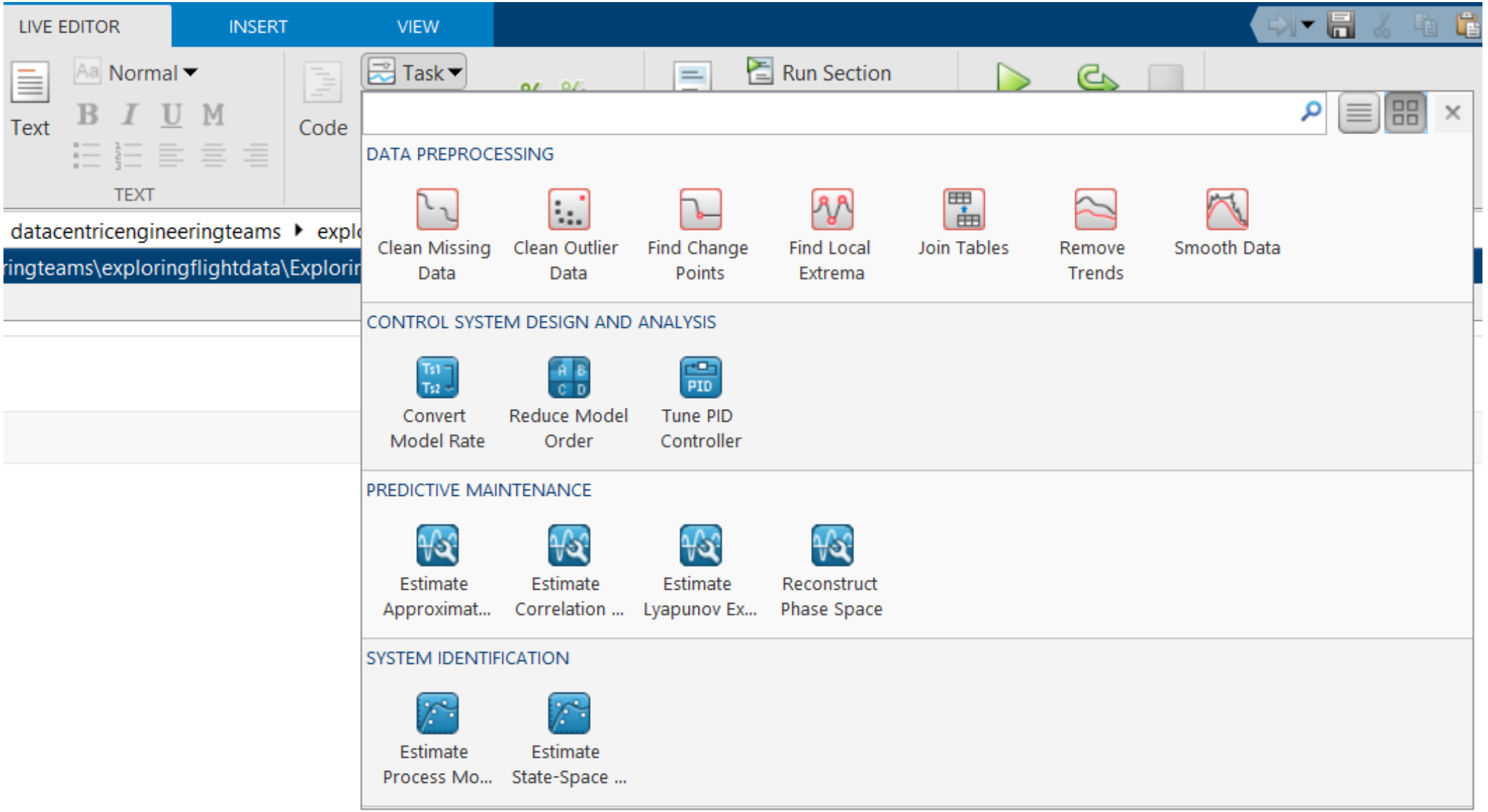
	Time	ABRK	ACMT	AIL_1
1	10-May-20…	119.9836	56	83.7775
2	10-May-20…	119.9836	56	83.8184
3	10-May-20…	119.9836	0	83.8798
4	10-May-20…	119.9836	56	84.0230
5	10-May-20…	119.9836	56	83.8389

Getting Started: **Exploring a New Dataset**

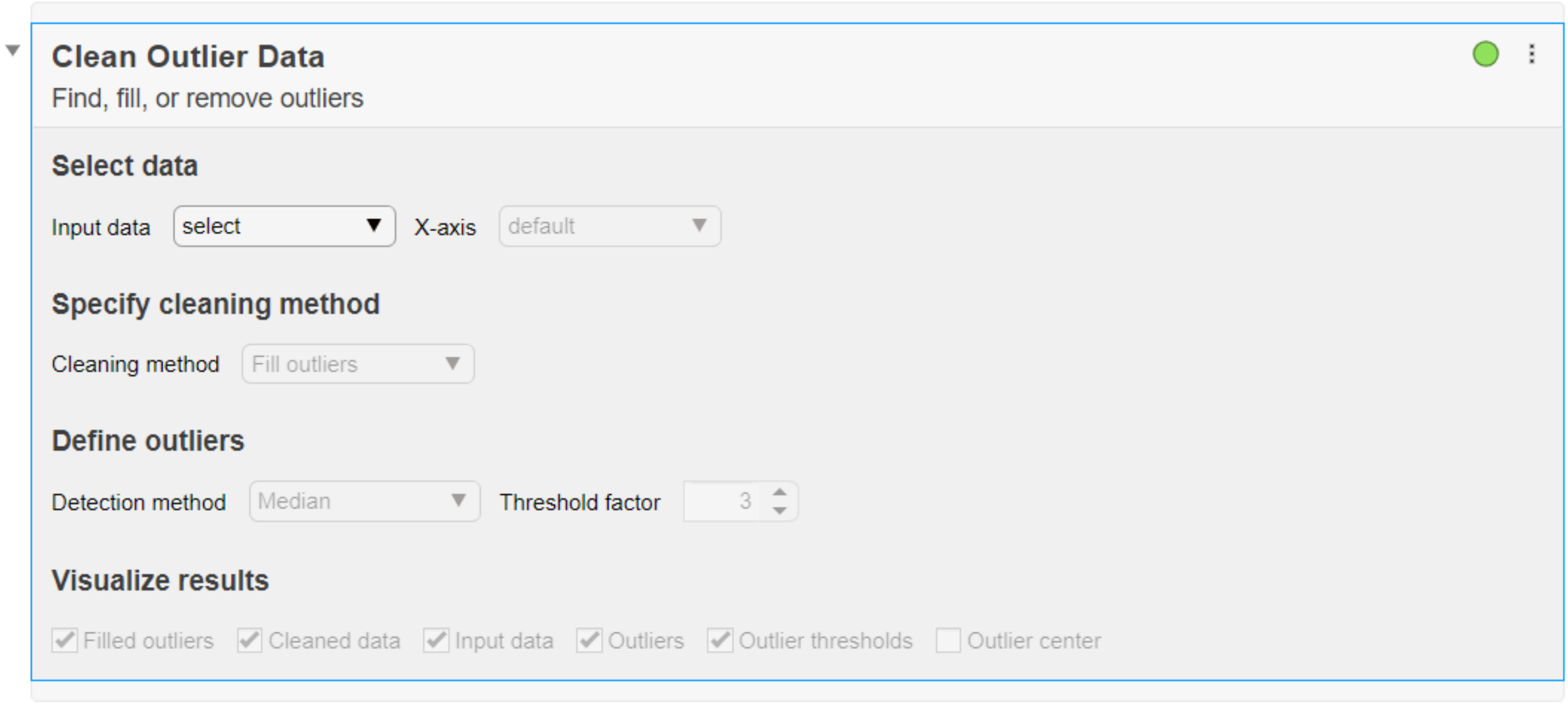
```
stackedplot(t1hz,{'OIT_1','OIT_2','OIT_3','OIT_4'});
```



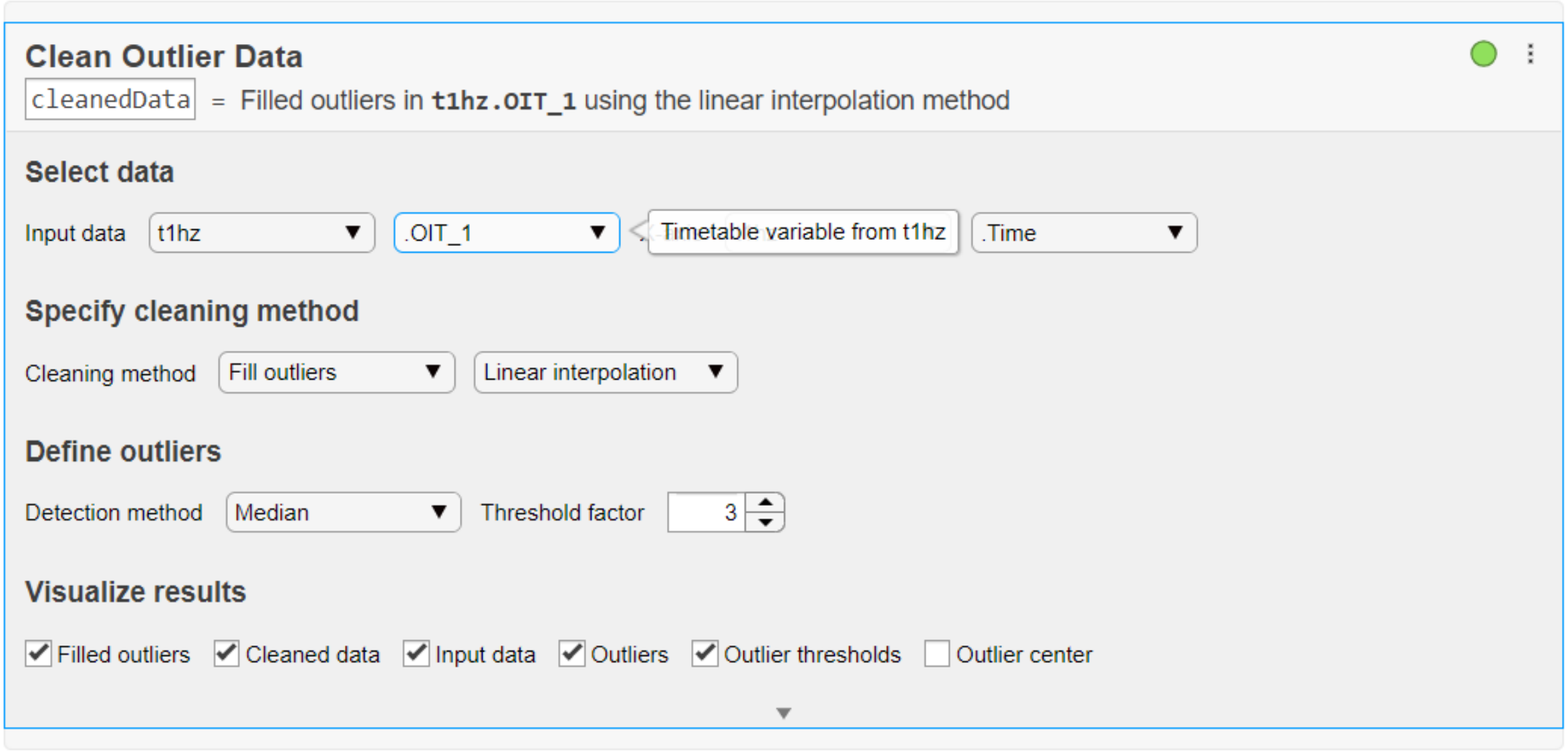
Getting Started: **Exploring a New Dataset**



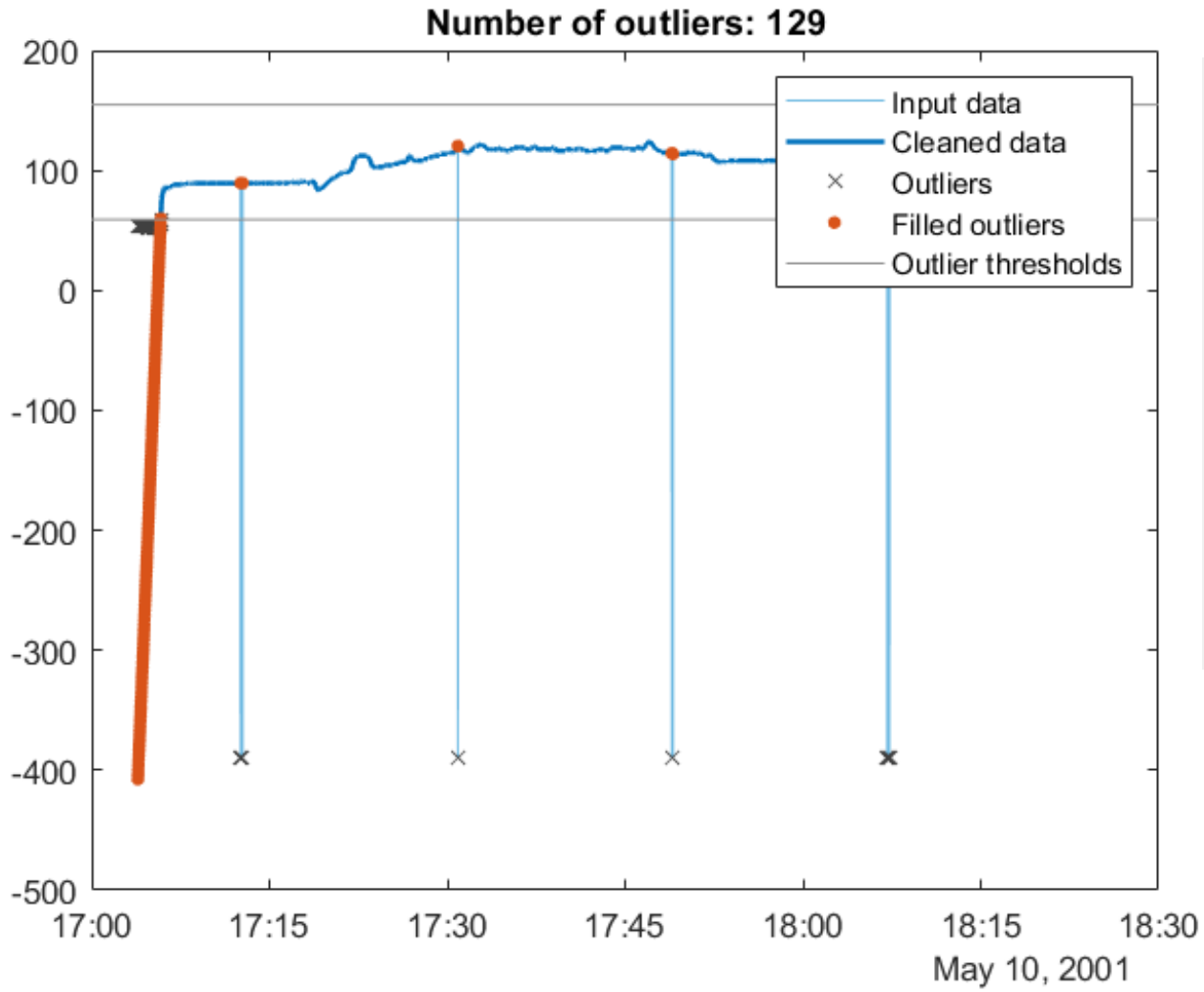
Getting Started: **Exploring a New Dataset**



Getting Started: **Exploring a New Dataset**

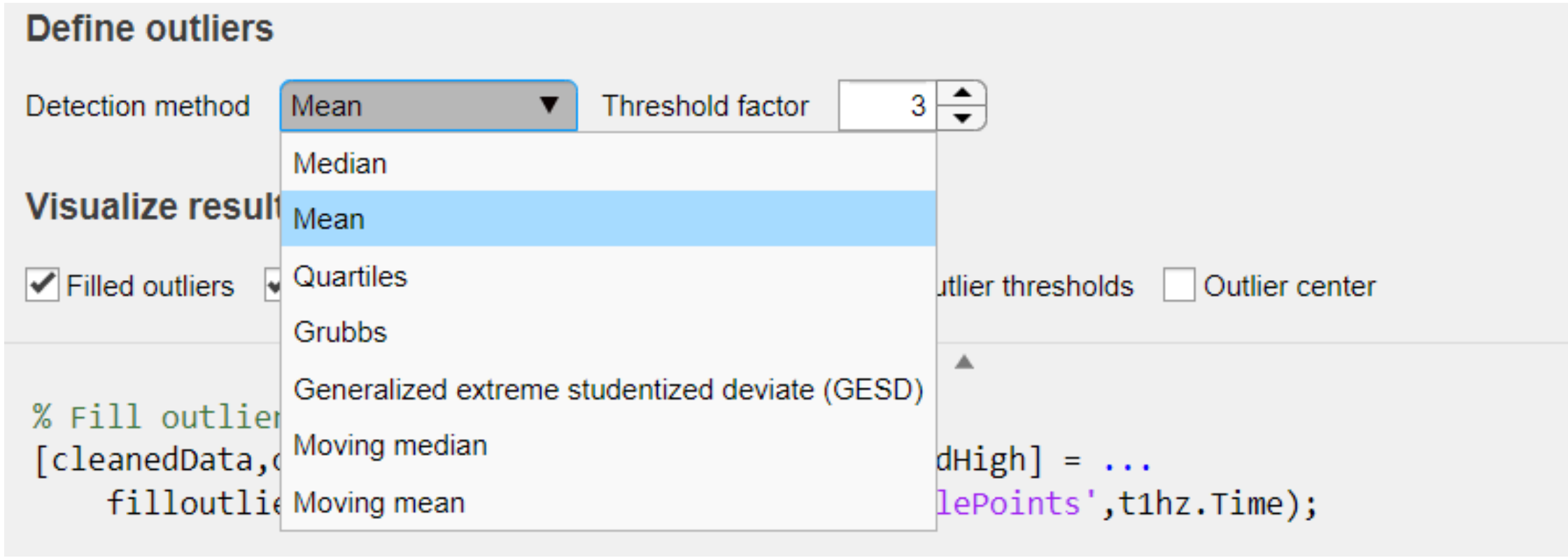


Getting Started: **Exploring a New Dataset**



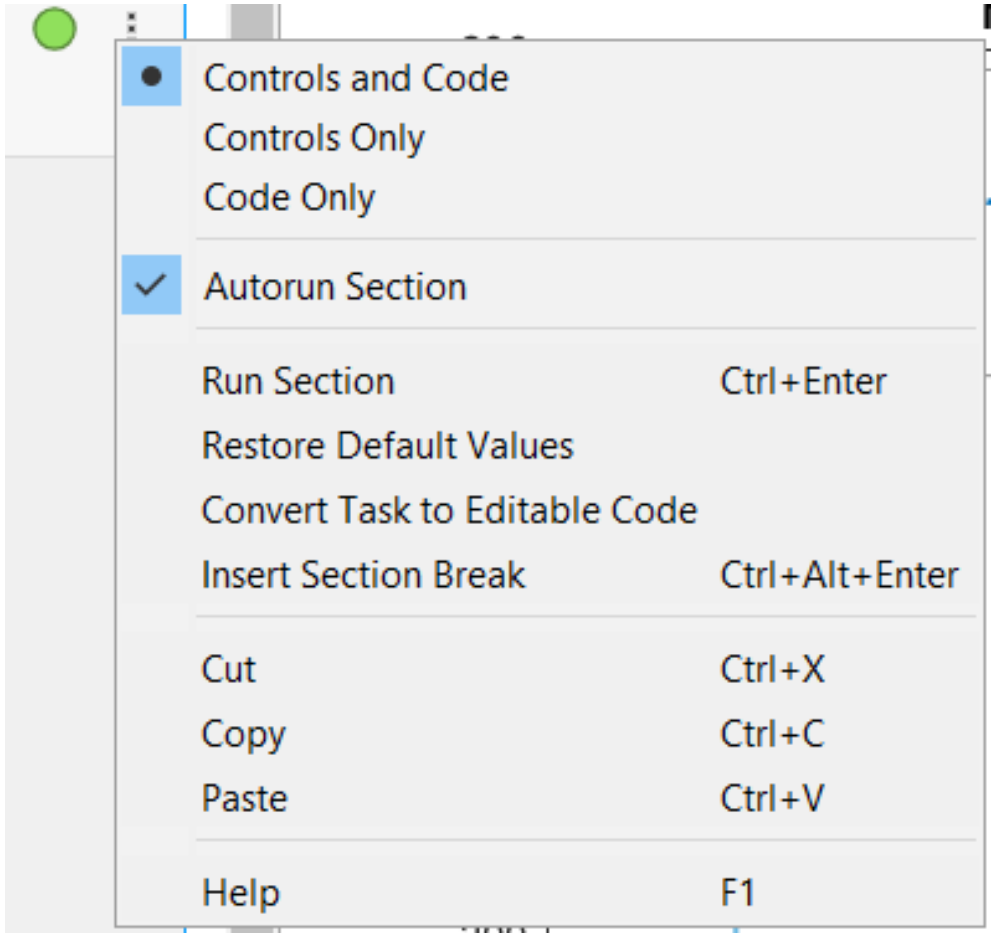

```
% Visualize results
clf
plot(t1hz.Time,t1hz.OIT_1,'Color',[109 185 226]/255,...
    'DisplayName','Input data')
hold on
plot(t1hz.Time,cleanedData,'Color',[0 114 189]/255,'LineWidth',1.5,...
    'DisplayName','Cleaned data')

% Plot outliers
plot(t1hz.Time(outlierIndices),t1hz.OIT_1(outlierIndices),'x',...
    'Color',[64 64 64]/255,'DisplayName','Outliers')
title(['Number of outliers: ' num2str(nnz(outlierIndices))])

% Plot filled outliers
plot(t1hz.Time(outlierIndices),cleanedData(outlierIndices),'.','MarkerSize',12,...
    'Color',[217 83 25]/255,'DisplayName','Filled outliers')

% Plot outlier thresholds
plot([xlim missing xlim],[thresholdLow*[1 1] NaN thresholdHigh*[1 1]],...
    'Color',[145 145 145]/255,'DisplayName','Outlier thresholds')
```

Getting Started: **Exploring a New Dataset**




```
% Fill outliers
[cleanedData,outlierIndices,thresholdLow,thresholdHigh] = ...
    filloutliers(t1hz.OIT_1,'linear','mean','SamplePoints',t1hz.Time);
```

Getting Started: **Exploring a New Dataset**



Missing Data

`ismissing`
`rmmissing`
`fillmissing`

Outliers

`isoutlier`
`rmoutliers`
`filloutliers`

Change Points

`ischange`

Noisy Data

`smoothdata`

and more…

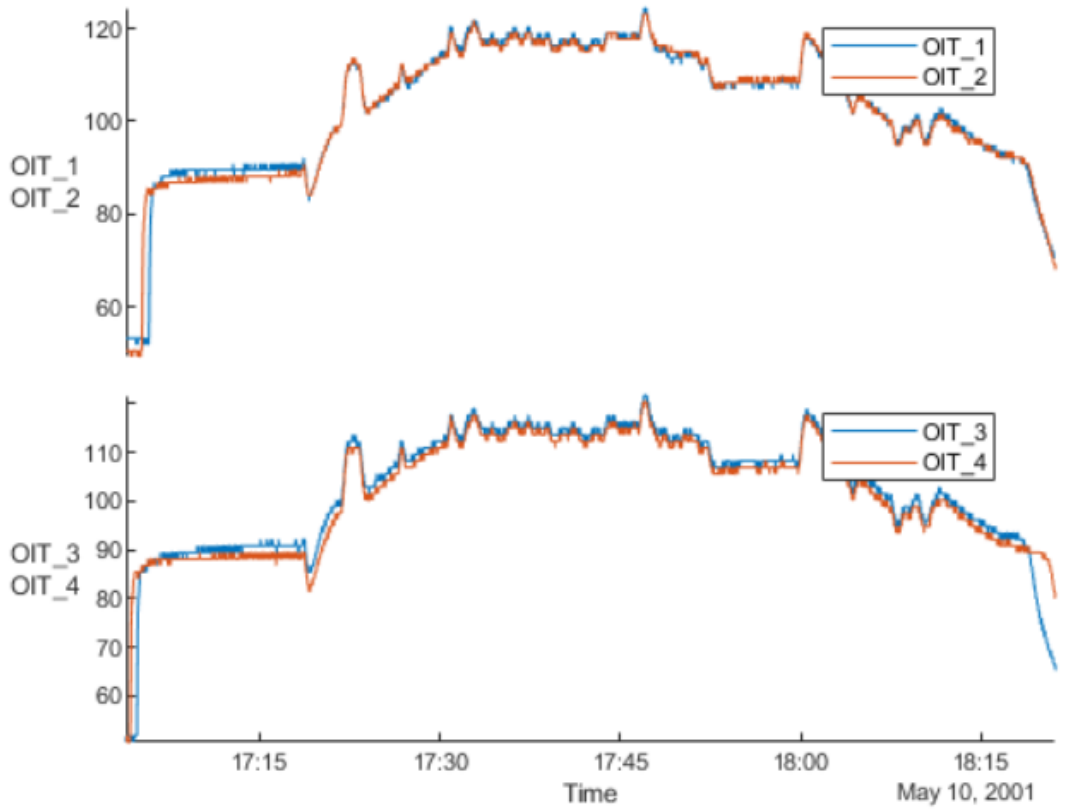
https://www.mathworks.com/help/matlab/preprocessing-data.html

Getting Started: **Exploring a New Dataset**

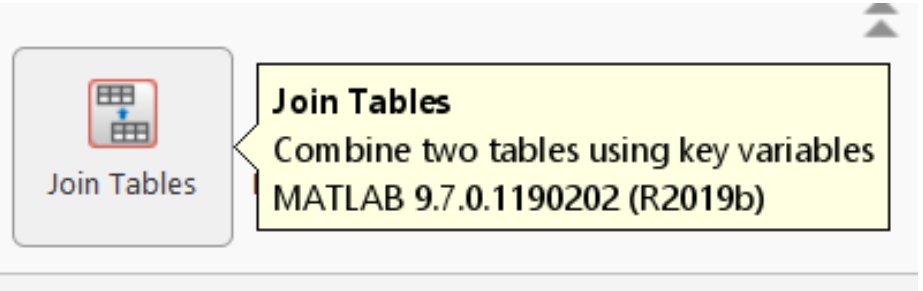
Apply outlier filling method to all oil temperature variables in our timetable.

```
t1hz = filloutliers(t1hz,'linear','mean','DataVariables',{'OIT_1','OIT_2','OIT_3','OIT_4'});
```

```
stackedplot(t1hz,{{'OIT_1','OIT_2'},{'OIT_3','OIT_4'}});
```



Getting Started: **Exploring a New Dataset**

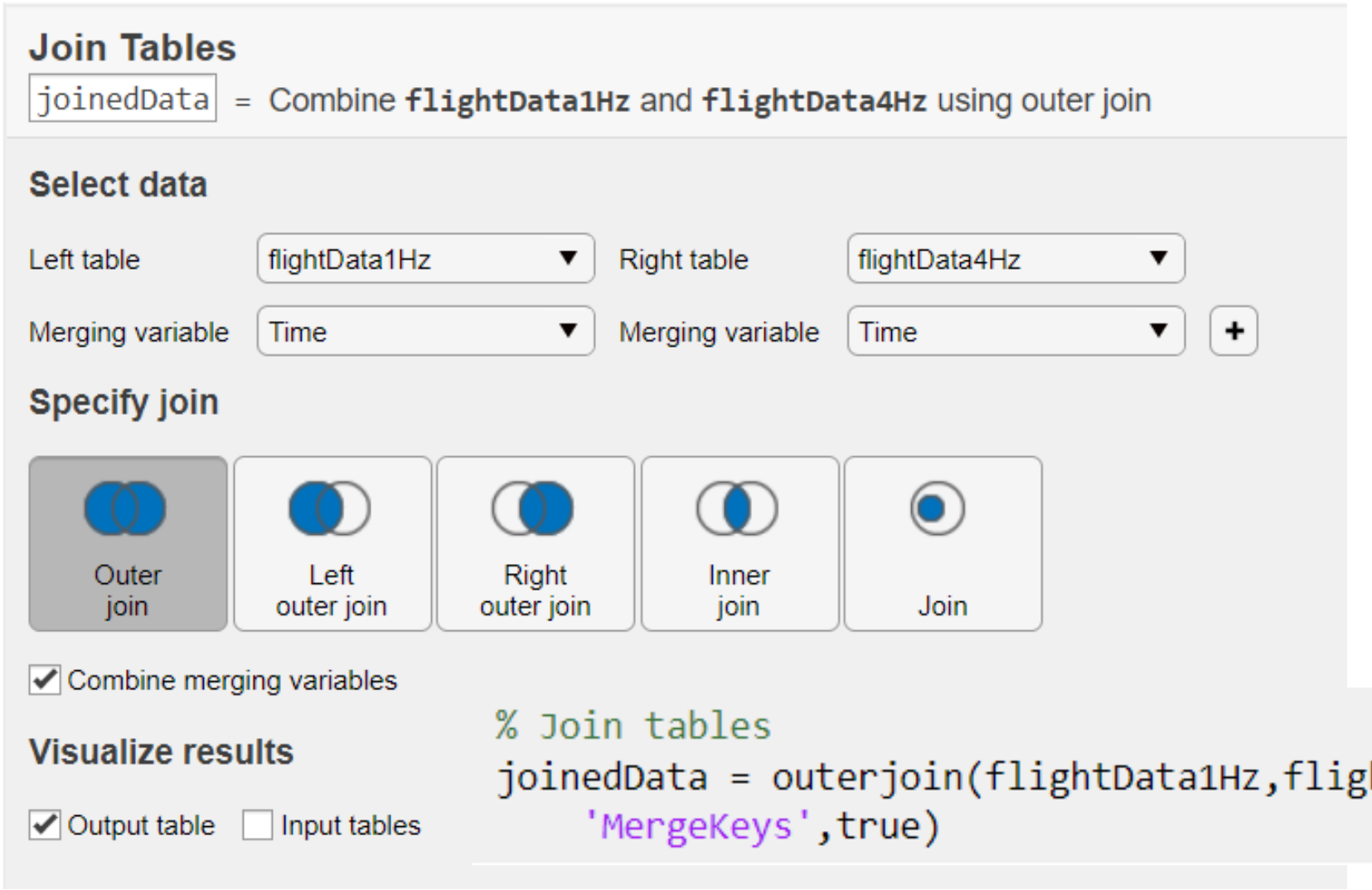



```
t4hz = table2timetable(flightData4Hz)
```

```
t4hz = 18544×49 timetable
```

	Time	ALT	ALTR	AOA1
1	10-May-20…	861	16	6.5918
2	10-May-2001 17:03:54.250	61	-16	6.5918
3	10-May-20…	861	-16	6.5918
4	10-May-20…	861	-16	6.5918
5	10-May-20…	861	0	6.5918
6	10-May-20…	861	0	6.5918
7	10-May-20…	861	-16	6.5918
8	10-May-20…	861	0	6.5918
9	10-May-20…	861	0	6.5918

Getting Started: **Exploring a New Dataset**




```
% Join tables
joinedData = outerjoin(flightData1Hz,flightData4Hz,'Keys','Time',...
    'MergeKeys',true)
```

Getting Started: **Exploring a New Dataset**

Synchronize 1Hz and 4Hz Data

Join the 1Hz and 4Hz data using the synchronize command.

The sychronize command gives us flexibility in how the synchronize occurs.

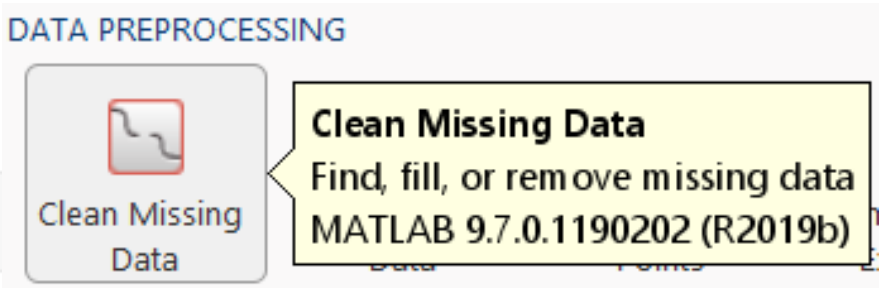
Here, we use the default synchronize method which synchronizes the data onto a union of the timestamps from t1hz and t4hz.

```
t = synchronize(t1hz,t4hz)
```

t = 18544×137 timetable

	Time	ABRK	ACMT	AIL_1	AIL_2	ALTS	APFD	ATEN	A_
1	10-May-20…	119.9836	59	91.8789	91.5925	6000	2	0	
2	10-May-20…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	10-May-20…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	10-May-20…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	10-May-20…	119.9836	59	91.8994	91.5925	6000	2	0	
6	10-May-20…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	10-May-20…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
8	10-May-20…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9	10-May-20…	119.9836	60	91.8994	91.5925	6000	2	0	

Getting Started: **Exploring a New Dataset**



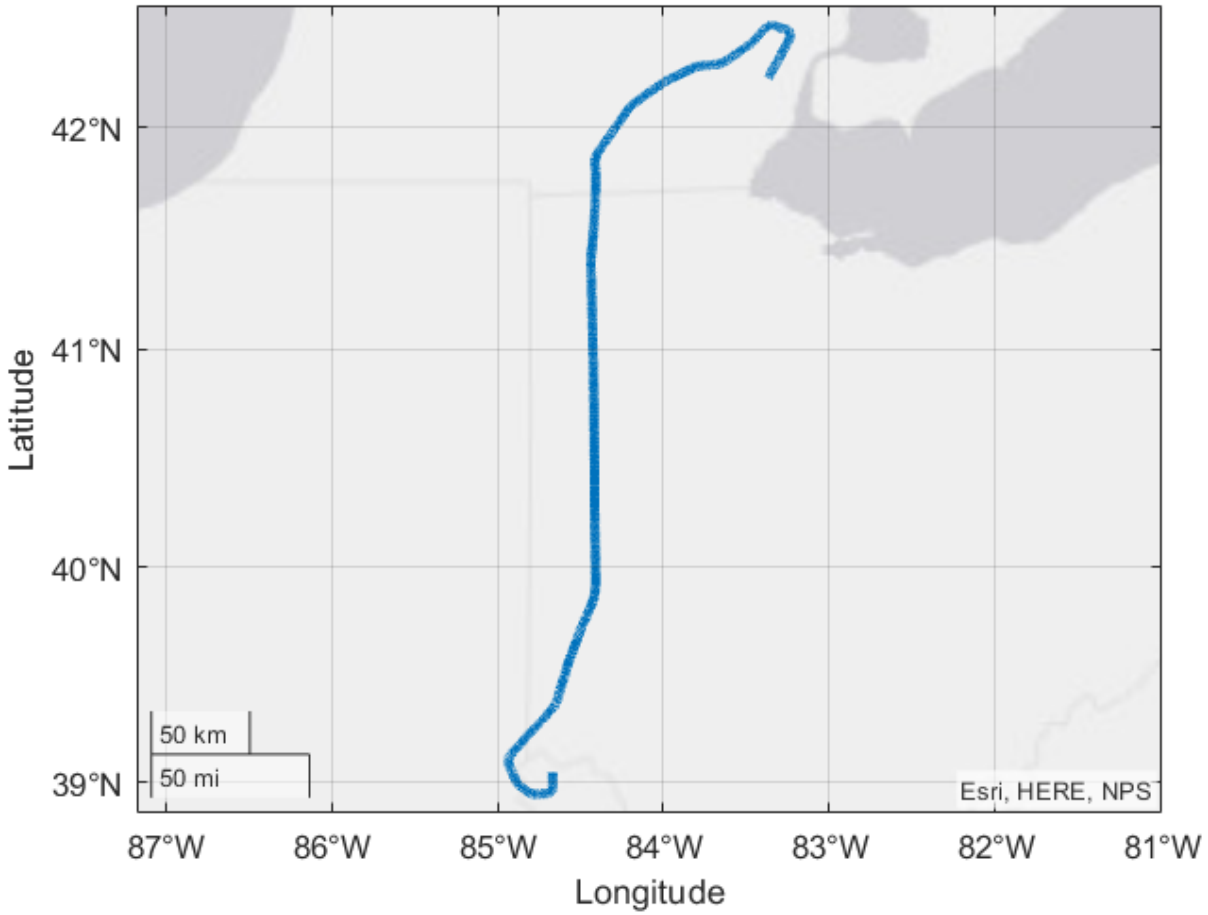

```
t = fillmissing(t,'previous')
```

t = 18544×137 timetable

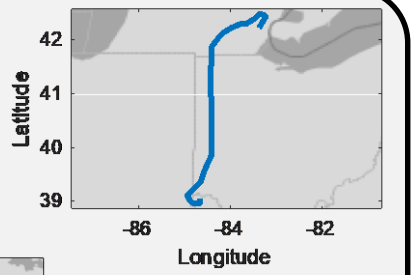
	Time	ABRK	ACMT	AIL_1	AIL_2
1	10-May-20...	119.9836	59	91.8789	91.5925
2	10-May-20...	119.9836	59	91.8789	91.5925
3	10-May-20...	119.9836	59	91.8789	91.5925
4	10-May-20...	119.9836	59	91.8789	91.5925
5	10-May-20...	119.9836	59	91.8994	91.5925
6	10-May-20...	119.9836	59	91.8994	91.5925
7	10-May-20...	119.9836	59	91.8994	91.5925
8	10-May-20...	119.9836	59	91.8994	91.5925
9	10-May-20...	119.9836	60	91.8994	91.5925

Getting Started: **Exploring a New Dataset**

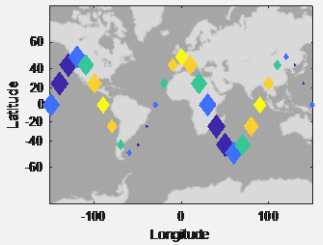
```
inFlight = t.WOW==1;
geoplot(t.LATP(inFlight),t.LONP(inFlight),'LineWidth',3);
```



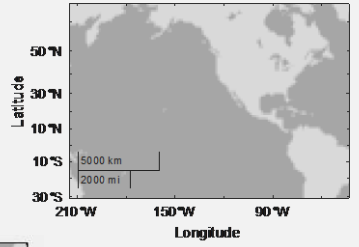
geoplot



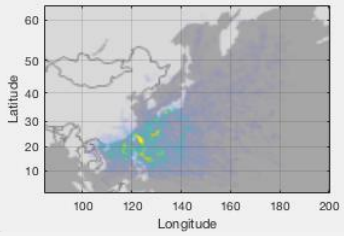
geoscatter



geobubble

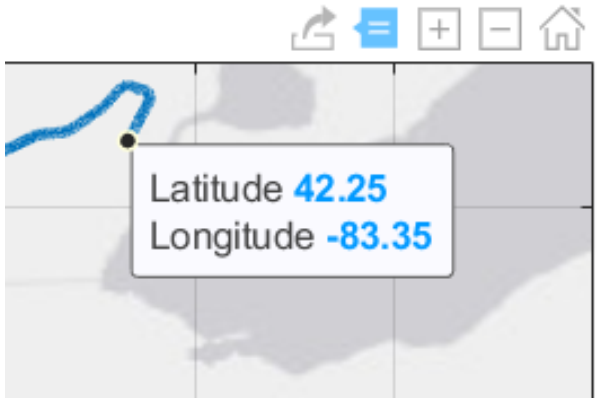


geodensityplot



https://www.mathworks.com/help/matlab/geographic-plots.html

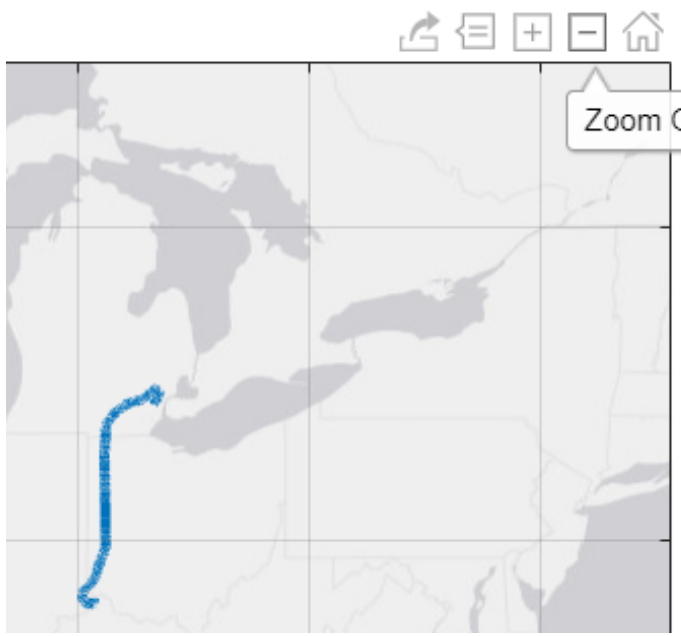
Getting Started: **Exploring a New Dataset**



Code

```
ax = gca;
chart = ax.Children(1);
datatip(chart,42.25,-83.35);
```

Update Code Copy



Code

```
geolimits([33.2 47.5],[-96.0 -72.2])
```

Update Code Copy

Data Science Maturity Levels

- **Explore and understand data**
- **Document analysis**
- **Tools will be re-used in next steps**

Data Science Maturity Levels

- **Apply to different datasets**
  - **Functions/Scripts**
  - **MATLAB Apps**
- **Trend: Work with BIG DATA**

Reusable Tools for Big Data Analysis

1. Custom datastore for reading all the data
2. Extension of the datastore for specific queries
3. Extending datastore with tall variables

```
ds = flightDataStore('\Data\666');
ds.SelectedVariableNames = {'FIRE_2'};
ds.AddFlightIdentifier = true;
```

```
while hasdata(ds)
    t = read(ds);
    if any(t.FIRE_2)
        results = [results; {t.Flight(1), sum(t.FIRE_2)}];
    end
end
```

```
t = tall(ds)
```

```
t =

  M×6 tall timetable
```

```
binEdges = 0:80;
histogram(t.FE,binEdges);
```

Big Data Requires New Tools

Create a datastore from all CSV files

```
ds = datastore('*.csv')
```

Read a single file of data

```
data = read(ds);
```

Reset the datastore back to the first file

```
reset(ds);
```

Find the maximum value of “Y” in each file

```
X = [];
while hasdata(ds)
        data = read(ds);
        X(end+1) = max(data.Y);
end
```

Built-In Datastores	
General	`datastore`
	`spreadsheetDatastore`
	`tabularTextDatastore`
	`fileDatastore`
Database	`databaseDatastore`
Image	`imageDatastore`
	`denoisingImageDatastore`
	`randomPatchExtractionDatastore`
	`pixelLabelDatastore`
	`augmentedImageDatastore`
Audio	`audioDatastore`
Predictive Maintenance	`fileEnsembleDatastore`
	`simulationEnsembleDatastore`
Simulink	`SimulationDatastore`
Automotive	`mdfDatastore`

Big Data Requires New Tools

Custom Datastore

- Customize a datastore to work with your dataset
- Gives you control over how data is loaded and formatted
- MATLAB subclass: “fill-in-the-blanks”
- Build a piece of infrastructure, then re-use it in your analyses

```
function [data,info] = read(ds)
    ...
end
```

```
function tf = hasdata(ds)
    ...
end
```

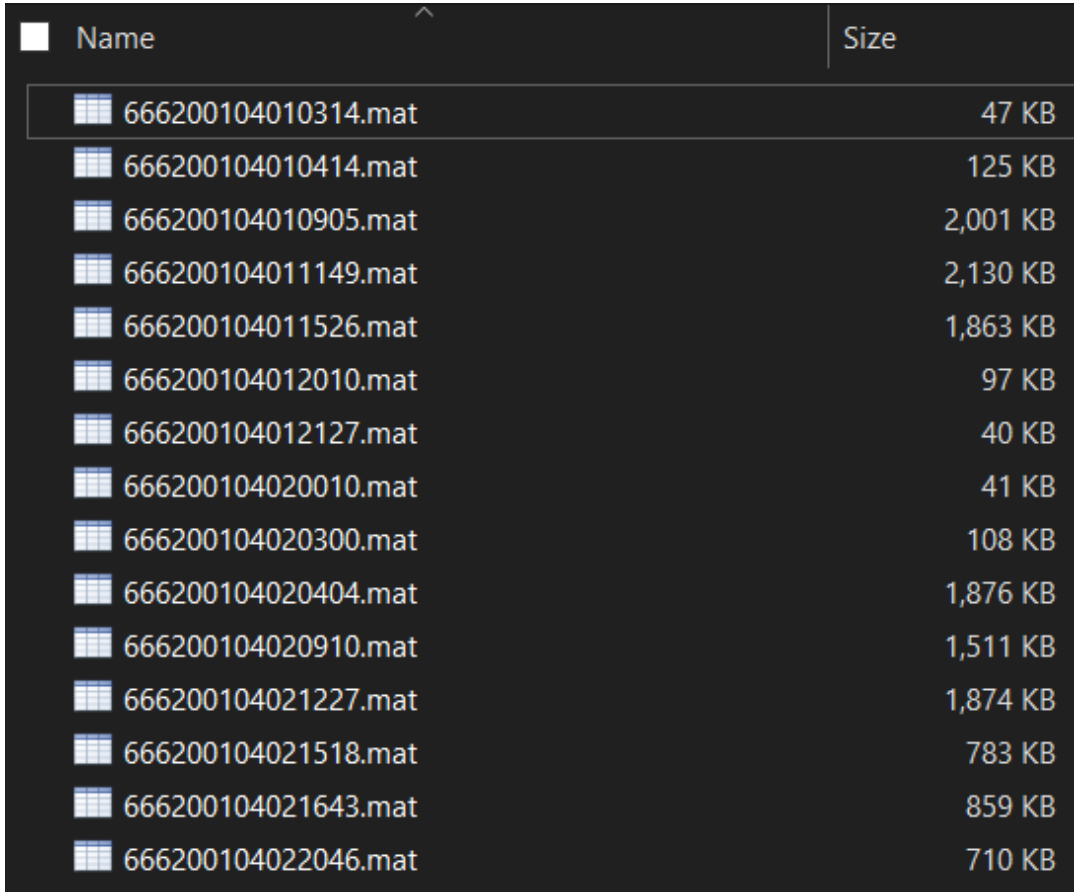
```
function reset(ds)
    ...
end
```

```
function p = progress(ds)
    ...
end
```

```
function data = readall(ds)
    ...
end
```

A Custom Datastore for Flight Data

```
ds = flightDataStore('\Data\666');
ds.SelectedVariableNames = {'FIRE_2'};
ds.AddFlightIdentifier = true;
```




```
ds =
  flightDataStore with properties:

          CurrentFileIndex: 1
             NumberOfFiles: 6691
             VariableNames: {1×186 cell}
     SelectedVariableNames: {'FIRE_2'}
       AddFlightIdentifier: 1
```

```
preview(ds)
```

ans = 8×2 timetable

	Time	FIRE_2	Flight
1	01-Apr-200…	0	666200104…
2	01-Apr-200…	0	666200104…
3	01-Apr-200…	0	666200104…
4	01-Apr-200…	0	666200104…
5	01-Apr-200…	0	666200104…
6	01-Apr-200…	0	666200104…
7	01-Apr-200…	0	666200104…
8	01-Apr-200…	0	666200104…

A Custom Datastore for Flight Data

```
t = readFullFlight(ds,'666200303212002')
```

t = 44032×187 timetable

	Time	ACID	DATE_DAY	DATE_MONTH
1	21-Mar-200…	666	21	3
2	21-Mar-200…			
3	21-Mar-200…			
4	21-Mar-200…			
5	21-Mar-200…			
6	21-Mar-200…			
7	21-Mar-200…			
8	21-Mar-200…			
9	21-Mar-200…			

```
signalMetaInfo(ds)
```

ans = 186×5 table

	Signals	Rate	Units	Description	Alpha
1	'ABRK'	1.0000	'DEG'	'AIRBRAKE PO…	'ABRK'
2	'ACID'	0.2500	''	'AIRCRAFT NU…	'ACID'
3	'ACMT'	1.0000	''	'ACMS TIMING…	'ACMT'
4	'AIL_1'	1.0000	'DEG'	'AILERON POS…	'AIL.1'
5	'AIL_2'	1.0000	'DEG'	'AILERON POS…	'AIL.2'
6	'ALT'	4.0000	'FEET'	'PRESSURE A…	'ALT'
7	'ALTR'	4.0000	'FT/MIN'	'ALTITUDE RA…	'ALTR'
8	'ALTS'	1.0000	'FEET'	'SELECTED AL…	'ALTS'
9	'AOA1'	4.0000	'DEG'	'ANGLE OF AT…	'AOA1'

A Custom Datastore for Flight Data

Read a file from the Dash Link Flight Data

https://c3.nasa.gov/dashlink/projects/85/

```
function tt = readFlightFile(filename,varnames,addidentifier)

% Load the data
if nargin==1
    % Load the entire file
    s = load(filename);
elseif nargin>=2
    % Load only the specified variables
    if ~isempty(varnames)
        s = load(filename,varnames{:});
    else
        s = struct([]);
    end
end
```

ExtractStartTime

Extract the flight start time from the flight identifier (located in the flight filename). The flight identifier is of the form:

NNNyyyyMMddhhmm

Where NNN is the tail number, and yyyyMMddhhmm is a timestamp.

Note: Some files have a trailing "_#" (in the case where there are multiple files from the same aircraft that started at the same minute).

```
function st = extractStartTime(s,flightIdentifier)

    if contains(flightIdentifier,'_') % Handle files that have an underscore and numeric identifier
        flightIdentifier = extractBefore(flightIdentifier,'_');
    end

    % The starting second is not in the flight identifier, but we can get it from the raw data
    sec = s.GMT_SEC.data(1);

    % Add the seconds and the timestamp together to get the start time
    st = datetime(flightIdentifier(4:end),...
        'InputFormat','yyyyMMddHHmm',...
        'TimeZone','Europe/London') ...
        + seconds(sec);
end
```

A Custom Datastore for Flight Data

```
classdef flightDataStore < matlab.io.Datastore & ...
        matlab.io.datastore.Partitionable & ...
        matlab.io.datastore.HadoopFileBased

    properties (SetAccess = protected)
        CurrentFileIndex double
        NumberOfFiles double
        VariableNames cell
    end
    properties (Dependent)
        SelectedVariableNames cell
    end
    properties
        AddFlightIdentifier logical
    end
    properties (Access = protected, Hidden = true)
        FileSet matlab.io.datastore.DsFileSet
    end
```

```
function fds = flightDataStore(location)
    %FLIGHTDATASTORE Datastore for reading flight data
    %   DS = FLIGHTDATASTORE(LOCATION) creates a flightDataStore DS
    %   of the data. LOCATION is a folder that contains .MAT files
    %   that contain flight data.  Alternatively, LOCATION can be
    %   an individual .MAT file of flight data.  If LOCATION is a
    %   folder, .MAT files are searched for recursively (there can
    %   be multiple levels of folders).
    %

    % Use the DsFileSet class to find and keep track of files
    fds.FileSet = matlab.io.datastore.DsFileSet(location,...
        'FileExtensions','.mat', ...
        'FileSplitSize','file',...
        'IncludeSubfolders',true);
    fds.CurrentFileIndex = 1;
    fds.AddFlightIdentifier = false;

    fds.NumberOfFiles = fds.FileSet.NumFiles;
    fds.VariableNames = extractVariableNames(fds);
    fds.SelectedVariableNames = fds.VariableNames;

    reset(fds);
end
```

A Custom Datastore for Flight Data

```
%% Basic datastore implementation
function [data,info] = read(fds)
    % Read data and information about the extracted data.

    % Info about the next file to read
    fileInfoTbl = nextfile(fds.FileSet);

    % Call the file reader
    data = readFlightFile(fileInfoTbl.FileName,fds.SelectedVaria

    % Return info about the data that has been read
    info.Size = size(data);
    info.FileName = fileInfoTbl.FileName;
    info.Offset = fileInfoTbl.Offset;

    % Update CurrentFileIndex for tracking progress
    fds.CurrentFileIndex = fds.CurrentFileIndex + 1 ;

end
```

```
%% Basic datastore implementation
function [data,info] = read(fds)...

function tf = hasdata(fds)...

function reset(fds)...

function frac = progress(fds)...

function data = readall(fds,iterDisp)...
```

```
%% Custom methods for reading individual flight data
function data = readFullFlight(fds,flightIdentifier)...

function data = readFlight(fds,flightIdentifier)...
```

```
%% Extract metadata that is helpful to understand the dataset
function signalInfo = signalMetaInfo(fds)...
```

```
%% Support for parallel computing
function subds = partition(fds,n,i)...
```

```
%% Support for Hadoop
function initializeDatastore(fds,hadoopInfo)...

function loc = getLocation(fds)...
```

Find Rare Events, then Dive Deeper

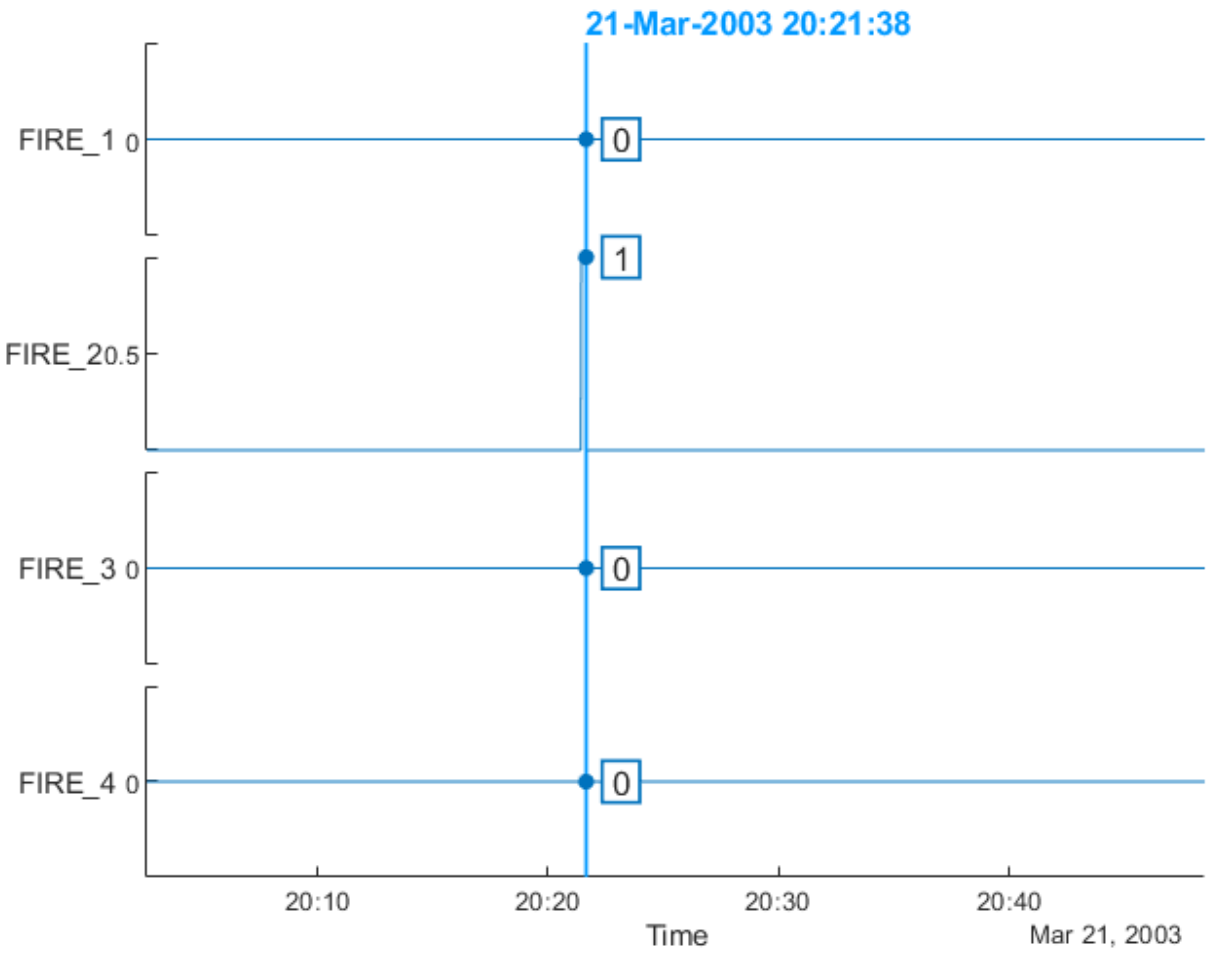
```
results = table('Size',[0 2],'VariableTypes',{'categorical','double'},...
    'VariableNames',{'Flight','Fire_Seconds'});
while hasdata(ds)
    t = read(ds);
    if any(t.FIRE_2)
        results = [results; {t.Flight(1), sum(t.FIRE_2)}];
    end
end
```

results = 38×2 table

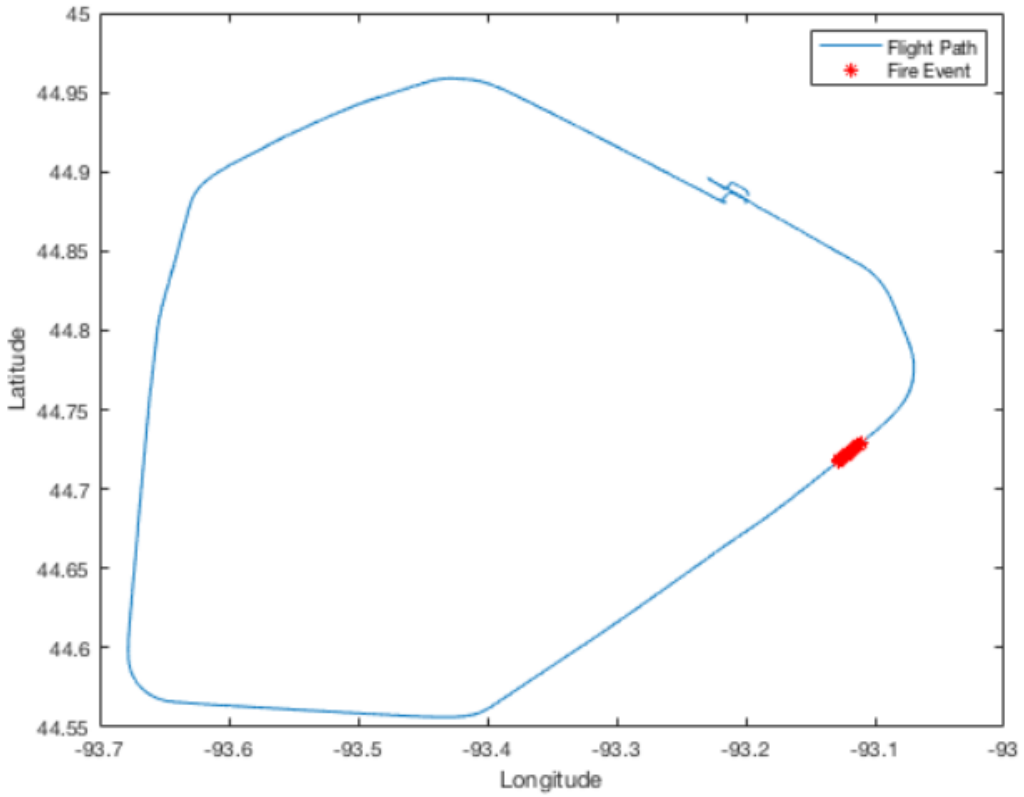
	Flight	Fire_Seconds
21	666200210...	1
22	666200210...	1
23	666200302...	3
24	666200303...	1
25	666200303212002	15
26	666200303...	2
27	666200303...	1
28	666200304...	1
29	666200304...	2

Find Rare Events, then Dive Deeper

```
stackedplot(t,contains(t.Properties.VariableNames,'FIRE_'));
```

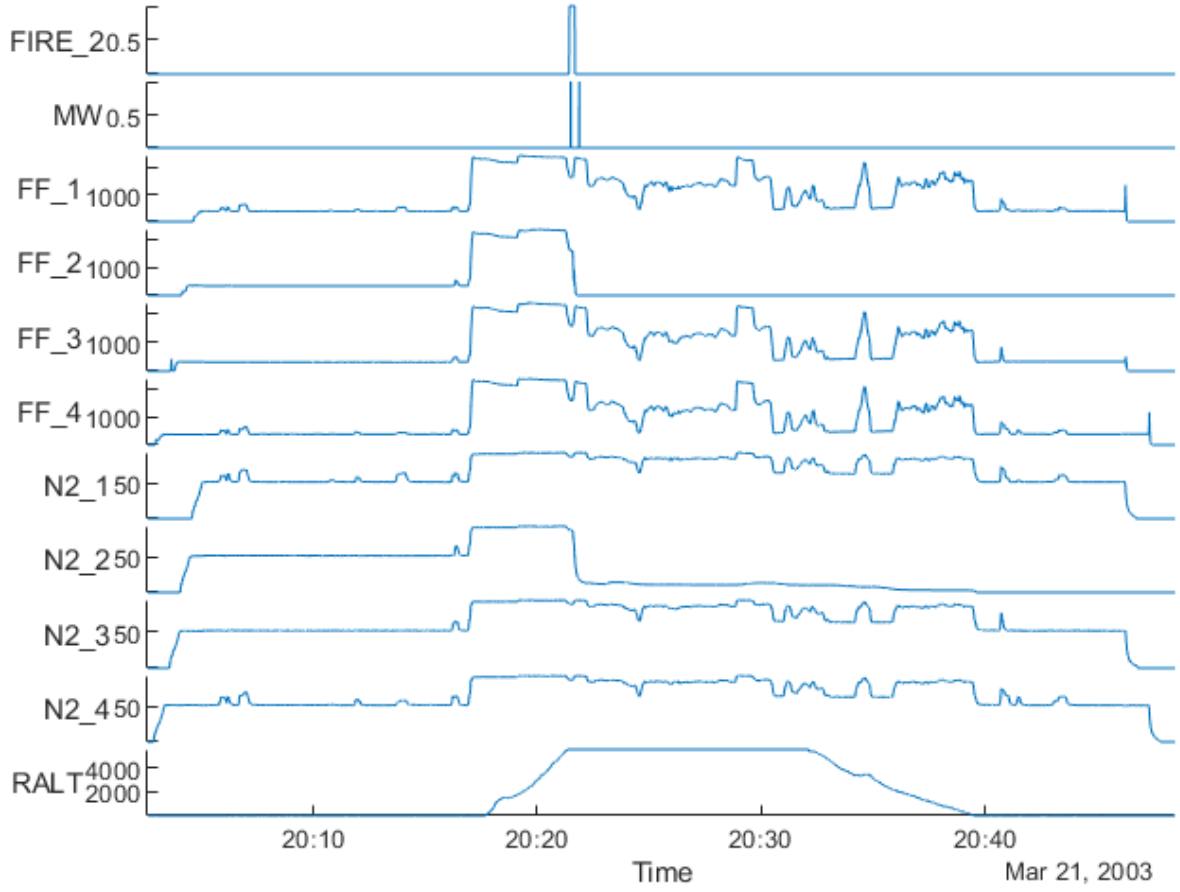


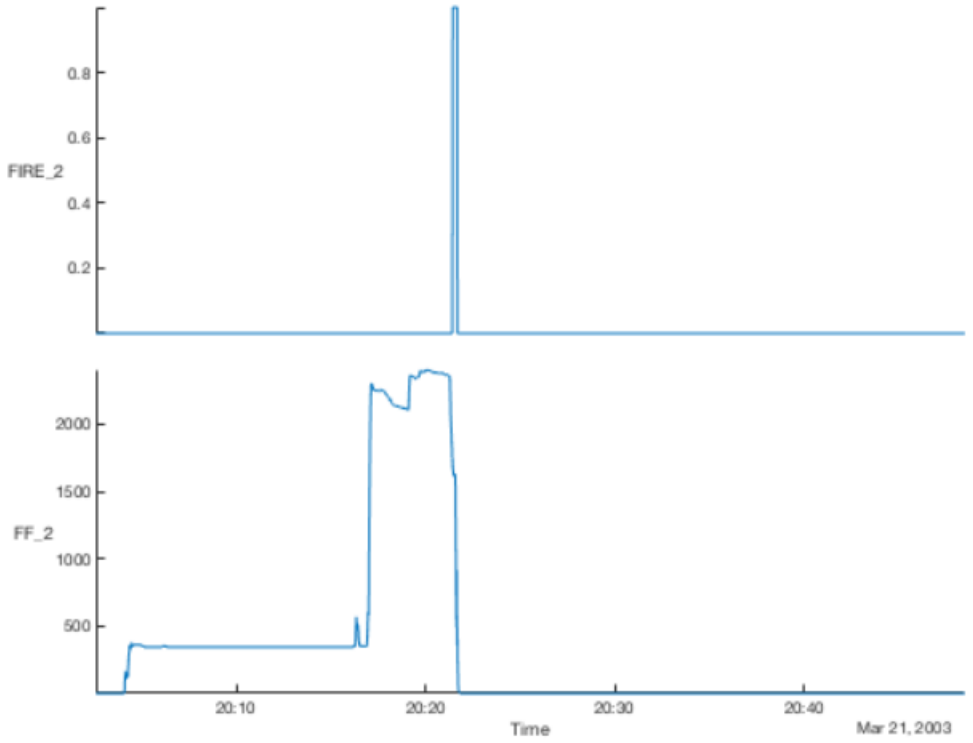

```
plot(t.LONP,t.LATP)
plot(t.LONP(logical(t.FIRE_2)),t.LATP(logical(t.FIRE_2)),'*r')
xlabel('Longitude')
ylabel('Latitude')
legend('Flight Path','Fire Event')
```



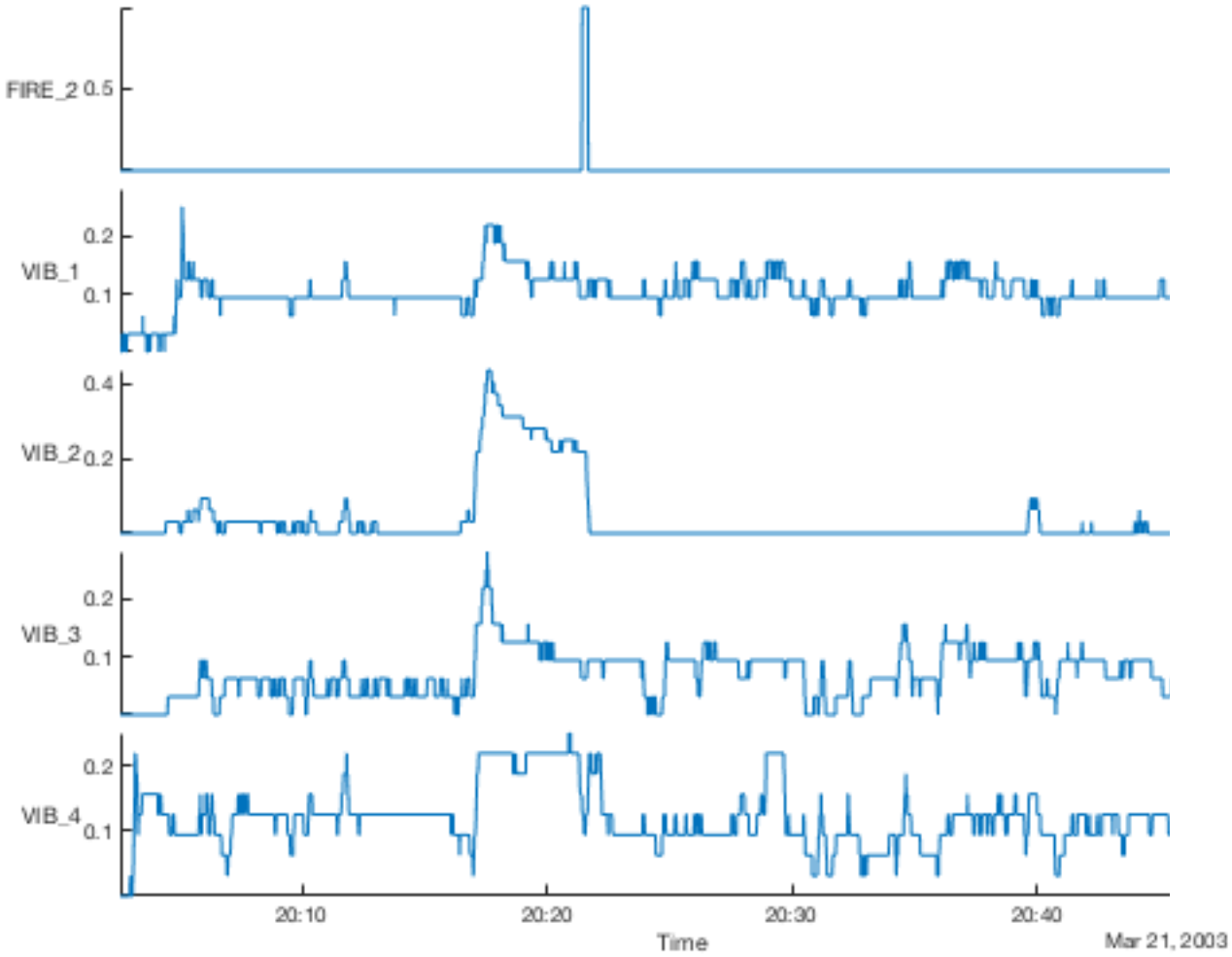
Find Rare Events, then Dive Deeper

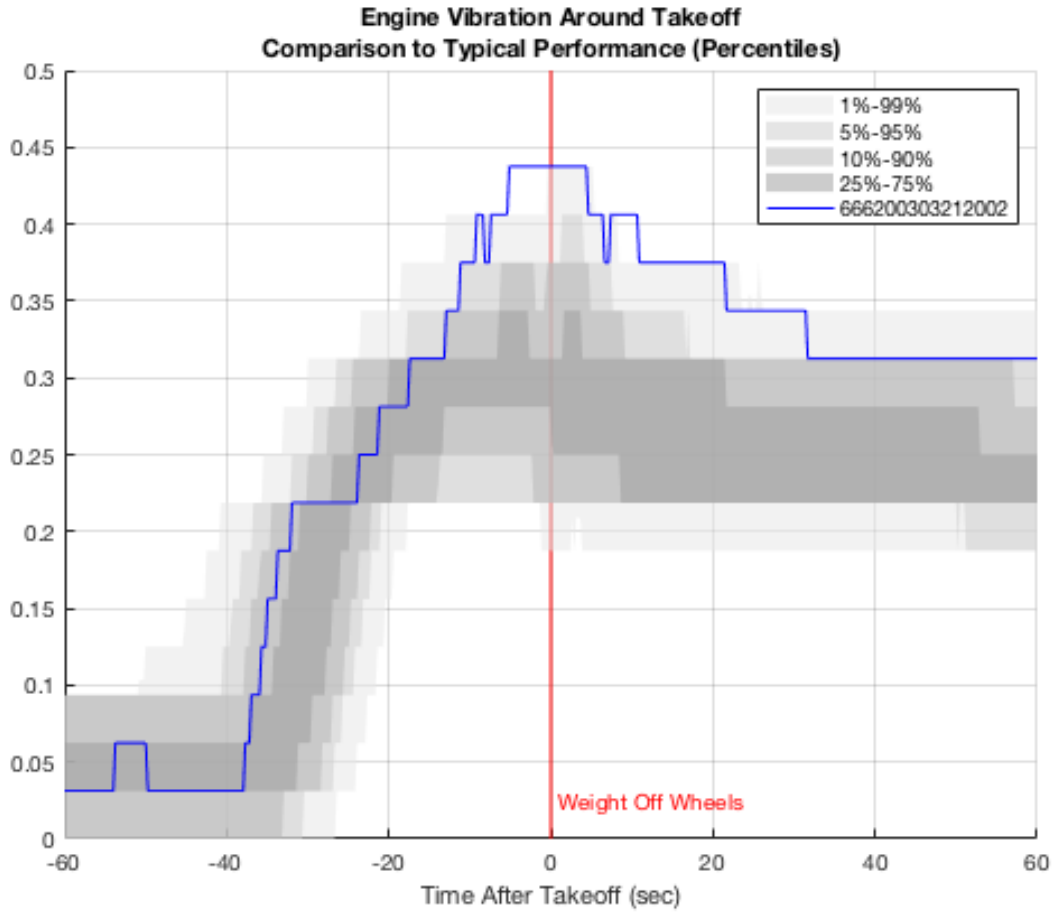
```
varidx = contains(t.Properties.VariableNames,{'FIRE_2','RALT','FF','N2','MW'});
stackedplot(t,varidx);
```





Find Rare Events, then Dive Deeper



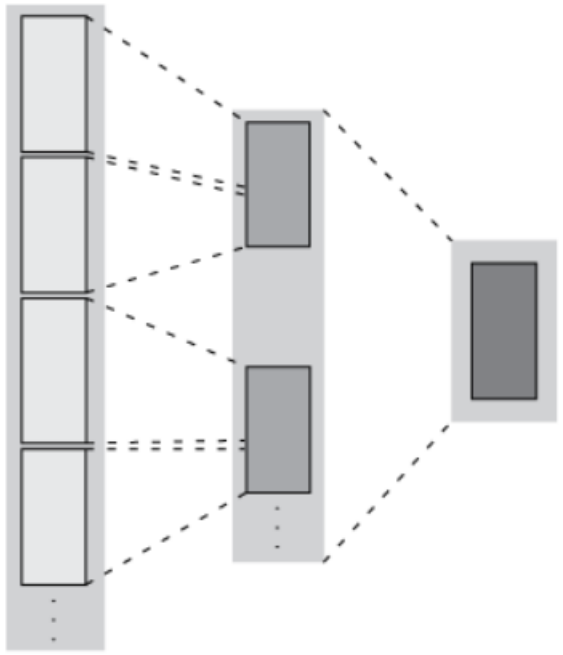


Big Data Requires New Tools

Tall Arrays

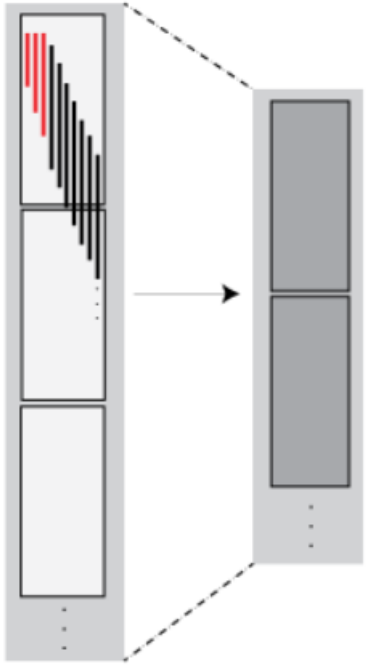
```
t = tall(ds)
```

```
t =

  M×6 tall timetable
```



- When to use tall
  - the function has already been implemented
  - the output of the operation does not fit in memory
  - multiple independent outputs are required
  - the algorithm works over a moving window

```
binEdges = 0:80;
histogram(t.FE,binEdges);
```



Perform Fleet-Wide Calculations

TAS: True Airspeed (knots)

FF_N: Fuel Flow N (lbs/hr)

WOW: Weight On Wheels (logical)

```
ds.SelectedVariableNames = {'TAS','FF_1','FF_2','FF_3','FF_4','WOW'}
```

```
t = tall(ds)
```

```
t =

  M×6 tall timetable
```

Time	WOW	TAS	FF_1	FF_2	FF_3	FF_4
01-Apr-2001 03:15:30	0	0	0	0	0	0
01-Apr-2001 03:15:30	NaN	0	0	0	0	0
01-Apr-2001 03:15:30	NaN	0	0	0	0	0
01-Apr-2001 03:15:30	NaN	0	0	0	0	0
01-Apr-2001 03:15:31	0	0	0	0	0	0
01-Apr-2001 03:15:31	NaN	0	0	0	0	0
01-Apr-2001 03:15:31	NaN	0	0	0	0	0
01-Apr-2001 03:15:31	NaN	0	0	0	0	0
:	:	:	:	:	:	:
:	:	:	:	:	:	:

Limit to only cases where the aircraft was in the air

```
inAir = t.WOW==1;
t = t(inAir,:);
```

Sum the total fuel flow to the engine (units are LBS/HR)

```
t.FF_TOTAL = t.FF_1+t.FF_2+t.FF_3+t.FF_4;
```

Convert airspeed to MPH and calculate fuel efficiency in LBS/MILE

```
knots2mph = 1.15078;
t.FE = t.FF_TOTAL ./ (t.TAS*knots2mph);
t.FE(isinf(t.FE)) = 0;
```

Time	WOW	TAS	FF_1	FF_2	FF_3	FF_4	FF_TOTAL	FE
12-Jan-2001 09:22:28	1	147.81	2600	2632	2600	2592	10424	61.282
12-Jan-2001 09:22:29	1	149.62	2608	2624	2600	2592	10424	60.539
12-Jan-2001 09:22:30	1	149.62	2600	2624	2608	2592	10424	60.539
12-Jan-2001 09:22:31	1	149.88	2600	2616	2600	2592	10408	60.346
12-Jan-2001 09:22:32	1	148.94	2592	2616	2600	2584	10392	60.632
12-Jan-2001 09:22:33	1	148.62	2592	2616	2600	2584	10392	60.76
12-Jan-2001 09:22:34	1	147.94	2592	2608	2592	2584	10376	60.948
12-Jan-2001 09:22:35	1	148.62	2584	2608	2592	2576	10360	60.573
:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:

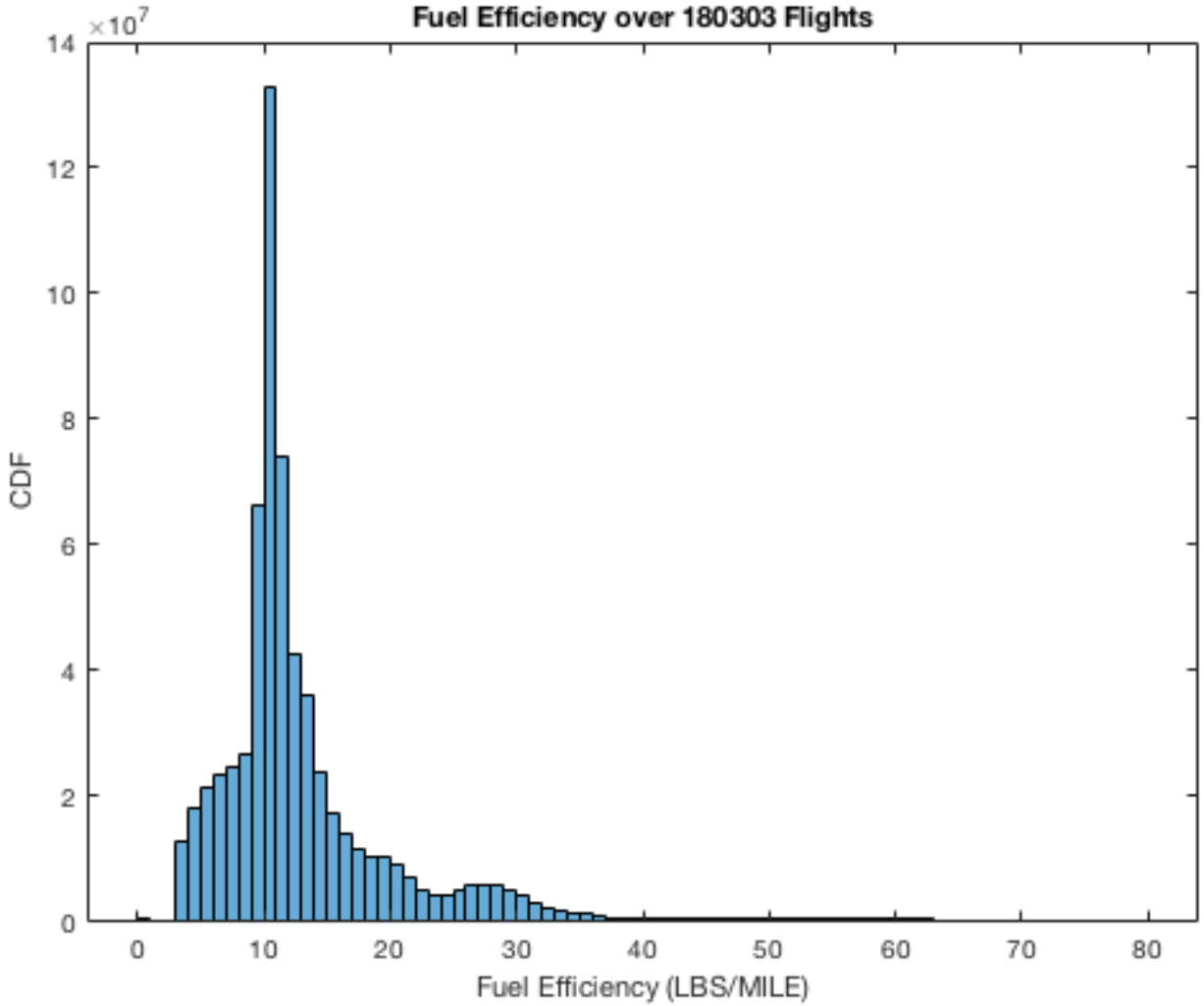
Perform Fleet-Wide Calculations

Plot Data Using Tall Arrays

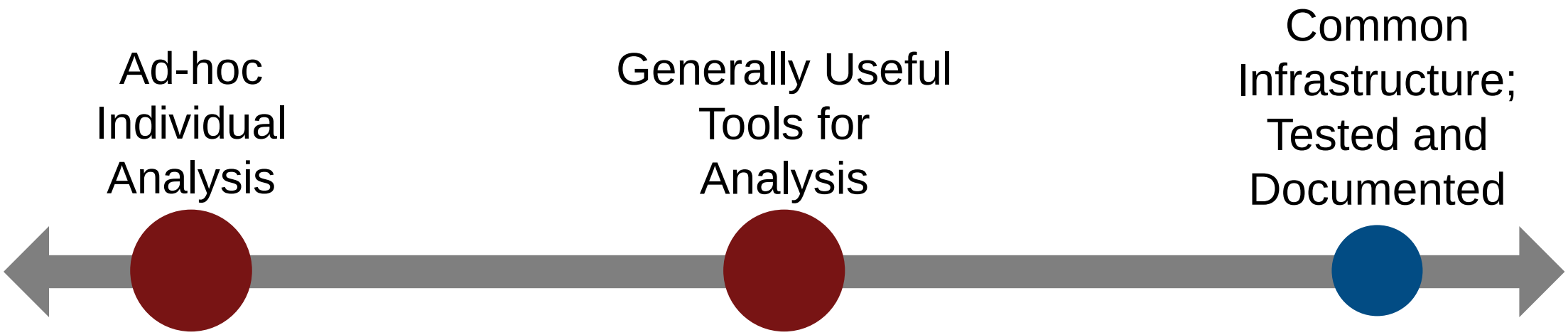
Summarize Fuel Efficiency using a histogram

```
binEdges = 0:80;
histogram(t.FE,binEdges);
```

```
Evaluating tall expression using the Parallel Pool 'local'
```

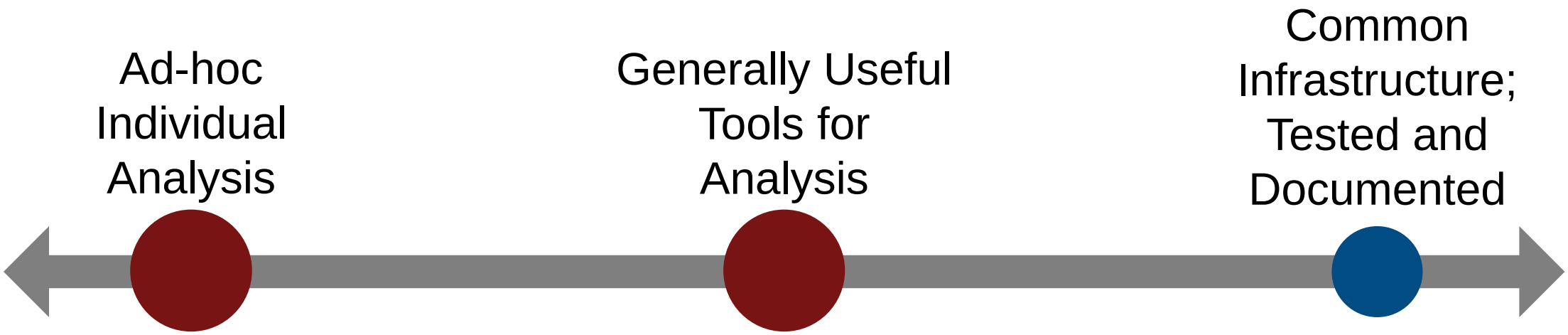


Data Science Maturity Levels



- **Make it easy to navigate the data**
- **Re-use each time you analyze the dataset**

Data Science Maturity Levels

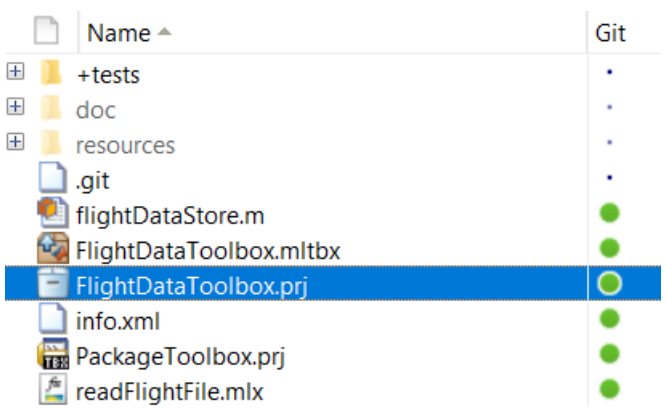


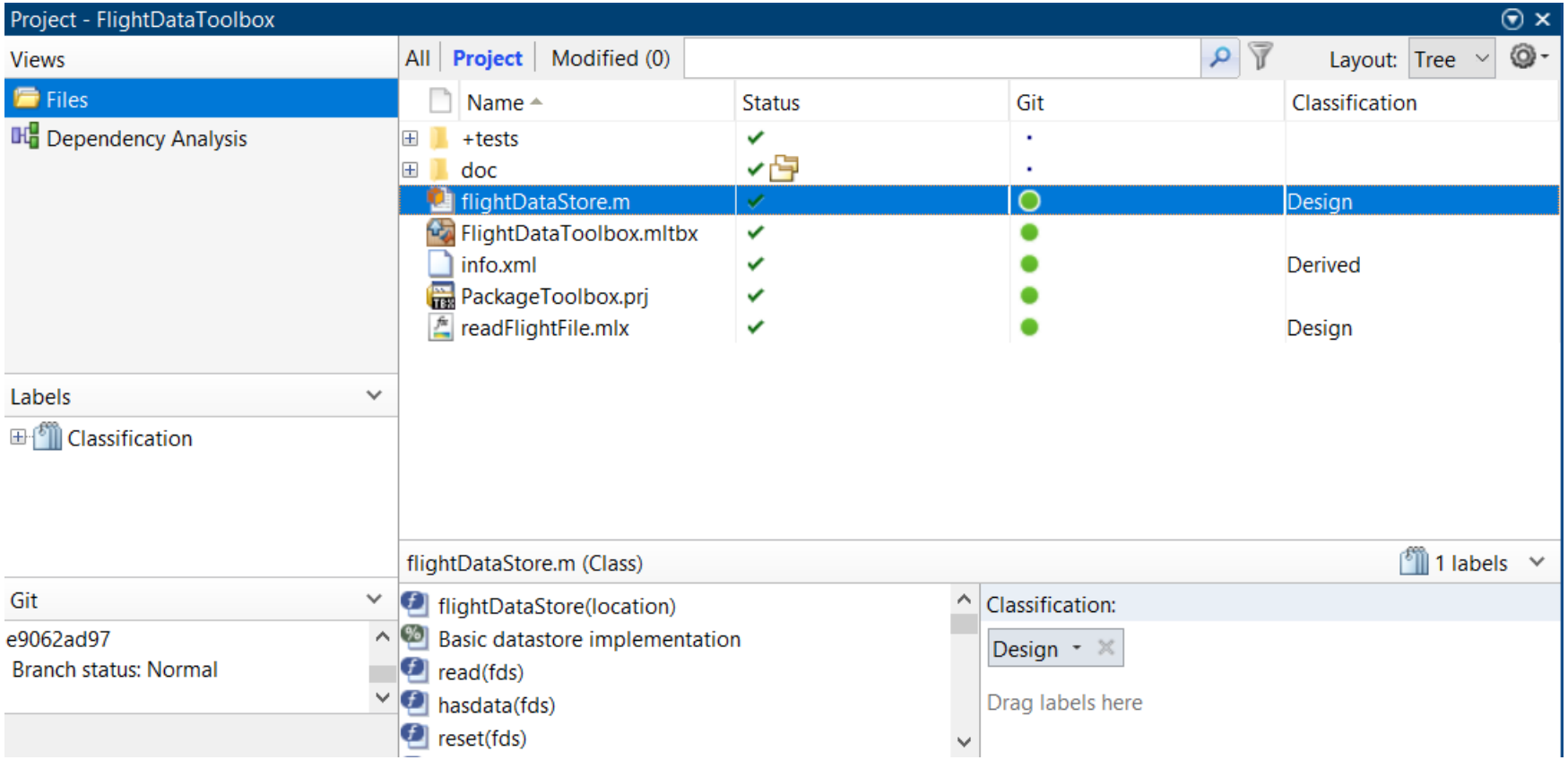
- **Collaborate: Work with others on a common code base**
- **Verify: Write well-tested software**
- **Share: Build tools for others**

MATLAB for Data Science Teams

1. MATLAB Projects
2. Unit Testing
3. Toolbox Packaging

MATLAB Projects



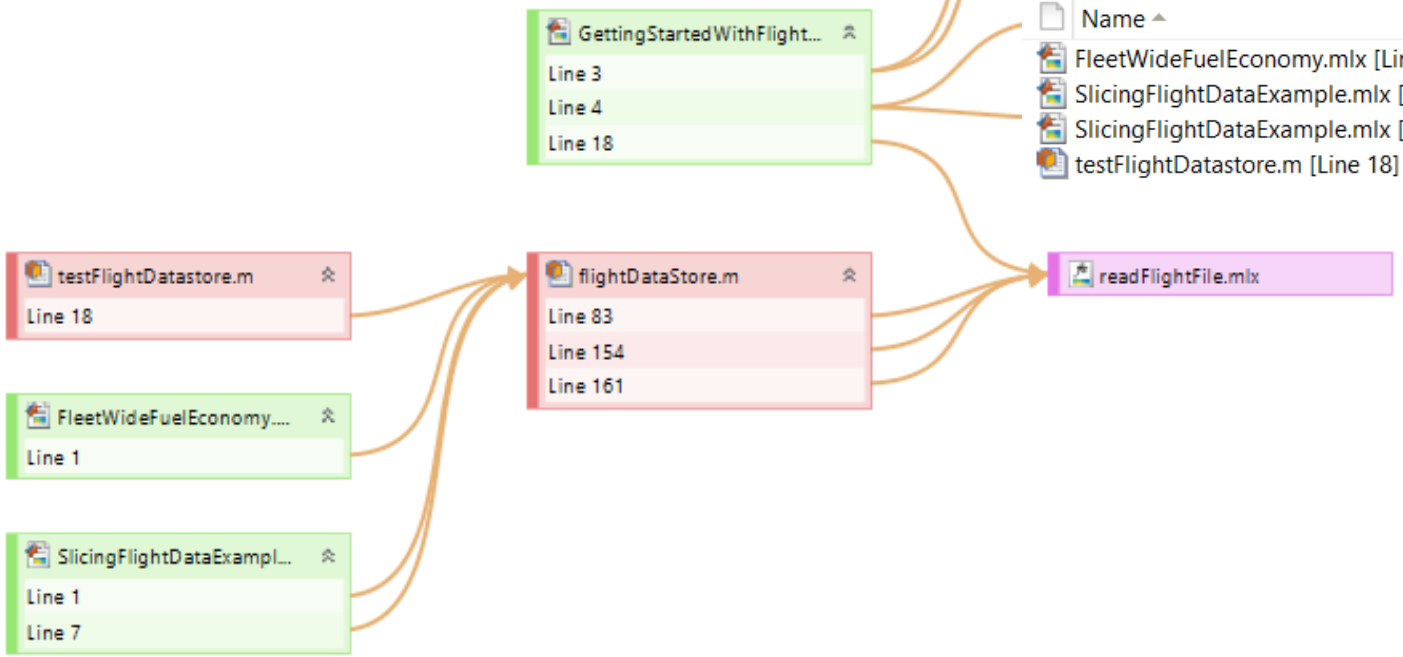


MATLAB Projects

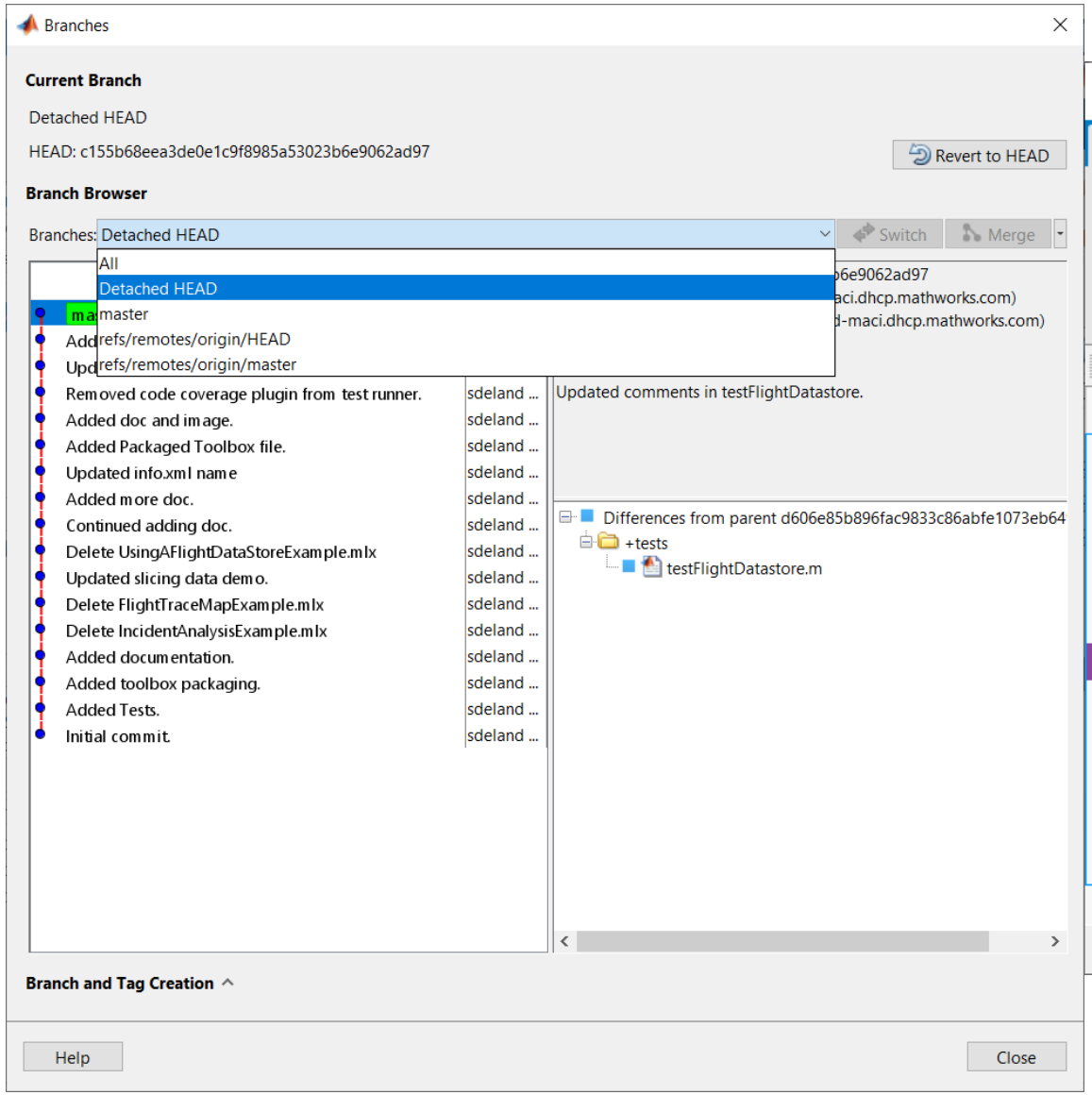
Name	Path	Problem Description	Project Status
A_T.data		Missing file	✗
A_T.Rate		Missing file	✗
ALT.data		Missing file	✗
ALT.Rate		Missing file	✗
DevelopingAFlightDataStore.mlx	$\doc\examples	Missing project file	✗
fdt_examples.m	$\doc		✓
fdt_functions.m	$\doc		✓
fdt_release_notes.m	$\doc		✓
FleetWideFuelEconomy.mlx	$\doc\examples		✓
flightDataStore.m	$\		✓
FlightTraceMapExample.mlx	$\doc\examples	Missing project file	✗
GettingStartedWithFlightData.mlx	$\doc\examples		✓
helptoc.xml	$\doc		✓
IncidentAnalysisExample.mlx	$\doc\examples	Missing project file	✗

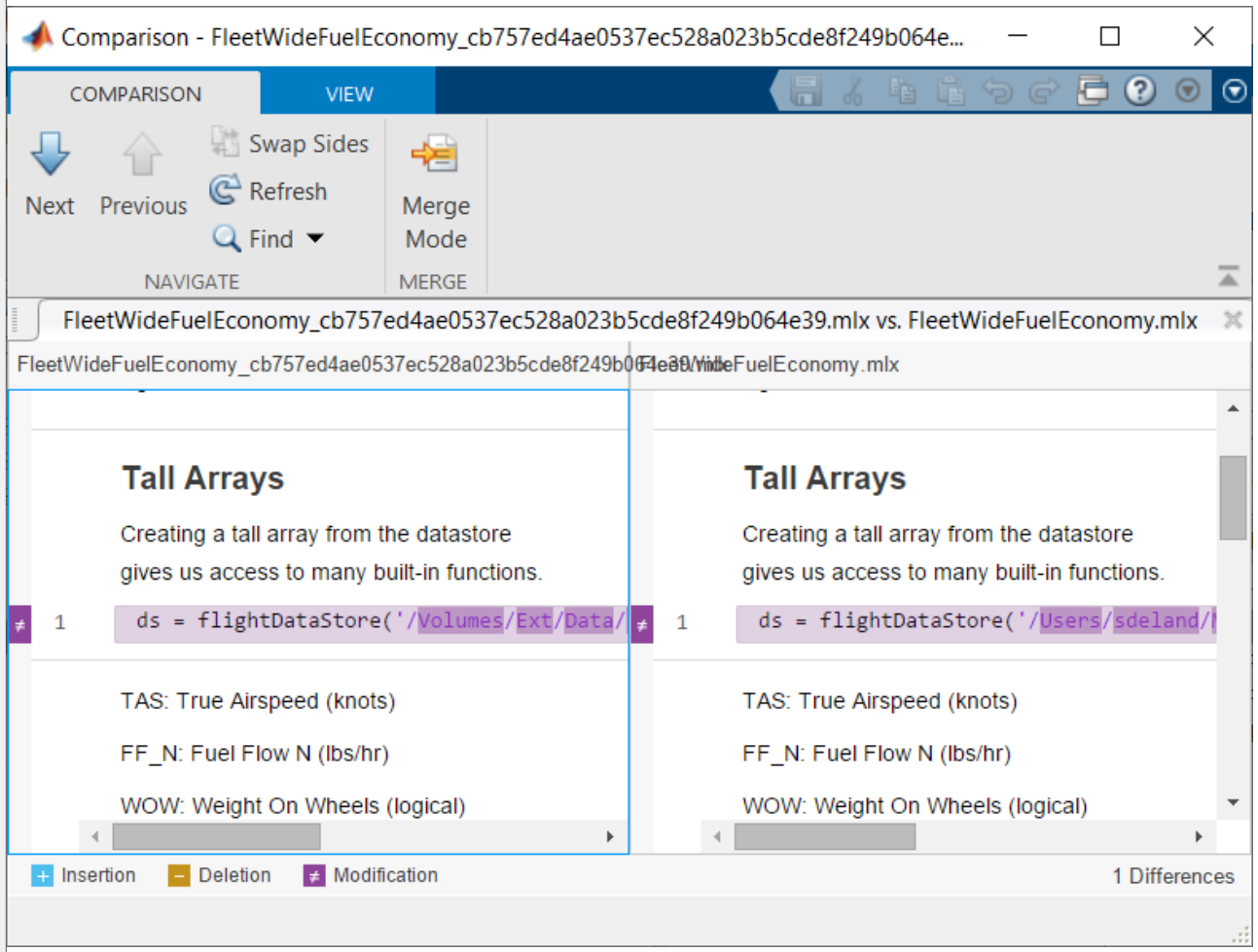
Show: Directly Impacted Files

Name	Path	Requires
FleetWideFuelEconomy.mlx [Line 1]	$\doc\examples	flightDataStore.m
SlicingFlightDataExample.mlx [Line 1]	$\doc\examples	flightDataStore.m
SlicingFlightDataExample.mlx [Line 7]	$\doc\examples	flightDataStore.m
testFlightDatastore.m [Line 18]	$\+tests	flightDataStore.m

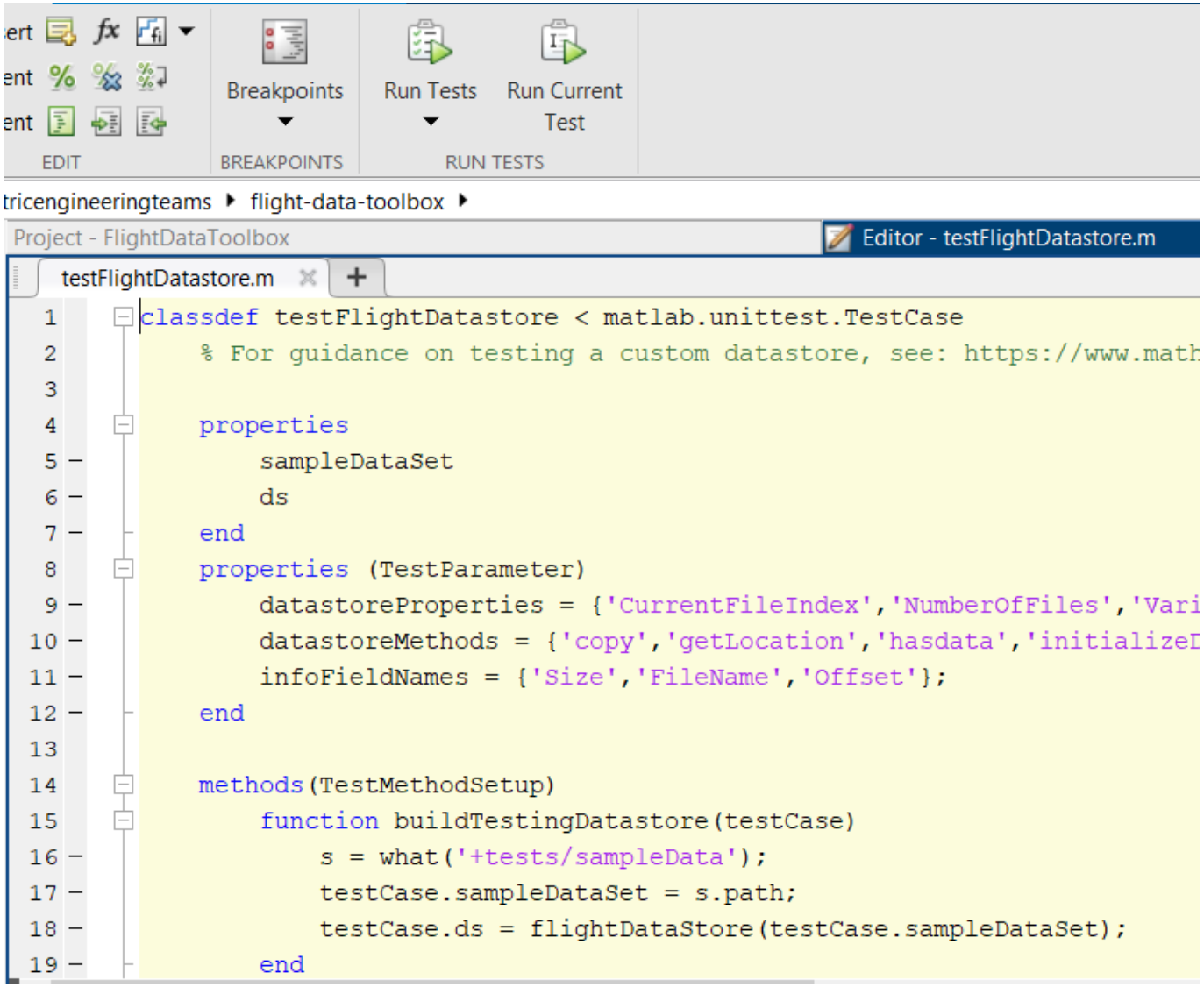


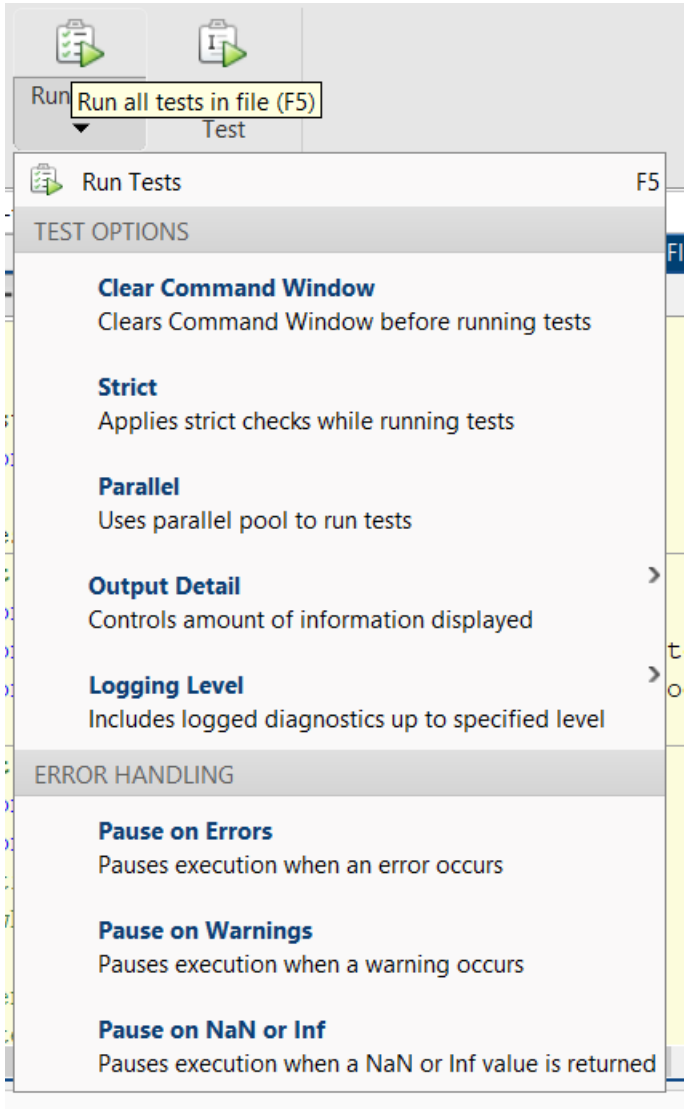
MATLAB Projects





Testing


Breakpoints
Run Tests
Run Current Test
EDIT
BREAKPOINTS
RUN TESTS
tricengineeringteams ▸ flight-data-toolbox ▸
Project - FlightDataToolbox
Editor - testFlightDatastore.m
testFlightDatastore.m
classdef testFlightDatastore < matlab.unittest.TestCase
% For guidance on testing a custom datastore, see: https://www.math
properties
sampleDataSet
ds
end
properties (TestParameter)
datastoreProperties = {'CurrentFileIndex','NumberOfFiles','Vari
datastoreMethods = {'copy','getLocation','hasdata','initializeD
infoFieldNames = {'Size','FileName','Offset'};
end
methods(TestMethodSetup)
function buildTestingDatastore(testCase)
s = what('+tests/sampleData');
testCase.sampleDataSet = s.path;
testCase.ds = flightDataStore(testCase.sampleDataSet);
end


Run
Run all tests in file (F5)
Test
Run Tests F5
TEST OPTIONS
Clear Command Window
Clears Command Window before running tests
Strict
Applies strict checks while running tests
Parallel
Uses parallel pool to run tests
Output Detail
Controls amount of information displayed
Logging Level
Includes logged diagnostics up to specified level
ERROR HANDLING
Pause on Errors
Pauses execution when an error occurs
Pause on Warnings
Pauses execution when a warning occurs
Pause on NaN or Inf
Pauses execution when a NaN or Inf value is returned

Testing

Testing Guidelines for Custom Datastores

All datastores that are derived from the custom datastore classes share some common behaviors. This test procedure provides guidelines to test the minimal set of behaviors and functionalities that all custom datastores should have. You will need additional tests to qualify any unique functionalities of your custom datastore.

If you have developed your custom datastore based on instructions in Develop Custom Datastore, then follow these test procedures to qualify your custom datastore. First perform the unit tests, followed by the workflow tests:

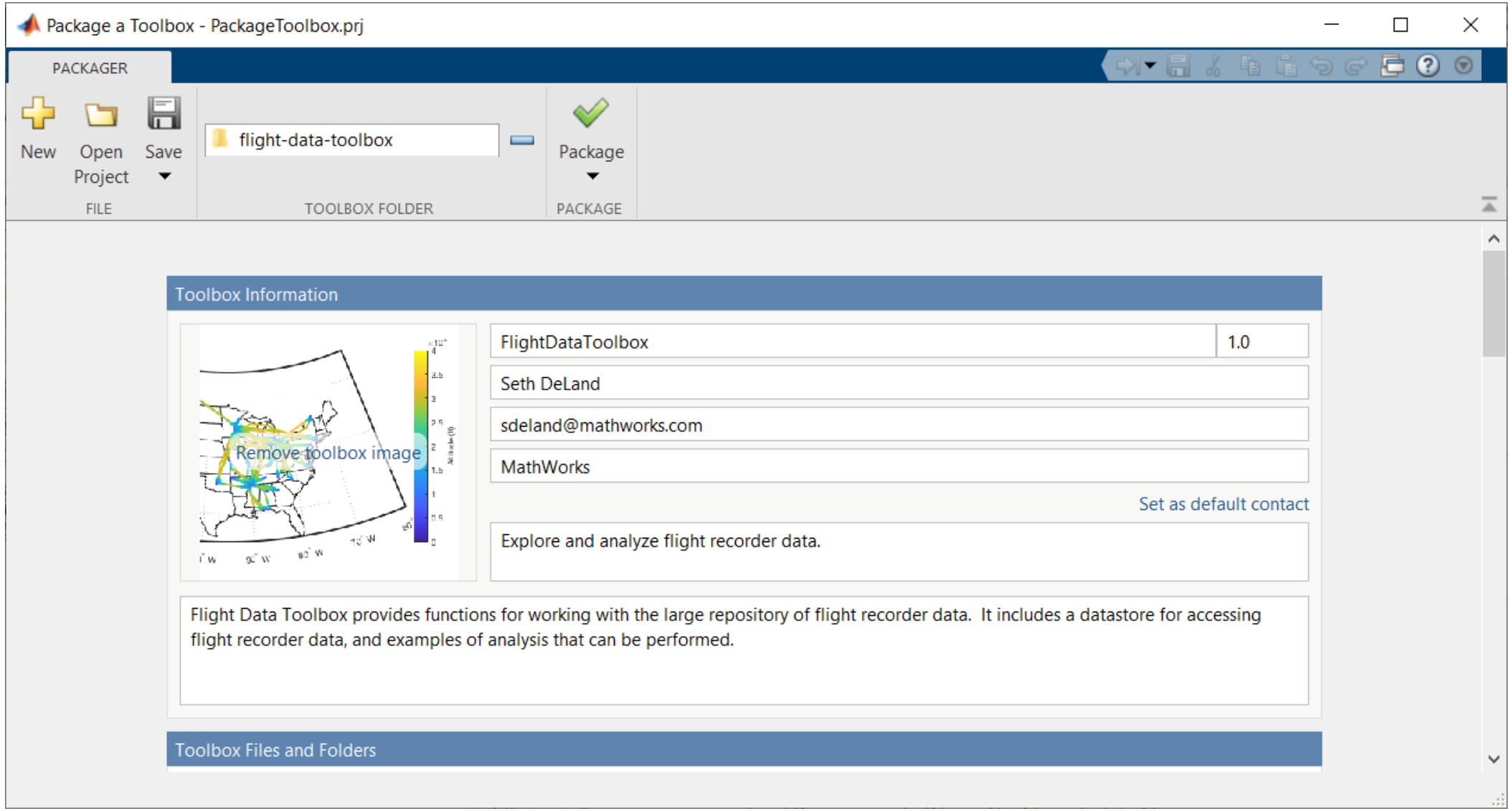
- Unit tests qualify the datastore constructor and methods.
- Workflow tests qualify the datastore usage.

`hasdata`

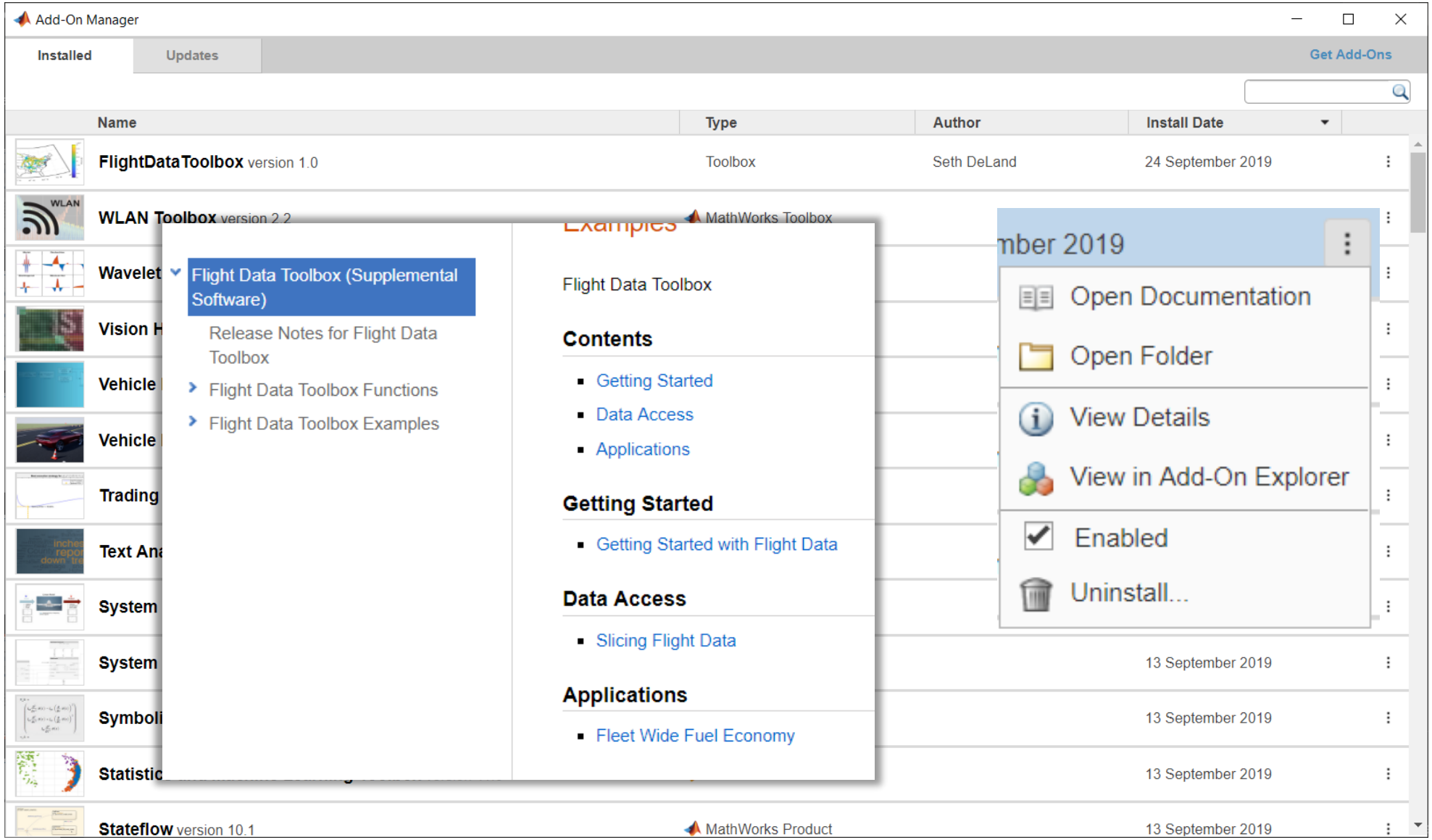
Unit test guidelines for the `hasdata` method

Test Case Description	Expected Output
Call the `hasdata` method on the datastore object before making any calls to `read`	`true`
Call the `hasdata` method on the datastore object after making a few calls to `read`, but before all the data is read	`true`
When more data is available to read, call the `readall` method, and then call the `hasdata` method.	`true`
When no more data is available to read, call the `hasdata` method.	`false`

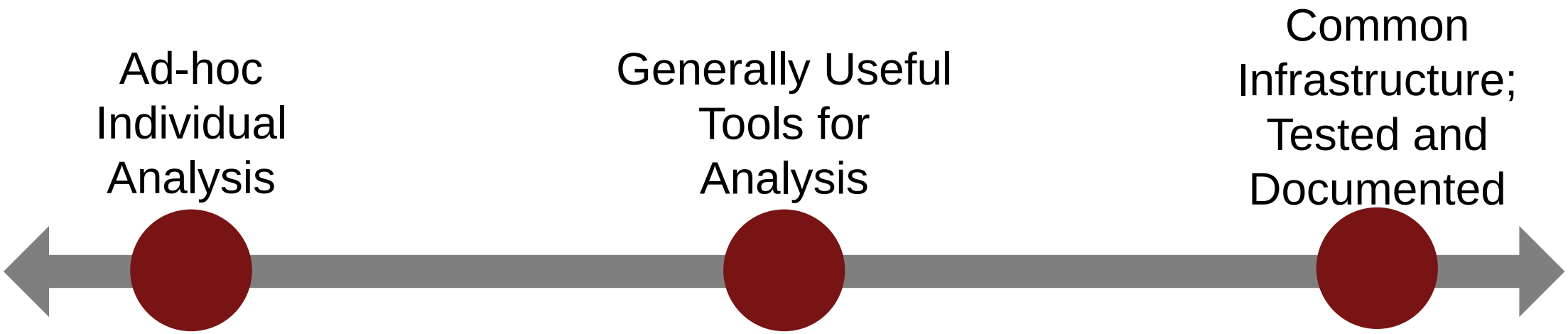
Creating a Toolbox



Creating a Toolbox



Data Science Maturity Levels



- **Scale-out to larger group of users**
- **Easier to maintain and share**

Takeaways

- MATLAB has many new tools to help you **better work with and utilize your data**
- Create tools for **you** / **your team** / **your organization** to explore and analyze data
- Increasing maturity with data science is a journey; we're here to help