

# AITK and DSDL Operationalization and Use Case Workshop

## Lab Guide

This lab guide contains the hands-on exercises for the AITK and DSDL Operationalization and Use Case workshop. Before proceeding with these exercises, please ensure that you have a copy of the workshop slide deck, which will help to put into context the tasks you are carrying out.

Download the workshop slide deck: <https://splk.it/MLTK-DSDL-Attendee>

### Prerequisites

In order to complete these exercises, you will need your own Splunk instance. Splunk's hands-on workshops are delivered via the [Splunk Show portal](#) and you will need a splunk.com account in order to access this.

If you don't already have a Splunk.com account, please create one [here](#) before proceeding with the rest of the workshop.



### Troubleshooting Connectivity

If you experience connectivity issues with accessing either your workshop environment or the event page, please try the following troubleshooting steps. If you still experience issues please reach out to the team running your workshop.

- **Use Google Chrome** (if you're not already)
- If the event page (i.e. <https://show.splunk.com/event/<eventID>>) didn't load when you clicked on the link, try **refreshing the page**
- **Disconnect from VPN** (if you're using one)
- **Clear your browser cache and restart your browser** (if using Google Chrome, go to: Settings > Privacy and security > Clear browsing data)
- **Try using private browsing mode** (e.g. Incognito in Google Chrome) to rule out any cache issues
- **Try using another computer** such as your personal computer - all you need is a web browser! Cloud platforms like AWS can often be blocked on corporate laptops.

## Table of Contents

<b>Overview</b>	<b>3</b>
<b>Module 0: Introduction</b>	<b>4</b>
<b>Module 1: Anomaly Detection</b>	<b>12</b>
Task 1: Data Exploration	12
Task 2: Model Training	17
Task 3: Model Testing	19
Task 4: Model Application	23
<b>Module 2: Clustering</b>	<b>25</b>
Task 1: Data Exploration	25
Task 2: Feature Engineering	30
Task 3: Clustering and Exploration	34
<b>Module 3: Predictive Analysis</b>	<b>39</b>
Task 1: Review Data and Engineered Features	39
Task 2: Model Training, Testing, and Application for Algorithm Prediction	41
Task 3: Model Training, Testing, and Application for Class Prediction	47
<b>Module 4: Data Science &amp; Deep Learning</b>	<b>53</b>
Task 1: Verify DSDL Setup	53
Task 2: Review the Example results in the DSDL interface	60
Task 3: Access the Jupyter Lab	64
Task 4: Test Sample Dataset Creation	66
Task 5: Final Sample Dataset Creation	69
Task 6: Modeling	73
Task 7: Review Results and Visualization Creation	76

## Overview

Welcome to the AITK and DSDL Operationalization and Use Case Workshop lab guide! This workshop consists of 5 modules. The modules are designed to be independent of each other, however for the full experience we recommend going through each module in order.

A few quick notes about how the exercise guide is formatted:

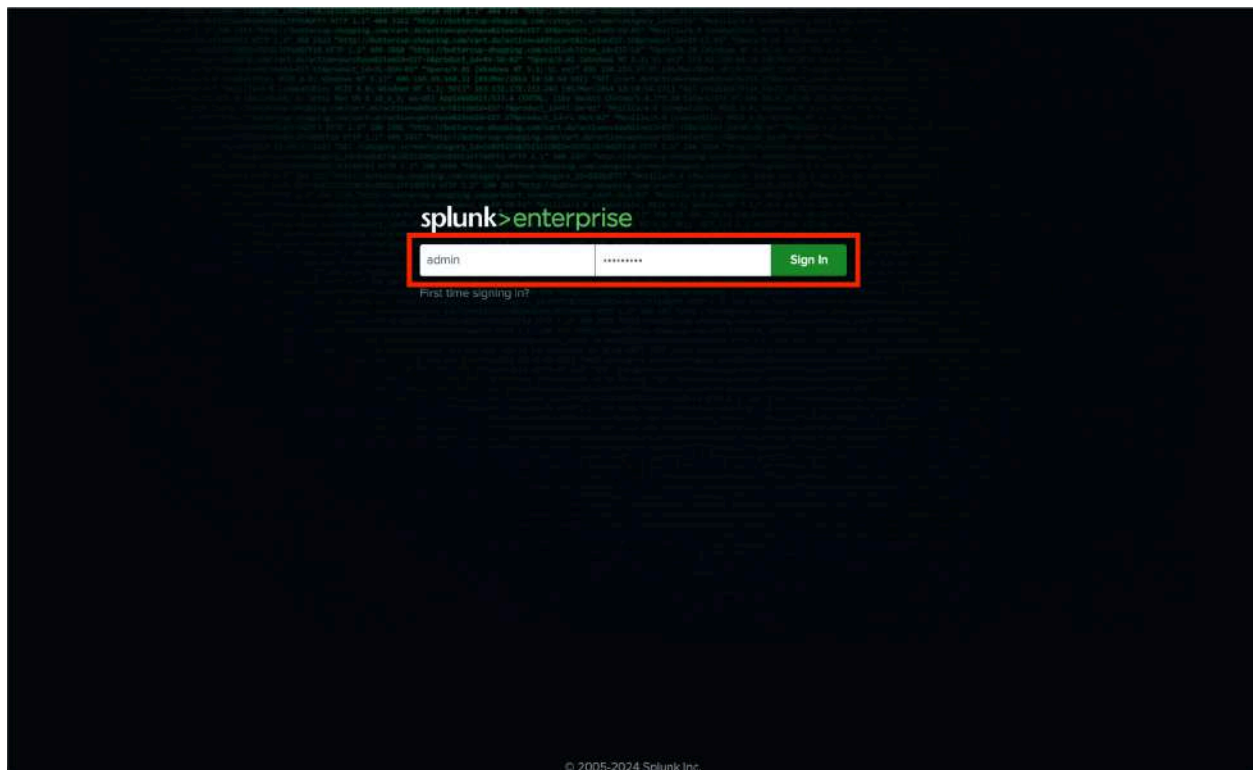
- Mono-spaced text `like this` and code blocks indicate specific callouts you need to type or pay attention to.
- Bolded text **like this** indicates an action you need to take such as clicking a button in a browser window.
- The screenshots in the lab use the light theme so it's easier to read in the instructions. However, your environment might be configured to use the dark theme. If you would like to change your interface theme setting, you're welcome to do so by navigating to **{{ your username provided by Show }}** > **Preferences** then changing **Theme** to **Light**, clicking **Apply**, then refreshing your browser tab. In the screenshots below, the username is `Administrator`.
- The **red** boxed sections in the screenshots indicate where you need to interact with the browser window by clicking or typing something, and **blue** boxes are areas of interest.
- Comments and notes are *italicized*.

## Module 0: Introduction

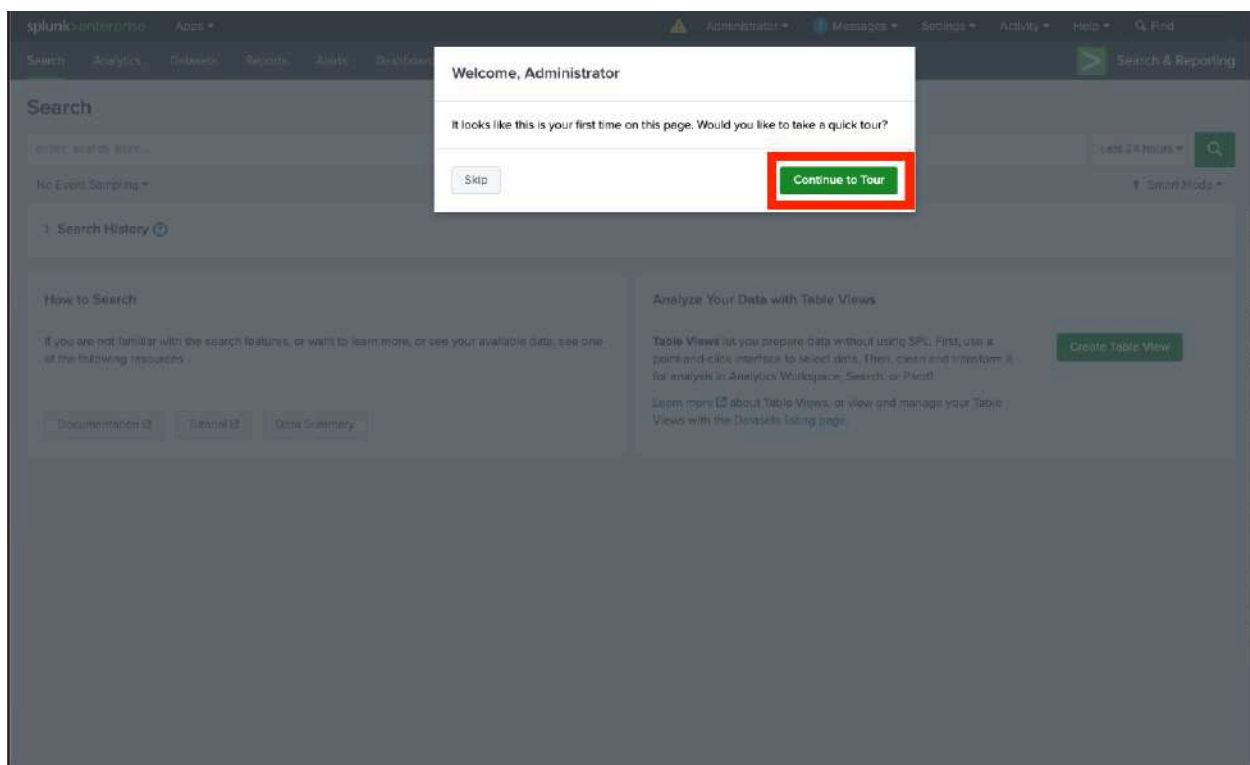
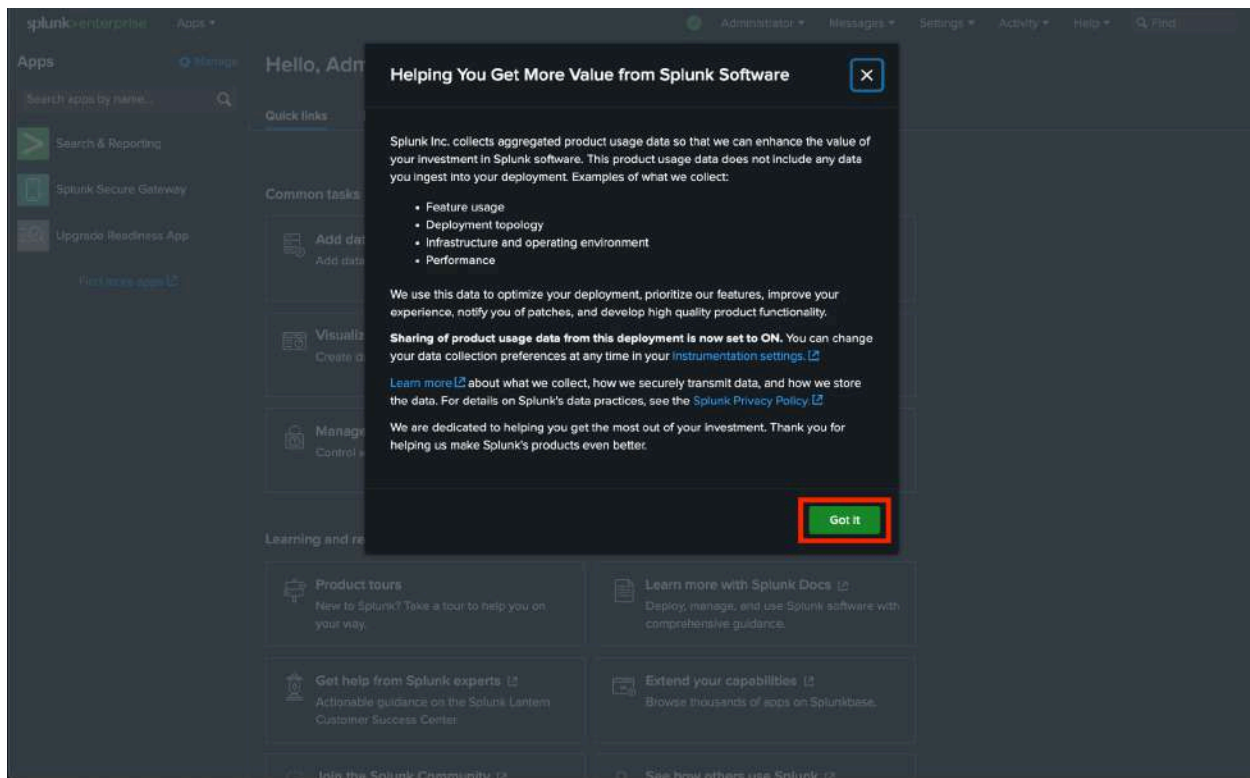
This module is designed to help you get acquainted with the environment that will be used during this workshop. You will be running a search to verify data is present in the environment, and running through one of the Splunk Machine Learning Toolkit (MLTK) Forecasting Examples in the MLTK Showcase. The MLTK Showcase is a great place to see what the MLTK can do, and find examples of machine learning workflows in Splunk.

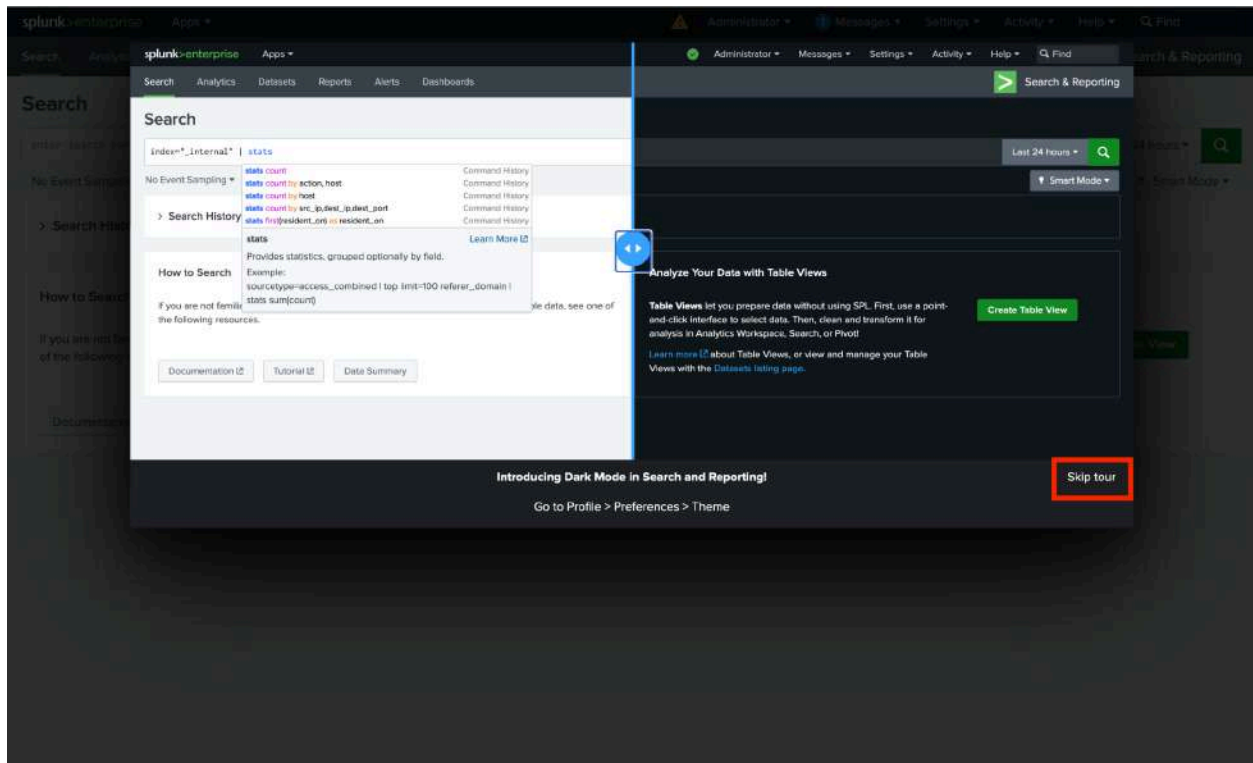
This module is not meant to be challenging and really just designed to make sure you can log into everything before the next module. If you run into problems or need help please ask!

1. Open a browser tab and navigate to the Splunk Enterprise URL provided by Splunk Show. Log in using the provided Username and Password.



2. If you are prompted by a Helping You Get More Value from Splunk Software, Welcome, user, or Introducing Dark Mode in Search and Reporting! pop up, feel free to dismiss it by clicking **Skip**, **Skip tour**, or **Got it**.

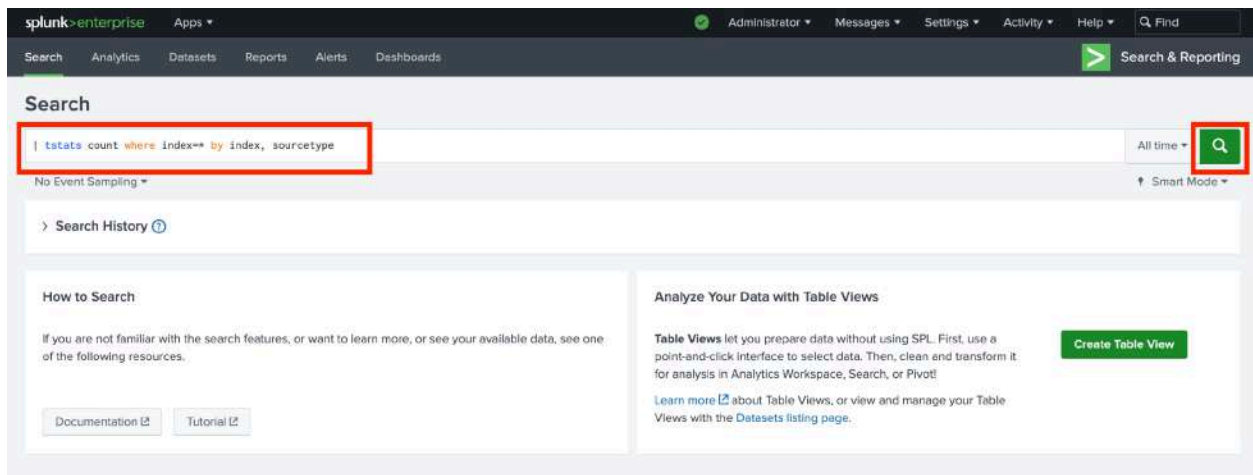




- To discover what datasets are available in the Splunk instance, run the following search (copying and pasting it is easiest).

| `tstats count where index=* by index, sourcetype`

*This search shows what data is available to be searched, split by index and sourcetype.*



You should see five sourcetypes across three different indexes.

*Please note that the time range picker is set to All time. This was set intentionally and should not be changed during this workshop.*

splunk>enterprise Apps Administrator Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

### New Search

Save As Create Table View Close

| `tstats count where index=* by index, sourcetype` All time

7,805,167 events (1/17/0 12:00:00.000 AM to 5/23/24 6:03:28.000 PM) No Event Sampling Job

Events Patterns **Statistics (5)** Visualization

20 Per Page Format Preview

index	sourcetype	count
botnet	netflow_botnet	5950739
botnet	netflow_botnet_balanced	25608
dga	dga_domains	100000
dga	dga_test	1728112
dsdl	logins	708

- Navigate to the Splunk Machine learning Toolkit app by clicking **Apps** on the menu bar, then selecting **Splunk Machine Learning Toolkit**.

splunk>enterprise **Apps** Administrator Messages Settings Activity Help Find

Search Search & Reporting Alerts Dashboards Search & Reporting

New Search

| `tstats count where index=* by index, sourcetype` All time

7,805,167 events (1/17/0 12:00:00.000 AM to 5/23/24 6:03:28.000 PM) No Event Sampling Job

Events Patterns **Statistics (5)** Visualization

20 Per Page Format Preview

index	sourcetype	count
botnet	netflow_botnet	5950739
botnet	netflow_botnet_balanced	25608
dga	dga_domains	100000
dga	dga_test	1728112
dsdl	logins	708

Search & Reporting

- Botnet App for Splunk
- DGA App for Splunk
- Parallel Coordinates
- Python for Scientific Computing
- Splunk App for Data Science and Deep Learning
- Splunk Machine Learning Toolkit**
- Splunk Secure Gateway
- Upgrade Readiness App
- Manage Apps
- Find More Apps

- Click **Forecast Time Series**, select the **Forecast Time Series** tab (you may need to scroll down), then click the **Forecast the Number of Employee Logins** example.

splunk>enterprise Apps Administrator Messages Settings Activity Help Find

Showcase Experiments Search Models Settings Docs Video Tutorials Splunk Machine Learning Toolkit

## Showcase

Welcome to the Machine Learning Toolkit Showcase. Watch and learn from interactive end-to-end examples using real datasets. Click on an example to pre-populate the Assistant with the sample dataset and its settings. Inspect the Search Processing Language as well as the underlying code of these examples to see how it all works.

View examples by ML Operation Industry Filter Examples

### Predict Fields

View examples that predict the value of a numeric or categorical field using the values from other fields in the event.

15 Examples Available

### Detect Outliers

View examples that detect numeric and categorical values that differ significantly from values in the rest of the data. Identified outliers are indicative of interesting, unusual, and possibly dangerous events.

14 Examples Available

### Forecast Time Series

View examples that predict the next value in a sequence of time series data by using past time series data.

9 Examples Available

### Cluster Events

View examples that partition events with multiple fields into groups of events based on the values of those fields.

Featured Examples

#### Forecast App Expenses from Multiple Variables

This example uses the Smart Forecasting Assistant to forecast CRM and ERP for the next month, based on three months of data. The Smart Forecasting Assistant leverages the StateSpaceForecast algorithm.

IT

#### Forecast App Logons with Special Days

This example uses the Smart Forecasting Assistant to forecast the number of logons for the next 30 points, based on nine months of data. A lookup file excludes special days from the forecast. The Smart Forecasting Assistant leverages the StateSpaceForecast algorithm.

IT

#### Forecast Internet Traffic

This example uses the Forecast Time Series Assistant and the Kalman Filter algorithm to forecast future internet traffic.

IT

#### Forecast Monthly Sales

This example uses the Forecast Time Series Assistant and the Kalman Filter algorithm to forecast future souvenir sales numbers.

Business Analytics

Smart Forecasting Forecast Time Series

### Forecasting Examples

View examples that forecast future values given past values of a numeric time series metric.

#### Forecast Internet Traffic

This example uses the Forecast Time Series Assistant and the Kalman Filter algorithm to forecast future internet traffic.

IT

#### Forecast the Number of Employee Logins

This example uses the Forecast Time Series Assistant and the Kalman Filter algorithm to forecast future employee login numbers.

Security

#### Forecast Monthly Sales

This example uses the Forecast Time Series Assistant and the Kalman Filter algorithm to forecast future souvenir sales numbers.

Business Analytics

#### Forecast the Number of Bluetooth Devices

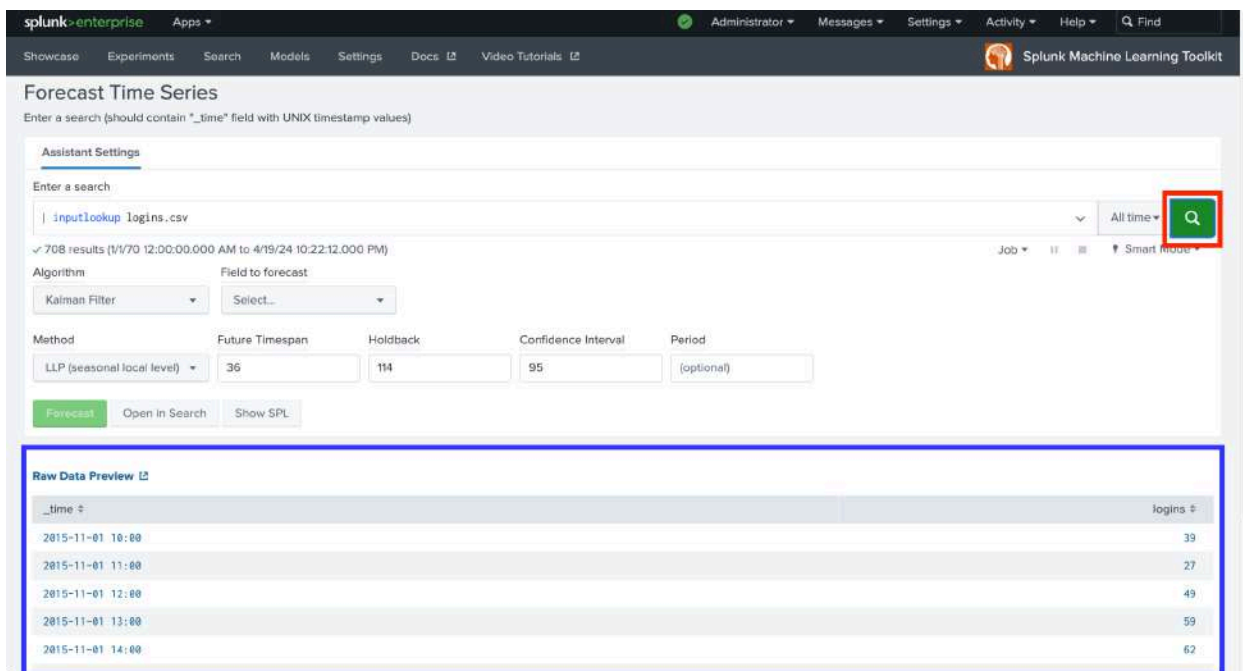
This example uses the Forecast Time Series Assistant and the Kalman Filter algorithm to forecast distinct addresses.

#### Forecast Exchange Rate TWI using ARIMA

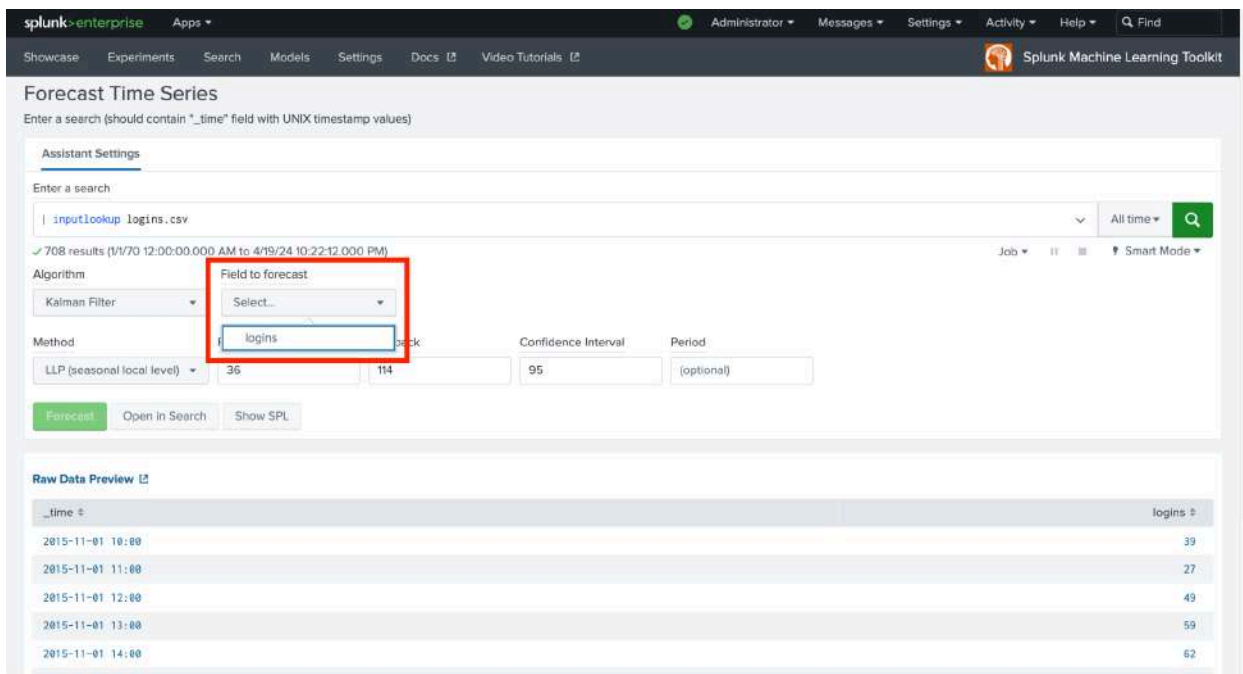
This example uses the Forecast Time Series Assistant and the ARIMA algorithm to forecast the exchange rate.

6. Click the search icon (magnifying glass) on the right to run a search that displays the training data.



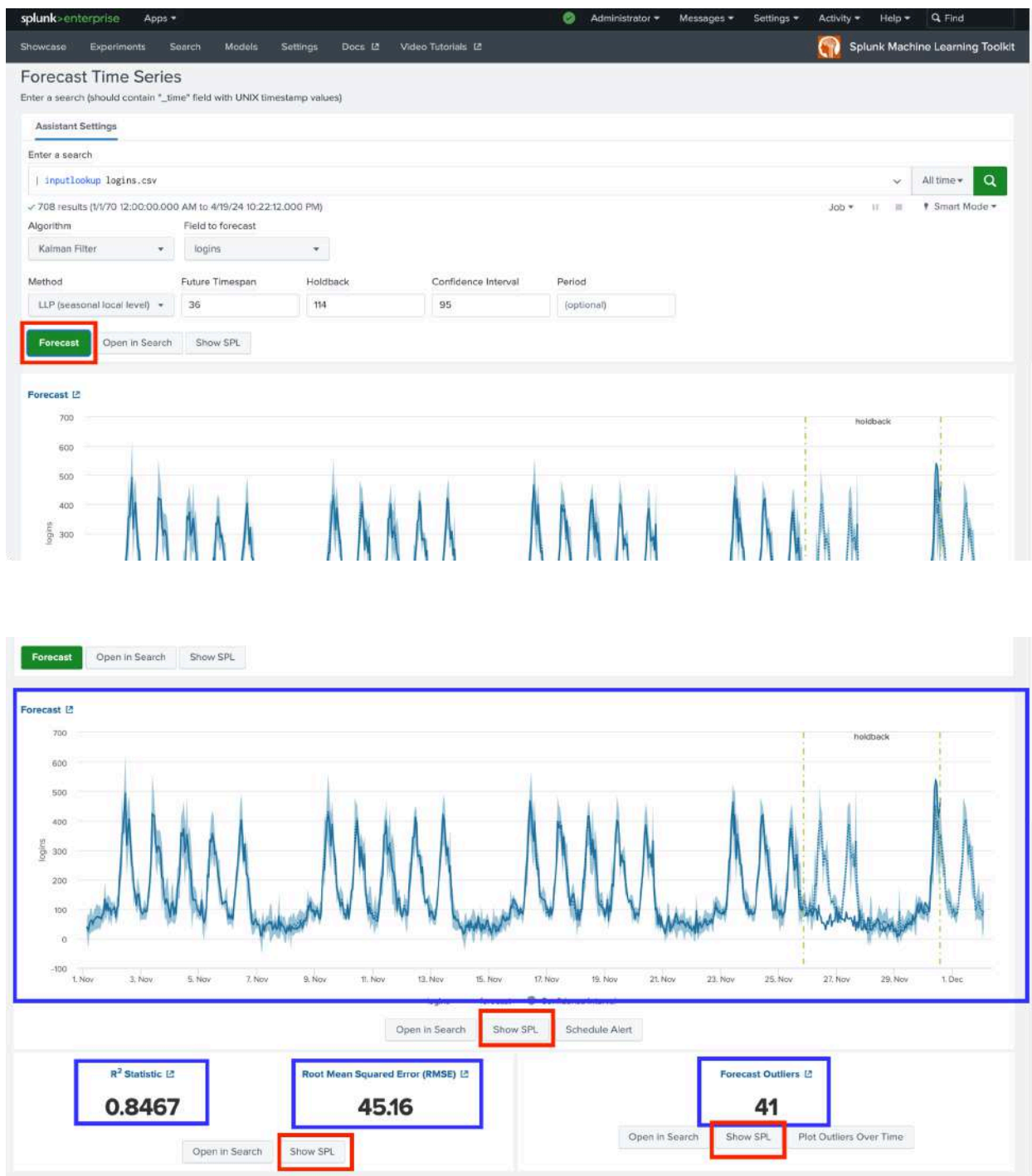


- Click the **Field to forecast** dropdown, and select `logins`.

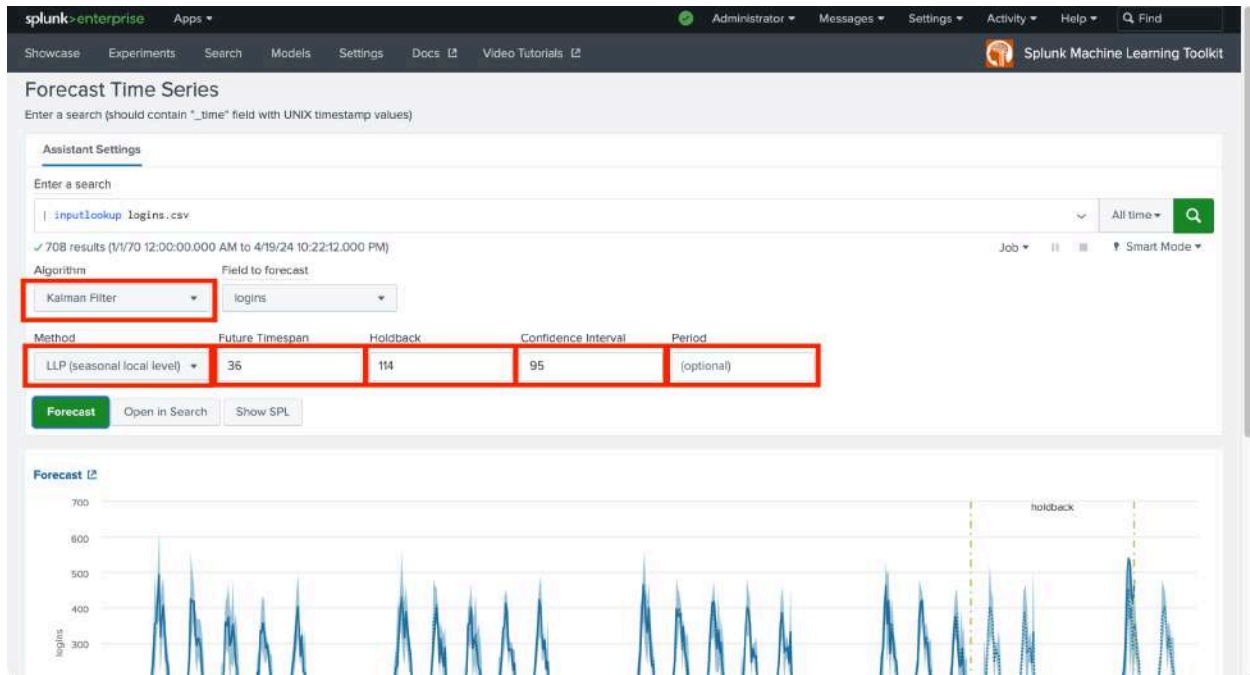


- Click **Forecast** to start the forecast process, then view the results and accuracy under the **Forecast**. The forecast accuracy can be viewed under the  $R^2$  Statistic, Root Mean Squared Error (RMSE), and Forecast Outliers panels below the **Forecast** panel. For any of these panels, feel free to click **Show SPL** to view the SPL used to generate the dashboard panel shown.

*If you want to learn more about  $R^2$  Statistic, Root Mean Squared Error (RMSE), or Forecast Outliers, you can learn more by hovering above that text highlighted in blue.*



9. Also, feel free to experiment by changing the forecast **Algorithm** and any of the hyperparameters such as **Method**, **Future Timespan**, **Holdback**, **Confidence Interval**, and **Period**.



That's it for Module 0! The real ML work will start in the next module.

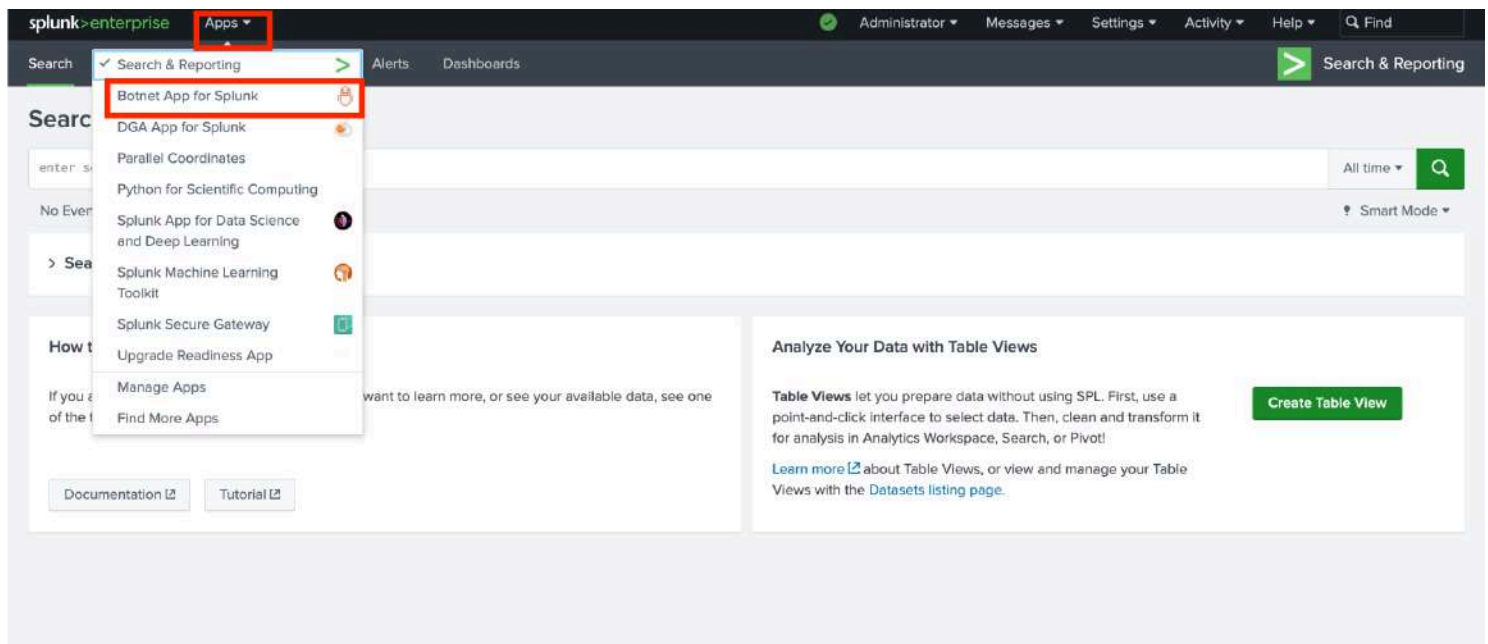
## Module 1: Anomaly Detection

In this module, you will be working with refined netflow data to train anomaly detection models to identify traffic being generated by botnets. This module is loosely based on the [Botnet App for Splunk](#) in that it uses the same data sets and dashboards will be referenced during this module. This app is valuable to show what's possible with Splunk in regards to machine learning. However, this module will be deconstructing and walking through various parts of the app to understand how it works.

### Task 1: Data Exploration

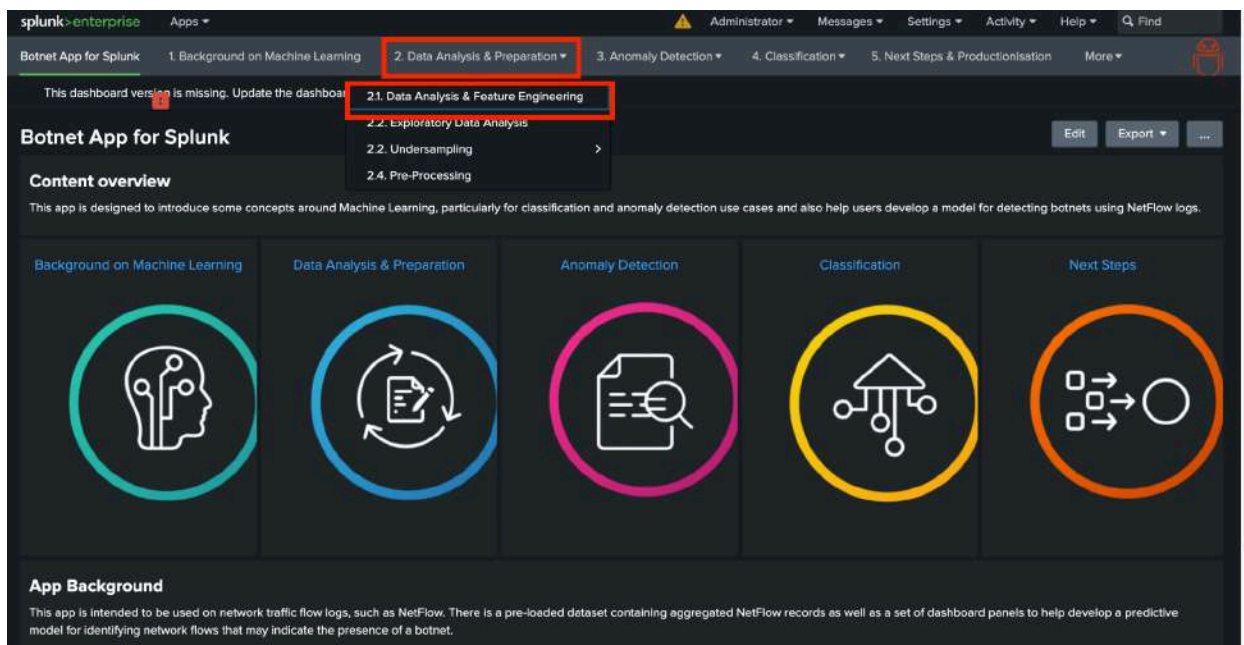
In this task, you will be exploring the data being used in this module. First, you'll take a look at the tables created in the Botnet App for Splunk, then create searches to explore the data in similar ways using the Search Processing Language (SPL). All of the data is in the `botnet` index, with the training dataset set as the `netflow_botnet` sourcetype, and the testing dataset set as the `netflow_botnet_balanced` sourcetype.

1. In the Splunk Enterprise host, navigate to the Botnet App for Splunk.

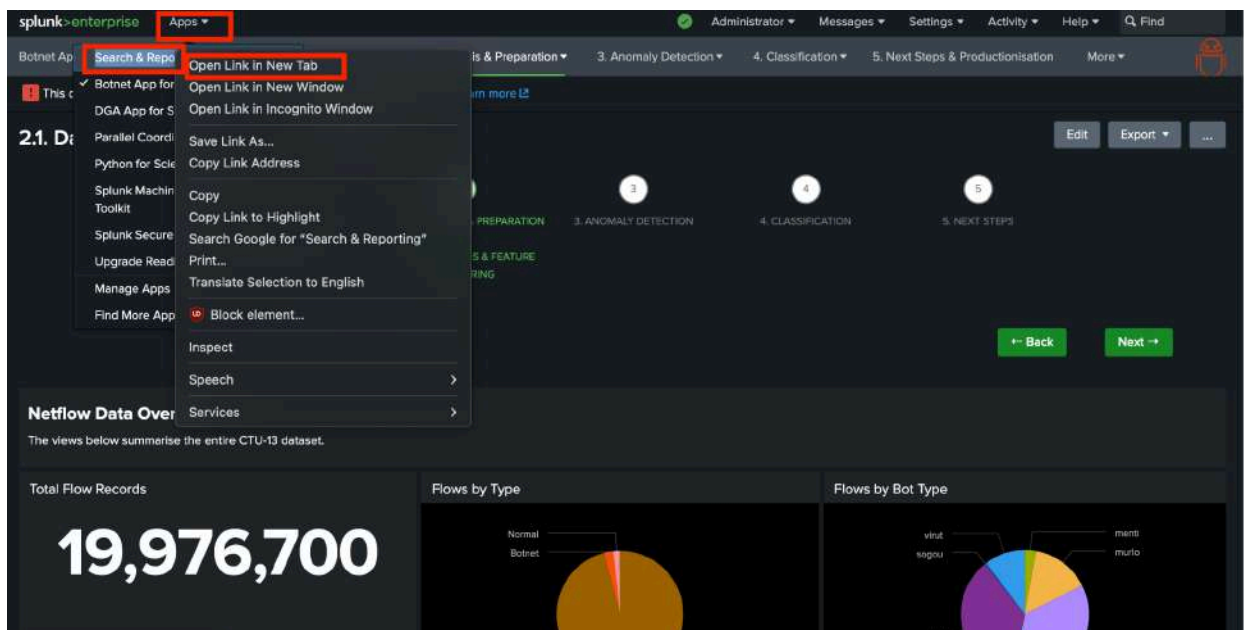


2. Expand the 2. Data Analysis & preparation > 2.1 Data Analysis & Feature Engineering dashboard.

*This panel might take a few minutes to load, and you do not need to wait for it to complete.*

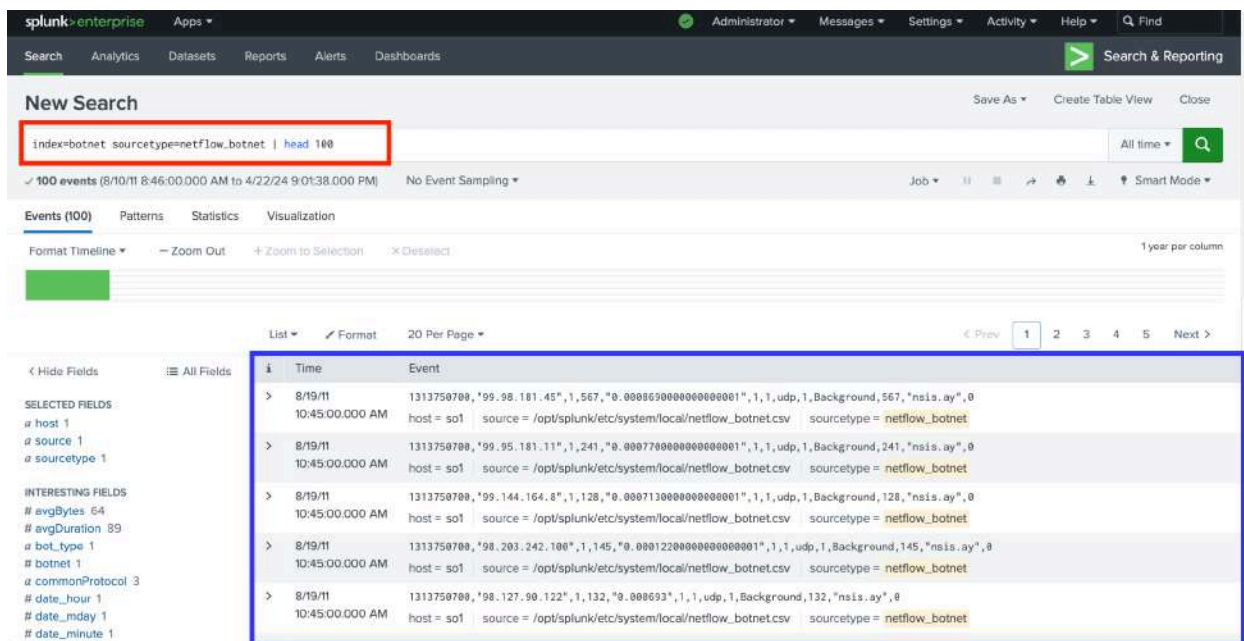


3. Open the **Search & Reporting** app in a new tab. You can do this by clicking **Apps**, right-clicking **Search & Reporting**, and clicking **Open link in New Tab**.



4. Run the search below to view a sample of the training data.

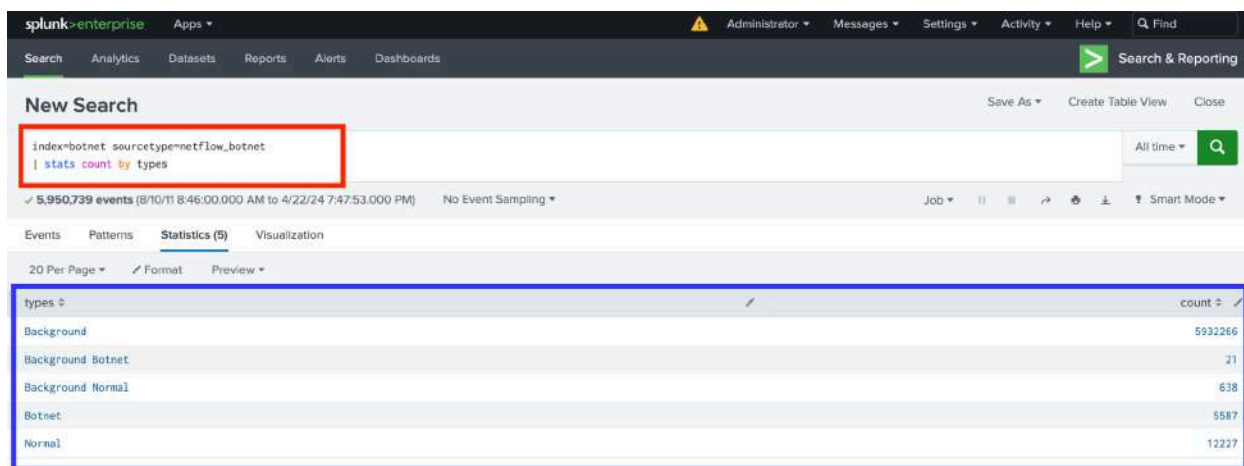
`index=botnet sourcetype=netflow_botnet | head 100`



- Run the following search to view the count of events in the training dataset by traffic type. This is for the most part recreating the **Flows by Type** panel on the 2.1 Data Analysis & Feature Engineering dashboard.

```
index=botnet sourcetype=netflow_botnet
| stats count by types
```

*As you can see, most of the events originate from normal Background traffic.*



6. Similarly, run the following search to view the count of events in the training dataset by botnet type. This is for the most part recreating the **Flows by Bot Type** panel on the **2.1 Data Analysis & Feature Engineering** dashboard.

```
index=botnet sourcetype=netflow_botnet botnet=1
| stats count by bot_type
```



**New Search**

index=botnet sourcetype=netflow\_botnet botnet=1  
| stats count by bot\_type

✓ 5,608 events (8/10/11 8:46:00.000 AM to 4/22/24 7:51:50.000 PM) No Event Sampling

Events Patterns **Statistics (7)** Visualization

20 Per Page Format Preview

bot_type	count
mentl	123
murlo	1054
neris	1891
nsis.ay	145
rbot	1391
sogou	5
virut	999

- Run the search below to re-create the Feature Relationships on the 2.1 Data Analysis & Feature Engineering dashboard, then click **Visualization**, and select the **Parallel Coordinates** visualization. This search and the related visualization are useful for identifying how different features relate to each other in the training data set.

*Note how the search is stratifying by botnet type by using the command | sample 500 by bot\_type.*

*Also, note how the features like bot\_type and numFlows are related to each other.*

```
index=botnet sourcetype=netflow_botnet botnet=1
| sample 500 by bot_type
| table bot_type numFlows sumBytes avgBytes avgDuration uniqueDstIp uniqueDstPort
uniqueProtocols commonProtocol
```

**New Search**

index=botnet sourcetype=netflow\_botnet botnet=1  
| sample 500 by bot\_type  
| table bot\_type numFlows sumBytes avgBytes avgDuration uniqueDstIp uniqueDstPort uniqueProtocols commonProtocol

✓ 2,233 events (8/10/11 8:46:00.000 AM to 4/22/24 7:53:54.000 PM) No Event Sampling

Events Patterns **Statistics (2,233)** Visualization

20 Per Page Format Preview

bot_type	numFlows	sumBytes	avgBytes	avgDuration	uniqueDstIp	uniqueDstPort	uniqueProtocols	commonProtocol
nsis.ay	1	693	693	350.659973	1	1	1	udp
sogou	49	18350381	374497.5714285714	38.473360408163266	14	2	2	tcp
mentl	35	7978	227.94285714285715	1.3538947714285716	34	2	1	tcp
rbot	1	19248	19248	3519.105225	1	1	1	tcp
nsis.ay	1	90	90	0	1	1	1	udp
neris	7	2659	379.85714285714283	11.034884714285713	1	1	1	udp
sogou	1	1009	1009	1.2065810000000001	1	1	1	tcp
mentl	56	14168	253	0.7668824107142856	32	2	1	tcp
rbot	4	28821	7205.25	2.7775742500000002	2	2	2	tcp
nsis.ay	1	970	970	0.3325	1	1	1	udp
murlo	1	1076	1076	60.940037	1	1	1	tcp

8. Run the following search to view the test data set.

`index=botnet sourcetype=netflow_botnet_balanced`

The screenshot shows the Splunk Enterprise web interface. At the top, the navigation bar includes 'splunk>enterprise', 'Apps', and user roles like 'Administrator', 'Messages', 'Settings', 'Activity', 'Help', and 'Find'. Below this is a 'Search & Reporting' section with a 'New Search' button. The search bar contains the query `index=botnet sourcetype=netflow_botnet_balanced`, which is highlighted with a red box. To the right of the search bar are buttons for 'Save As', 'Create Table View', and 'Close'. Below the search bar, it indicates '25,608 events (8/10/11 8:46:00.000 AM to 4/22/24 7:57:46.000 PM)' and 'No Event Sampling'. The interface shows a 'Format Timeline' button and a 'Zoom Out' button. On the left, there is a 'SELECTED FIELDS' section with 'a host 1', 'a source 1', and 'a sourcetype 1'. Below this is an 'INTERESTING FIELDS' section with various statistics like '# avgBytes 100+', '# avgDuration 100+', '# bot\_type 7', '# botnet 2', '# commonProtocol 8', '# date\_hour 24', '# date\_mday 10', and '# date\_minute 60'. The main content area displays a table of search results. The table has columns for 'Time' and 'Event'. The 'Event' column contains detailed network flow data, including host, source, and sourcetype. The table is paginated, showing results 1 through 8. The table is highlighted with a blue box.

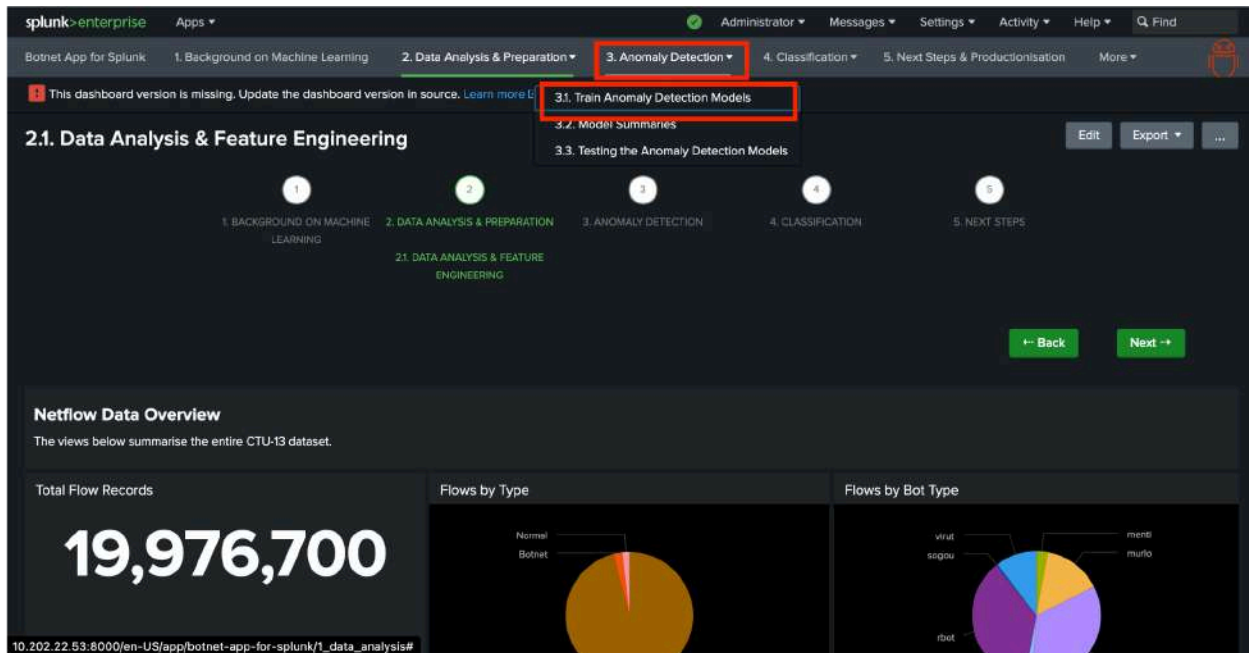
i	Time	Event
>	8/19/11 10:45:00.000 AM	1313758780,492,0.800649,nsis.ay,0,tcp,1,82.132.211.59,492,Background,1,1,1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet_balanced.csv sourcetype = netflow_botnet_balanced
>	8/19/11 10:45:00.000 AM	1313758780,134,0.000697,nsis.ay,0,udp,1,78.56.138.80,134,Background,1,1,1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet_balanced.csv sourcetype = netflow_botnet_balanced
>	8/19/11 10:45:00.000 AM	1313758780,549,0.000729,nsis.ay,0,udp,1,212.96.169.3,549,Background,1,1,1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet_balanced.csv sourcetype = netflow_botnet_balanced
>	8/19/11 10:44:00.000 AM	1313758640,366,0.9991040000000000,nsis.ay,0,tcp,1,95.29.208.147,366,Background,1,1,1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet_balanced.csv sourcetype = netflow_botnet_balanced
>	8/19/11 10:44:00.000 AM	1313758640,423,5.98739,nsis.ay,0,udp,1,94.113.3.54,423,Background,1,1,1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet_balanced.csv sourcetype = netflow_botnet_balanced



## Task 2: Model Training

The features in the testing and training datasets are already in a numeric format we can use, so feature engineering can be skipped! This means we can progress right to training the models. You will train multiple [Density Function](#) models referencing different features in the training data set, with the goal of using the trained models to identify botnet traffic (and by extension normal traffic because anything not anomalous for botnet traffic will be normal traffic).

1. In the browser tab open to the Botnet App for Splunk, expand the options under 3. Anomaly Detection, and open the 3.1 Train Anomaly Detection Models dashboard. In the app, this is the dashboard that is used to train the models.



2. In the browser tab open to the Search & Reporting app, run the following search to train 7 new Density Function models, each trained to look for anomalies in different features.

*In our testing, this search took about 90 seconds to complete.*

*Note how each `fit` command runs against a different feature from the dataset, this search is only looking for botnet data, and it is stratifying across botnet type.*

*Feel free to ignore any warnings about undersampling or having too few data sets.*

```

index=botnet sourcetype=netflow_botnet botnet=1
| fit DensityFunction numFlows by bot_type as anomaly_numFlows into module1_numFlows
| fit DensityFunction sumBytes by bot_type as anomaly_sumBytes into module1_sumBytes
| fit DensityFunction avgBytes by bot_type as anomaly_avgBytes into module1_avgBytes
| fit DensityFunction avgDuration by bot_type as anomaly_avgDuration into
module1_avgDuration
| fit DensityFunction uniqueDstIp by bot_type as anomaly_uniqueDstIp into
module1_uniqueDstIp
| fit DensityFunction uniqueDstPort by bot_type as anomaly_uniqueDstPort into
module1_uniqueDstPort
| fit DensityFunction uniqueProtocols by bot_type as anomaly_uniqueProtocols into
module1_uniqueProtocols

```

**New Search**

```

index=botnet sourcetype=netflow_botnet botnet=1
| fit DensityFunction numFlows by bot_type as anomaly_numFlows into module1_numFlows
| fit DensityFunction sumBytes by bot_type as anomaly_sumBytes into module1_sumBytes
| fit DensityFunction avgBytes by bot_type as anomaly_avgBytes into module1_avgBytes
| fit DensityFunction avgDuration by bot_type as anomaly_avgDuration into module1_avgDuration
| fit DensityFunction uniqueDstIp by bot_type as anomaly_uniqueDstIp into module1_uniqueDstIp
| fit DensityFunction uniqueDstPort by bot_type as anomaly_uniqueDstPort into module1_uniqueDstPort
| fit DensityFunction uniqueProtocols by bot_type as anomaly_uniqueProtocols into module1_uniqueProtocols

```

Too few training points in some groups will likely result in poor accuracy for those groups. Please see model summary to inspect such groups.

5,608 events (8/10/11 8:46:00.000 AM to 4/22/24 8:12:01.000 PM) No Event Sampling

Events (5,608) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect 1 year per column

Time	Event
8/19/11 10:44:00.000 AM	1313750648, "147.32.84.191", 2, 416, "5.2985995", 2, 2, udp, 1, Botnet, 208, "nsis.ay", 1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet.csv sourcetype = netflow_botnet
8/19/11 10:44:00.000 AM	1313750648, "147.32.84.165", 5, 1565, "13.296201199999999", 5, 5, udp, 1, Botnet, 313, "nsis.ay", 1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet.csv sourcetype = netflow_botnet

- You can view the specific model information for any trained models with the `summary` command. Run the following search to view the details of the `module1_numFlows` model. Feel free to repeat this throughout the workshop to view specific trained model parameters.

```
| summary module1_numFlows
```

**New Search** Save As Create Table View Close

| summary module\_numFlows All time Q

✓ 7 results (1/1/70 12:00:00.000 AM to 4/22/24 8:15:39.000 PM) No Event Sampling Job II III IV V Smart Mode

Events (0) Patterns **Statistics (7)** Visualization

20 Per Page Format Preview

bot_type	type	min	max	mean	std	cardinality	distance	other
menti	Auto: Gaussian KDE	2	62	37.642276422764226	10.186975648674139	123	metric: wasserstein, distance: 1.2282318341254344	bandwidth: 3.8910270117918455, parameter size: 123
murlo	Auto: Gaussian KDE	1	51	5.60146252285192	11.265462161614792	1094	metric: wasserstein, distance: 2.13669107519511	bandwidth: 2.779364773292085, parameter size: 1094
neris	Auto: Gaussian KDE	1	1738	130.56478053939713	111.51666031806609	1891	metric: wasserstein, distance: 6.9586888056125575	bandwidth: 24.608475658692825, parameter size: 1891
nsis.ay	Auto: Gaussian KDE	1	303	20.64761904761905	53.32952027185604	105	metric: wasserstein, distance: 14.655615371761751	bandwidth: 21.02470046351314, parameter size: 105
rbot	Auto: Beta	1	5718	-7.02766701106124e-30	15.329743005889315	1391	metric: wasserstein, distance: 83.50079683560685	Alpha: 0.4447624099370552, Beta: 859.753103909027
sogou	Auto: Beta	1	49	-0.09871650810625836	1.0987165881062585	5	metric: wasserstein, distance: 5.842944966034938	Alpha: 0.014234436360368505, Beta: 0.048718291842217556
virut	Auto: Gaussian KDE	2	154	40.94454494494494	22.599396113096667	999	metric: wasserstein, distance: 2.133823964248027	bandwidth: 5.67784765971427, parameter size: 999

## Task 3: Model Testing

Now that the models have been trained, it's time to test them to see how they perform against the test dataset! There is a dashboard in the Botnet App for Splunk, but you will be re-creating many of those panels in SPL.

1. In the browser tab open to the Botnet App for Splunk, expand the options under 3. Anomaly Detection, and open the 3.1 Training the Anomaly Detection Models dashboard. In the app, this is the dashboard that is used to test the anomaly detection models.

*Note the confusion matrix spread across the last four panels.*

**3. Anomaly Detection**

✓ 3.1. Train Anomaly Detection Models

3.1. Train Anomaly Detection Models

3.2. Model Summaries

3.3. Testing the Anomaly Detection Models

1. BACKGROUND ON MACHINE LEARNING

2. DATA ANALYSIS & PREPARATION

3. ANOMALY DETECTION

4. CLASSIFICATION

5. NEXT STEPS

3.1. TRAIN ANOMALY DETECTION MODELS

← Back

Next →

**Training Models**

On this dashboard you can train a set of anomaly detection models to baseline normal traffic for each botnet type. Note that here we are training models to understand the behaviour of the botnet, whereas in production you would be more likely to train a model to understand the baseline for normal traffic.

**Currently Selected Pre-Processing Options**

These are the most recent selections that have been saved on 2.3. Pre Processing.

Scaling Applied: PCA Applied: Outlier Detection Applied: Train/Test Split:

10.202.22.53:8000/en-US/app/botnet-app-for-splunk/2\_train\_anomaly\_detection\_models

2. In the browser tab open to the Search & Reporting app, recreate the confusion matrix from the dashboard by running the following search.

*In our testing, this search took about 2 minutes to complete.*

*Note how the search is scoped to a single day by setting the earliest and latest fields in the first search command. This was done to reduce the dataset size needing to run through all seven models, to reduce the time it takes for this search to run.*

*The search works by trying to identify any anomalous network flows, then checking if those flows are from botnets or not. Keep in mind that these models are trained off of what “normal” looks like for a botnet, so anything identified as anomalous is most likely normal/non-botnet traffic.*

- *If the anomaly score is zero, and the traffic originates from a botnet, it is a true positive (correctly identified as coming from a botnet)*
- *If the anomaly score is zero, and the traffic did not originate from a botnet, it is a false positive (incorrectly identified as coming from a botnet when it was normal traffic)*
- *If the anomaly score is greater than zero, and the traffic did not originate from a botnet, it is a true negative (correctly identified as normal traffic)*
- *If the anomaly score is greater than zero, and the traffic did originate from a botnet, it is a false negative (incorrectly identified as normal traffic but it originated from a botnet)*

```
index=botnet sourcetype=netflow_botnet_balanced earliest=1313020800 latest=1313107200
| apply module1_numFlows
| apply module1_sumBytes
| apply module1_avgBytes
| apply module1_avgDuration
| apply module1_uniqueDstIp
| apply module1_uniqueDstPort
| apply module1_uniqueProtocols
| eval anomaly_score =
anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_unique
DstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| eval tp=if(anomaly_score=0 AND botnet=1,1,0)
| eval fp=if(anomaly_score=0 AND botnet=0,1,0)
| eval tn=if(anomaly_score>0 AND botnet=0,1,0)
| eval fn=if(anomaly_score>0 AND botnet=1,1,0)
| stats sum(tp) as true_positives, sum(fn) as false_negatives, sum(tn) as
true_negatives, sum(fp) as false_positives
```

**New Search**

```

index=botnet sourcetype=netflow_botnet_balanced earliest=1313020800 latest=1313107200
| apply module1_numFlows
| apply module1_sumBytes
| apply module1_avgBytes
| apply module1_avgDuration
| apply module1_uniqueDstIp
| apply module1_uniqueDstPort
| apply module1_uniqueProtocols
| eval anomaly_score = anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_uniqueDstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| eval tp=if(anomaly_score=0 AND botnet=1,1,0)
| eval fp=if(anomaly_score=0 AND botnet=0,1,0)
| eval tn=if(anomaly_score>0 AND botnet=0,1,0)
| eval fn=if(anomaly_score>0 AND botnet=1,1,0)
| stats sum(tp) as true_positives, sum(fn) as false_negatives, sum(tn) as true_negatives, sum(fp) as false_positives

```

Too few training points in some groups will likely result in poor accuracy for those groups. Please see model summary to inspect such groups.

✓ 2,746 events (8/11/11 12:00:00.000 AM to 8/12/11 12:00:00.000 AM) No Event Sampling

Events Patterns **Statistics (1)** Visualization

20 Per Page Format Preview

true_positives	false_negatives	true_negatives	false_positives
196	8	827	1715

- Those scores aren't great, but what happens if a custom threshold parameter is defined (default is 0.01)? Run the following two searches; the first one having a threshold of 0.05, and the second a parameter of 0.1.

*Just another note that each of these searches took about 2 minutes to run while testing this workshop.*

```

index=botnet sourcetype=netflow_botnet_balanced earliest=1313020800 latest=1313107200
| apply module1_numFlows threshold=0.05
| apply module1_sumBytes threshold=0.05
| apply module1_avgBytes threshold=0.05
| apply module1_avgDuration threshold=0.05
| apply module1_uniqueDstIp threshold=0.05
| apply module1_uniqueDstPort threshold=0.05
| apply module1_uniqueProtocols threshold=0.05
| eval anomaly_score =
anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_unique
DstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| eval tp=if(anomaly_score=0 AND botnet=1,1,0)
| eval fp=if(anomaly_score=0 AND botnet=0,1,0)
| eval tn=if(anomaly_score>0 AND botnet=0,1,0)
| eval fn=if(anomaly_score>0 AND botnet=1,1,0)
| stats sum(tp) as true_positives, sum(fn) as false_negatives, sum(tn) as
true_negatives, sum(fp) as false_positives

```

```

index=botnet sourcetype=netflow_botnet_balanced earliest=1313020800 latest=1313107200
| apply module1_numFlows threshold=0.1
| apply module1_sumBytes threshold=0.1
| apply module1_avgBytes threshold=0.1

```

```

| apply module1_avgDuration threshold=0.1
| apply module1_uniqueDstIp threshold=0.1
| apply module1_uniqueDstPort threshold=0.1
| apply module1_uniqueProtocols threshold=0.1
| eval anomaly_score =
anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_unique
DstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| eval tp=if(anomaly_score=0 AND botnet=1,1,0)
| eval fp=if(anomaly_score=0 AND botnet=0,1,0)
| eval tn=if(anomaly_score>0 AND botnet=0,1,0)
| eval fn=if(anomaly_score>0 AND botnet=1,1,0)
| stats sum(tp) as true_positives, sum(fn) as false_negatives, sum(tn) as
true_negatives, sum(fp) as false_positives

```

*Note how much better the results look in the confusion matrix! Most of the time it's worth tuning and testing different hyperparameter settings because it can greatly improve model accuracy.*

The screenshot shows the Splunk Enterprise interface. At the top, there's a navigation bar with 'splunk>enterprise' and various user and system links. Below this is a 'New Search' panel. The search query is pasted into the input field and is highlighted with a red box. The query is the same as the one in the first block. Below the search bar, there's a warning message: 'Too few training points in some groups will likely result in poor accuracy for those groups. Please see model summary to inspect such groups.' Below the warning, it shows '2,746 events' and 'No Event Sampling'. At the bottom, there's a table with four columns: 'true\_positives', 'false\_negatives', 'true\_negatives', and 'false\_positives'. The values are 187, 17, 2536, and 6 respectively. The table is highlighted with a blue box.

```

index=botnet sourcetype=netflow_botnet_balanced earliest=1313020800 latest=1313107200
| apply module1_numFlows threshold=0.05
| apply module1_sumBytes threshold=0.05
| apply module1_avgBytes threshold=0.05
| apply module1_avgDuration threshold=0.05
| apply module1_uniqueDstIp threshold=0.05
| apply module1_uniqueDstPort threshold=0.05
| apply module1_uniqueProtocols threshold=0.05
| eval anomaly_score = anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_uniqueDstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| eval tp=if(anomaly_score=0 AND botnet=1,1,0)
| eval fp=if(anomaly_score=0 AND botnet=0,1,0)
| eval tn=if(anomaly_score>0 AND botnet=0,1,0)
| eval fn=if(anomaly_score>0 AND botnet=1,1,0)
| stats sum(tp) as true_positives, sum(fn) as false_negatives, sum(tn) as true_negatives, sum(fp) as false_positives

```

Too few training points in some groups will likely result in poor accuracy for those groups. Please see model summary to inspect such groups.

✓ 2,746 events (8/11/11 12:00:00 AM to 8/12/11 12:00:00 AM) No Event Sampling

Events Patterns **Statistics (1)** Visualization

20 Per Page Format Preview

true_positives	false_negatives	true_negatives	false_positives
187	17	2536	6



The screenshot shows the Splunk Enterprise interface. The search command is as follows:

```
index=botnet sourcetype=netflow_botnet_balanced earliest=1313020800 latest=1313107200
| apply module1_numFlows threshold=0.1
| apply module1_sumBytes threshold=0.1
| apply module1_avgBytes threshold=0.1
| apply module1_avgDuration threshold=0.1
| apply module1_uniqueDstIp threshold=0.1
| apply module1_uniqueDstPort threshold=0.1
| apply module1_uniqueProtocols threshold=0.1
| eval anomaly_score = anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_uniqueDstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| eval tp=if(anomaly_score<0 AND botnet=1,1,0)
| eval fp=if(anomaly_score=0 AND botnet=0,1,0)
| eval tn=if(anomaly_score>0 AND botnet=0,1,0)
| eval fn=if(anomaly_score>0 AND botnet=1,1,0)
| stats sum(tp) as true_positives, sum(fn) as false_negatives, sum(tn) as true_negatives, sum(fp) as false_positives
```

The results table shows the following data:

true_positives	false_negatives	true_negatives	false_positives
130	74	2542	8

## Task 4: Model Application

Now that the models are performing well, it's time to apply them.

1. An example search showing how the models can be applied is below. This search can be used to create an alert, or even an ES correlation search to identify hosts potentially communicating with botnets based on the observed network traffic. Any event that has an anomaly score below 1 is likely traffic originating from botnets.

*Notice again how the time range is scoped down to 1 day using the earliest and latest fields in the first search command. Again, this was just done to reduce the time workshop attendees would be sitting and waiting for searches to finish.*

*It's worth noting again that since the anomaly detection models are trained to specifically identify botnet traffic, any traffic that is not anomalous is likely to be botnet traffic.*

```
index=botnet sourcetype=netflow_botnet_balanced earliest=1313107200 latest=1313193600
| apply module1_numFlows threshold=0.05
| apply module1_sumBytes threshold=0.05
| apply module1_avgBytes threshold=0.05
| apply module1_avgDuration threshold=0.05
| apply module1_uniqueDstIp threshold=0.05
| apply module1_uniqueDstPort threshold=0.05
| apply module1_uniqueProtocols threshold=0.05
| eval anomaly_score =
anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_unique
DstIp+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| search anomaly_score<1
```

splunk>enterprise Apps Administrator Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

### New Search

Save As Create Table View Close

```

index=botnet sourcetype=netflow_botnet_balanced earliest=1313187200 latest=1313193600
| apply module1_numFlows threshold=0.05
| apply module1_sumBytes threshold=0.05
| apply module1_avgBytes threshold=0.05
| apply module1_avgDuration threshold=0.05
| apply module1_uniqueDstIP threshold=0.05
| apply module1_uniqueDstPort threshold=0.05
| apply module1_uniqueProtocols threshold=0.05
| eval anomaly_score = anomaly_numFlows+anomaly_sumBytes+anomaly_avgBytes+anomaly_avgDuration+anomaly_uniqueDstIP+anomaly_uniqueDstPort+anomaly_uniqueProtocols
| search anomaly_score<1

```

Too few training points in some groups will likely result in poor accuracy for those groups. Please see model summary to inspect such groups.

✓ 242 events (8/12/11 12:00:00.000 AM to 8/13/11 12:00:00.000 AM) No Event Sampling Job

Events (242) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect 1 hour per column

List Format 20 Per Page

< Hide Fields All Fields

SELECTED FIELDS

@ host 1

Time	Event
8/12/11 11:59:00.000 PM	1313193540,5530,5.886111,rbot,0,tcp,1,66.249.72.52,5530,Background,1,1,1 host = so1 source = /opt/splunk/etc/system/local/netflow_botnet_balanced.csv sourcetype = netflow_botnet_balanced

That's all for Module 1! In this module, you learned how to train machine learning models to identify botnet and normal traffic.



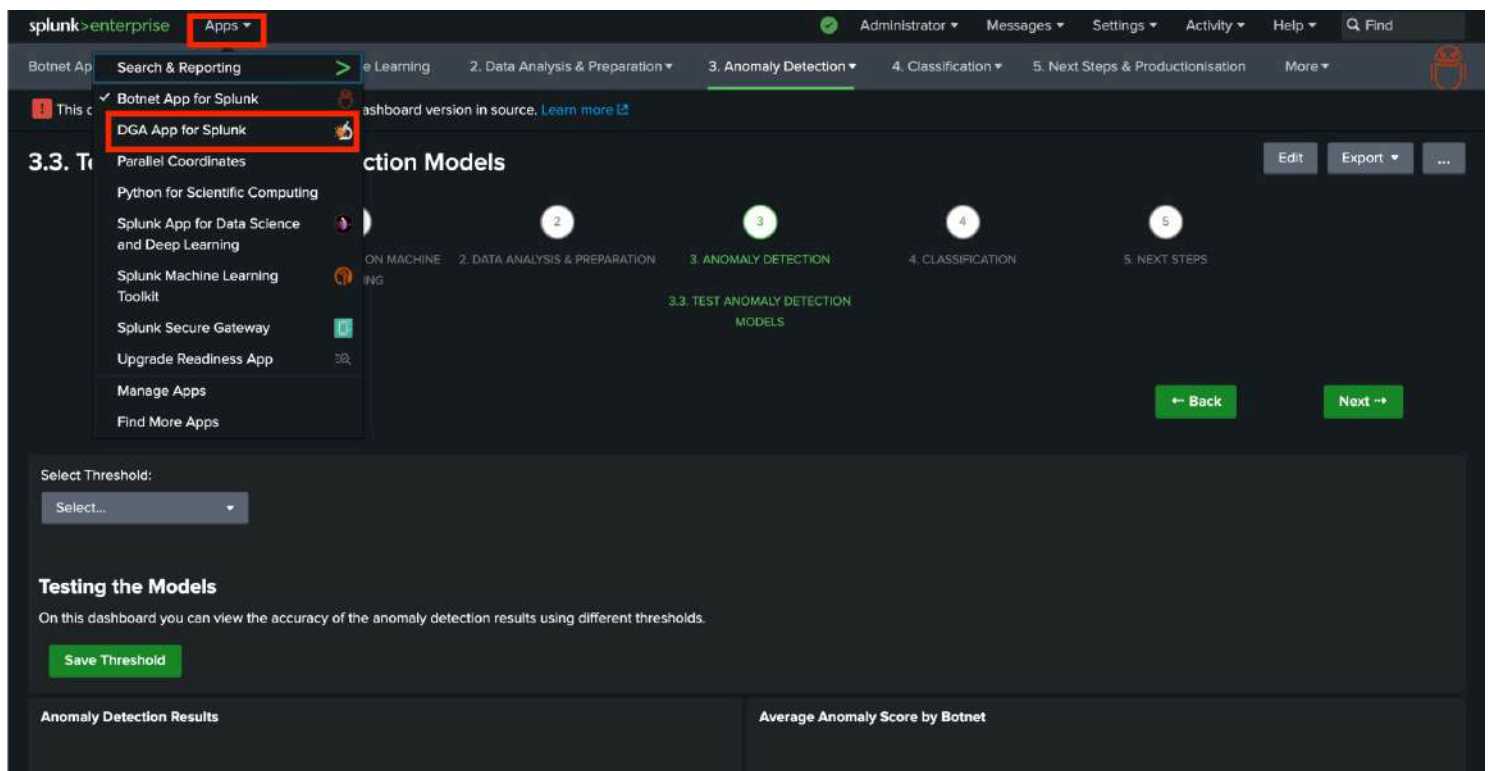
## Module 2: Clustering

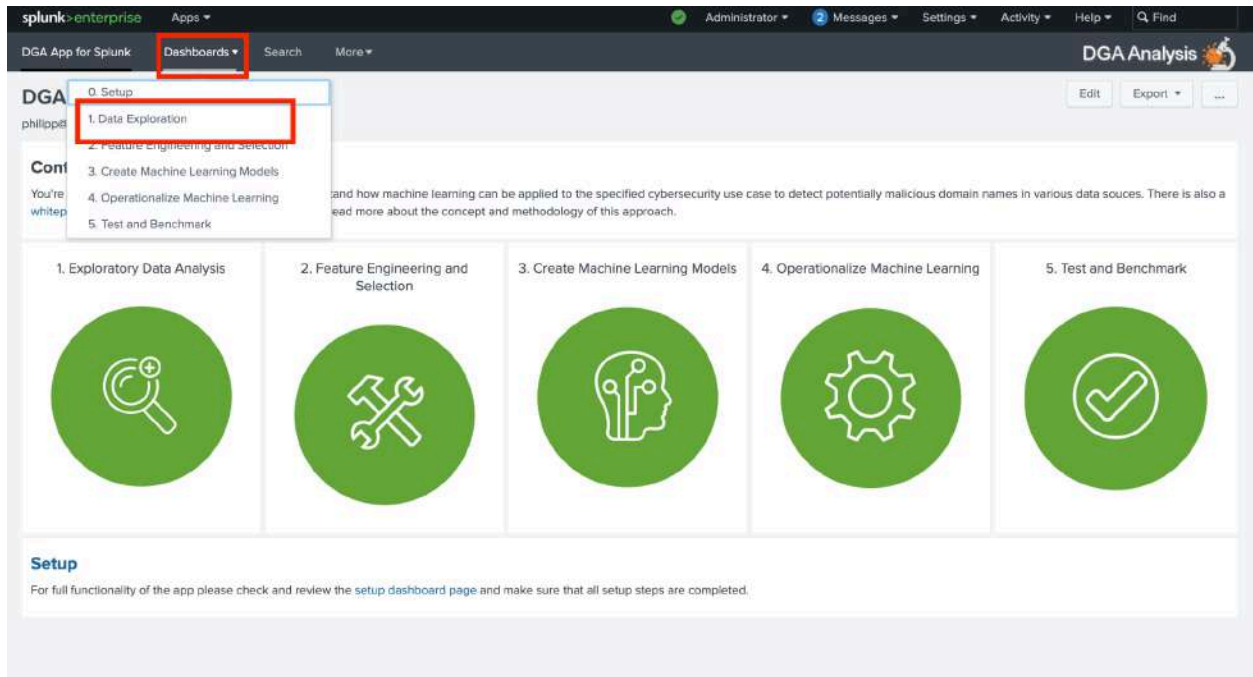
In this module, you will be working with domain names that are being communicated with from hosts in a network. Many of these domains have been generated by algorithms (aka Domain Generating/Generated Algorithms or DGAs). This module is loosely based on the [DGA App for Splunk](#) in that it uses data sets originating from and some of the dashboards from that app. Like the Botnet App for Splunk from Module 1, this app is valuable to show what is possible in regards to machine learning with Splunk. The goal of this module is to conduct feature engineering and exploration, then unsupervised learning with clustering.

### Task 1: Data Exploration

In this task, you will be exploring the data that will be used in this module and Module 3. First, you will look at the raw data, then the results of N-gram analysis conducted in the DGA App for Splunk. After that you'll re-create the N-gram analysis in SPL. All of the data is in the `dga` index, with the training dataset set as the `dga_domains` sourcetype, and the testing dataset (used in Module 3) set as the `dga_test` sourcetype. This dataset is nearly identical to the dataset referenced in the DGA App for Splunk, but a semi-randomly generated timestamp has been added to each event from the first week of April 2024.

1. In the browser tab open to the Botnet App for Splunk from Module 1, navigate to the **DGA App for Splunk**, then expand the **Dashboards** dropdown and select the **1. Data Exploration** dashboard.

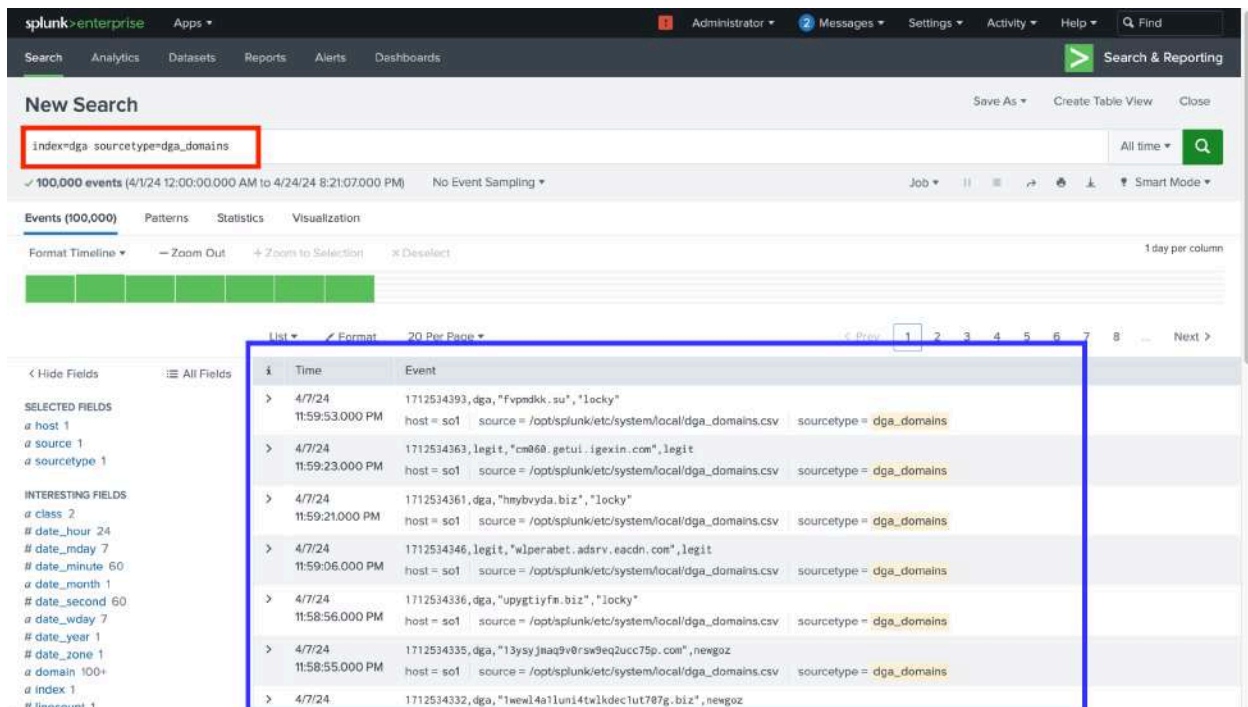




2. In the browser tab open to the Search & Reporting app, run the following search to show the raw data that this module and the next one will be referencing.

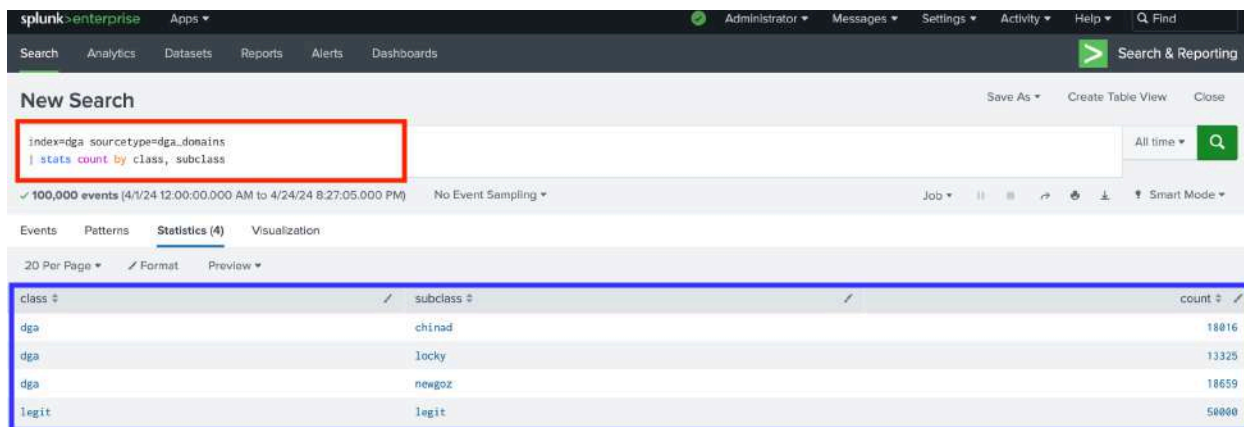
`index=dga sourcetype=dga_domains`

*Note that each event consists of a timestamp, a domain, and is labeled with information related to whether the domain is a legitimate domain, or one generated by an algorithm. If it is generated with an algorithm, the event also specifies which algorithm generated the data.*



- Run the following search to view the count of events stratified by class (whether the domain is algorithmically-generated or legitimate) and subclass (which DGA created the domain if it is algorithmically-generated). This is similar to the upper-right panel on the **Dataset Overview** panel in the DGA App for Splunk's 1. Data Exploration dashboard.

```
index=dga sourcetype=dga_domains
| stats count by class, subclass
```



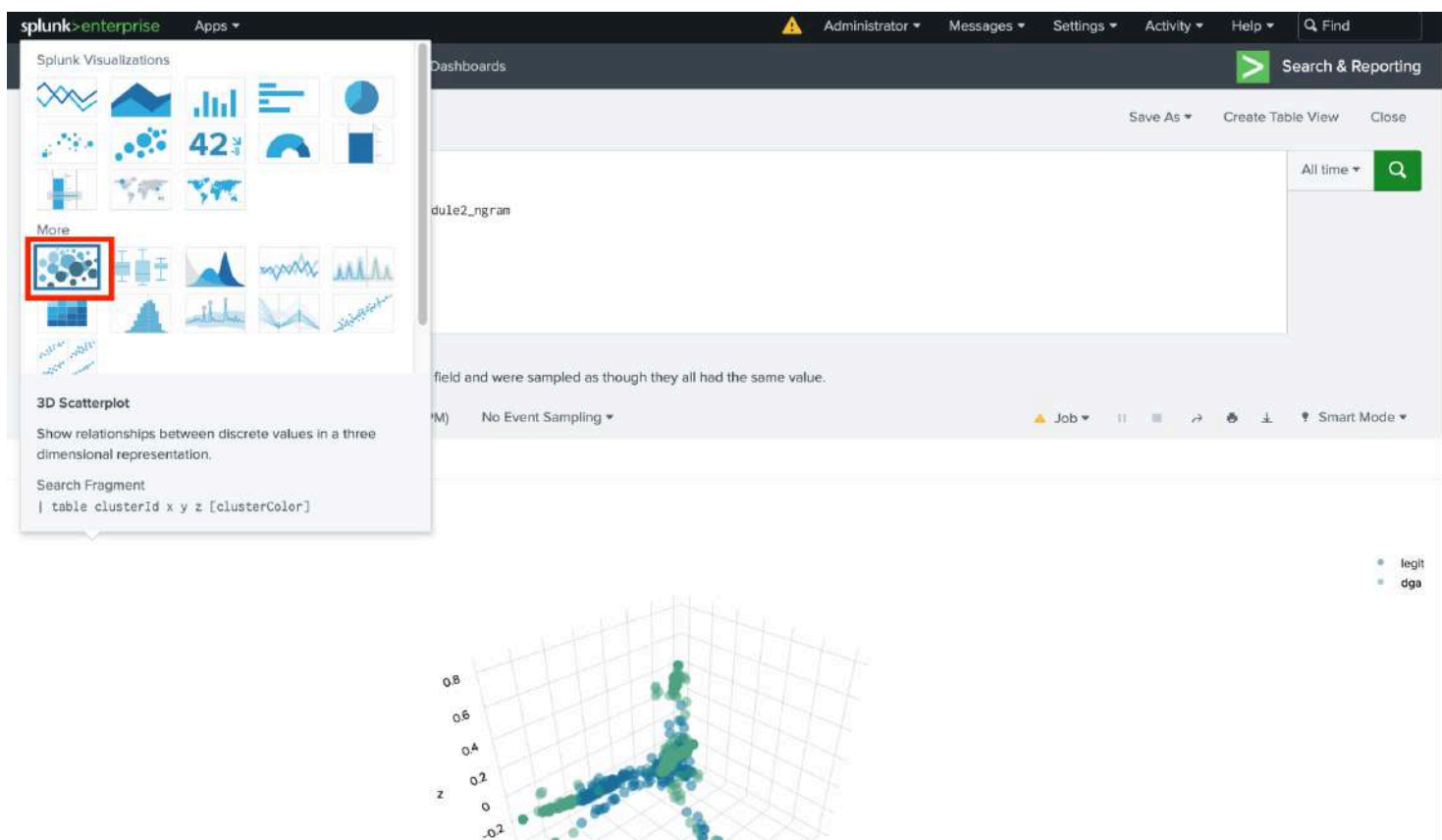
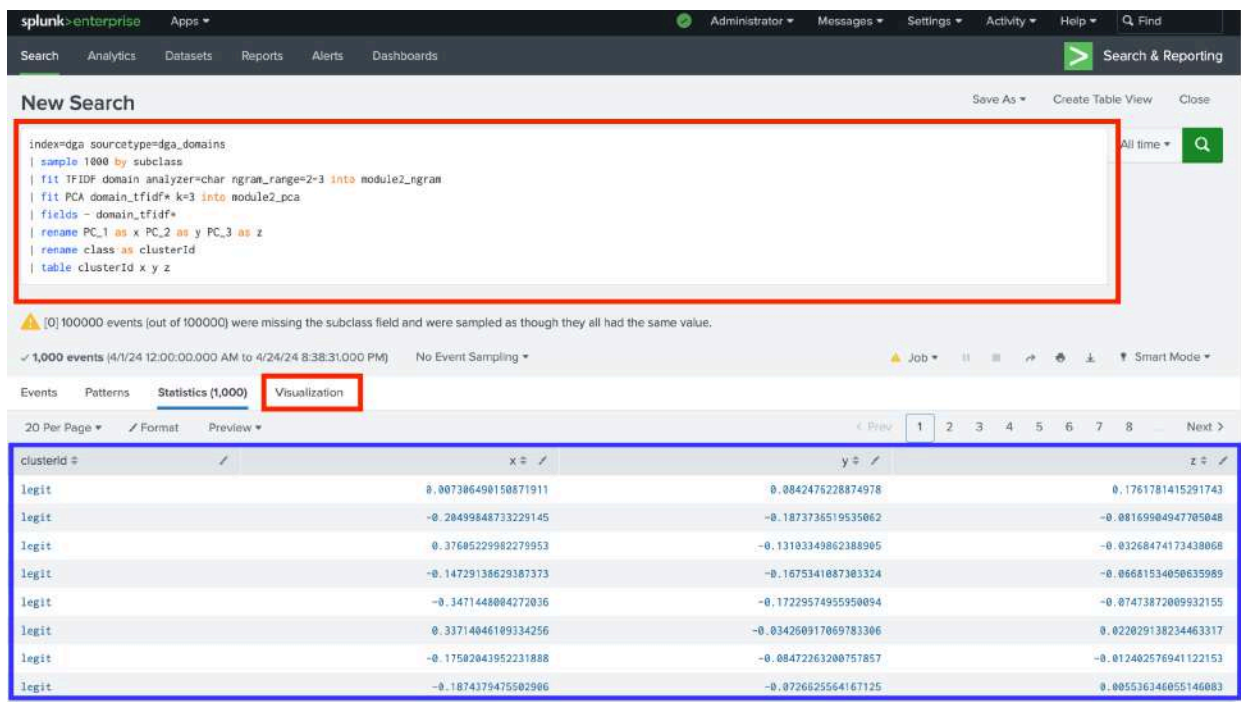
- Run the search below to conduct an N-gram analysis of the domains, stratified by subclass, sampling by subclass, and (after visualizing the results) coloring based on class. After the search runs, graph the data in the **3D Scatterplot** visualization. This visualization is replicating the n-gram analysis (2-3 char groups) of domain names with PCA k=3 by class panel in the DGA App for Splunk.

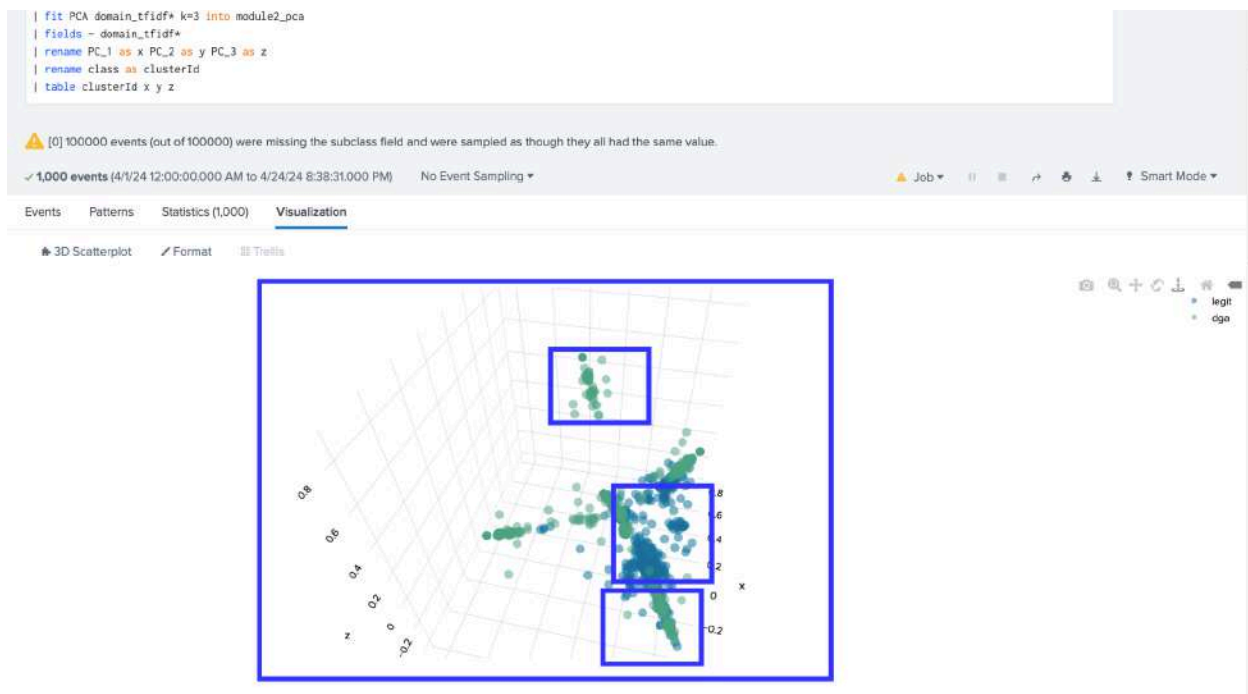
```
index=dga sourcetype=dga_domains
| sample 1000 by subclass
| fit TFIDF domain analyzer=char ngram_range=2-3 into module2_ngram
| fit PCA domain_tfidf* k=3 into module2_pca
| fields - domain_tfidf*
| rename PC_1 as x PC_2 as y PC_3 as z
| rename class as clusterId
| table clusterId x y z
```

*This search is using TFIDF (the Splunk MLTK's implementation of scikit-learn's TfidfVectorizer algorithm) to create new features in the data, then using PCA for dimensionality reduction. The reduced dimensions are then set as the X, Y, Z, coordinates on the visualization.*

*You can click and rotate the graph to get a better view of the data. You can also scroll in and out of the visualization to zoom in and out. Just keep in mind that if you click the graph it will re-run the search with a filter for that specific area you clicked, and might change the results.*

*Note how there are clusters of domains with different colors, representing clusters of both legitimate and DGA domains.*

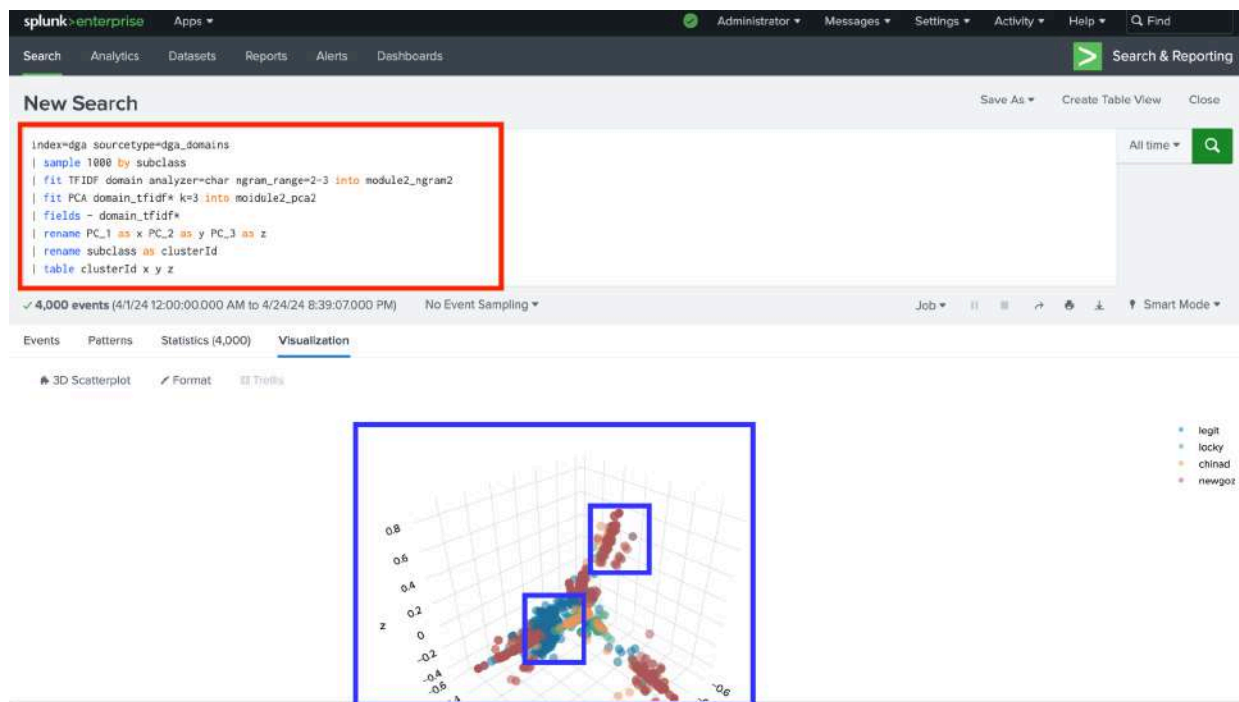




5. Similarly, run the search below to conduct an N-gram analysis of the domains, but coloring based on subclass this time. Likewise, this search is reconstructing the n-gram analysis (2-3 char groups) of domain names with PCA k=3 by subclass panel. Again, visualize the data using the **3D Scatterplot** visualization to make the data easier to interpret.

```
index=dga sourcetype=dga_domains
| sample 1000 by subclass
| fit TFIDF domain analyzer=char ngram_range=2-3 into module2_ngram2
| fit PCA domain_tfidf* k=3 into module2_pca
| fields - domain_tfidf*
| rename PC_1 as x PC_2 as y PC_3 as z
| rename subclass as clusterId
| table clusterId x y z
```

*Note how even when stratifying and coloring by subclass there are clear groupings. These groupings might be easier to see after rotating the graph.*



Since it's clear there are groupings or clusters of domains, we should be able to cluster the data based on additional features.

## Task 2: Feature Engineering

In this module you will create new features that ML models can train from. Specifically, you'll be making extensive use of [URL Toolbox](#), an app with custom SPL commands that can analyze domains and URLs. The goal of feature engineering here is to translate the domain into numeric features that can be used in training ML models.

1. In the browser tab open to the Botnet App for Splunk from Module 1, navigate to the **DGA App for Splunk**, then expand the **Dashboards** dropdown and select the **2. Feature Engineering and Selection** dashboard.
2. Run the following search to use URL Toolbox to add a [shannon entropy](#), meaning, and [Bayesian analysis](#) score to the dataset:

```
index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| table domain, class, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian
```

*Note how a `ut_shannon`, `ut_meaning_ratio`, and `ut_bayesian` scores have been added to each event.*



**New Search**

```
index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| table domain, class, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian
```

2,000 events (4/1/24 12:00:00.000 AM to 4/24/24 8:59:09.000 PM) No Event Sampling

Events Patterns **Statistics (2,000)** Visualization

20 Per Page Format Preview

domain	class	subclass	ut_shannon	ut_meaning_ratio	ut_bayesian
1qdvxl3t0q13act.org	dga	chinad	3.9219280948873623	0.25	0.999997371517718
zguzjqajdqsw1uk3ut1clir0t5.net	dga	newgoz	4.254195650150781	0.1724137931034483	0.999999918569825
1zfcsrgoz7y23cin.ru	dga	chinad	3.932138039759377	0.18526315789473684	0.997553165875814
a79hnyomtkb1946.org	dga	chinad	4.1219280948873624	0.15	0.998753135236821
m9hnyqzpw0ko0eb3.info	dga	chinad	3.975418017913833	0.09523809523809523	0.9999976170077138
ytyvtexuqagtttyehd.click	dga	locky	3.7950885863977324	0.2608695652173913	0.997579614353614
capitaloneservices.dendex.net	legit	legit	3.7714378294611266	0.6896551724137931	0.989294757230747
kdblayeahirvq.work	dga	locky	3.8268748818646396	0.5263157894736842	0.9952754241501632
7n7dsc8qv3ba06z8.cn	dga	chinad	3.82687488186464	0.05263157894736842	0.999999484574976
lyy6jxxvrxcom1web1m2zj6b.net	dga	newgoz	4.0062389286533895	0.2	0.999999438144888
1xfv2act1jonr716nckqyadjb1.org	dga	newgoz	4.349191770915039	0.28689655172413793	0.999999288824187

- Run the following search to also add digit, consonant, and vowel counts and percentage values to the dataset.

```
index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count,
ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count,
domain_digit_perc
```

*Note how fields related to statistics around numeric, vowel, and consonant in the domain have been added to each event. You may need to scroll to the right in the **Statistics** table to view these new fields.*

**New Search** Save As Create Table View Close

```

index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count, domain_digit_perc

```

2,000 events (1/24 12:00:00.000 AM to 4/24/24 9:46:35.000 PM) No Event Sampling

Events Patterns **Statistics (2,000)** Visualization

20 Per Page Format Preview < Prev 1 2 3 4 5 6 7 8 Next >

ibcless	ut_shannon	ut_meaning_ratio	ut_bayesian	ut_vowel_count	ut_consonant_count	domain_consonant_perc	domain_vowel_perc	ut_digit_count	domain_digit_perc
tinad	3.7216117239699025	0.05263157894736842	0.9999999937638745	4	7	0.3684210526315789	0.21052631578947367	7	0.3684210526315789
rego2	4.349191770915039	0.034482758620689655	0.9999786499439476	3	16	0.5517241379310345	0.10344827586206896	9	0.3103448275862069
icky	2.7773627950641693	0.07692307692307693	0.9742865855832492	3	9	0.6923076923076923	0.23076923076923078	0	0
rego2	4.324848729602135	0.06451612083225806	0.99999998755098	4	17	0.5483870967741935	0.12903225806451613	9	0.2903225806451613
icky	3.3082708345352603	0.11111111111111111	0.9881684527464919	4	13	0.7222222222222222	0.2222222222222222	0	0

- Run the following search to reduce the dimensions on the dataset using Principal Component Analysis (PCA), down from 9 dimensions down to 3. This search replicates the Domain dataset enriched with features panel on the 2. Feature Engineering and Selection dashboard in the DGA App for Splunk.

```

index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count,
ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count,
domain_digit_perc

```



```
| fit PCA "ut_shannon", "ut_meaning_ratio", "ut_bayesian", "ut_vowel_count",
"ut_consonant_count", "domain_consonant_perc", "domain_vowel_perc", "ut_digit_count",
"domain_digit_perc" k=3 into module2_pca3
| table subclass, ut*, PC*
```

Note how three new fields have been added: `PC_1`, `PC_2`, `PC_3`. These are the reduced dimensions the PCA algorithm created.

New Search Save As Create Table View Close

```
index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count, domain_digit_perc
| fit PCA "ut_shannon", "ut_meaning_ratio", "ut_bayesian", "ut_vowel_count", "ut_consonant_count", "domain_consonant_perc", "domain_vowel_perc", "ut_digit_count", "domain_digit_perc" k=3 into module2_pca3
| table class, subclass, ut*, PC*
```

✓ 2,000 events (4/1/24 12:00:00.000 AM to 4/24/24 9:48:56.000 PM) No Event Sampling Job Smart Mode

Events Patterns **Statistics (2,000)** Visualization

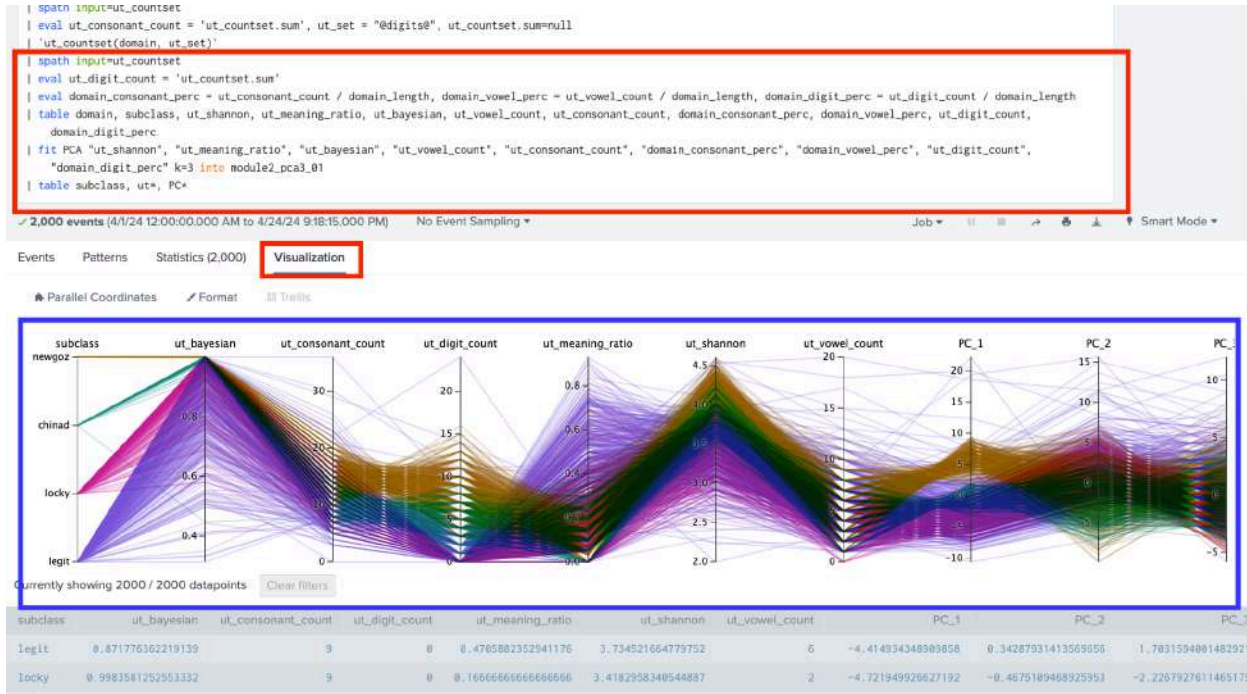
20 Per Page Format Preview < Prev 1 2 3 4 5 6 7 8 Next >

subclass	ut_bayesian	ut_consonant_count	ut_digit_count	ut_meaning_ratio	ut_shannon	ut_vowel_count	PC_1	PC_2	PC_3
locky	0.9994360903406596	13	0	0.17647058823529413	3.8521687236032816	3	-2.124862914886955	2.5342633300449573	-2.190155735091889
locky	0.9957313713544159	11	0	0.3333333333333333	3.640223928941851	3	-3.4209431890753343	1.0891612105450457	-1.6736217744171482
newgoz	0.9999996298413152	17	6	0.13793103448275862	4.306256857196538	5	5.197288856939322	2.147362618317813	-0.6143279297649785

- Run the following search, then visualize the data with the **Parallel Coordinates** visualization. This visualization is closely replicating the Parallel coordinate chart of classes and top features visualization on the 2. Feature Engineering and Selection dashboard in the DGA App for Splunk.

```
index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
```

```
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count, domain_digit_perc
| fit PCA "ut_shannon", "ut_meaning_ratio", "ut_bayesian", "ut_vowel_count", "ut_consonant_count", "domain_consonant_perc", "domain_vowel_perc", "ut_digit_count", "domain_digit_perc" k=3 into module2_pca3_01
| table subclass, ut*, PC*
```



*Note how it's easy to see relationships between feature values using this visualization. For example, the domains generated by the chinad and locky algorithms tend to have a high ut\_bayesian score.*

Now that the additional features have been engineered, it's time to move onto the clustering component of this module.

### Task 3: Clustering and Exploration

In this task, you will be using the features you engineered in Task 2 to cluster different domains together. There is no defined goal with clustering (unsupervised learning), but the idea is to see similarities between different classes by grouping them together in clusters.

1. Run the following search to cluster the data together, based on the PCA-reduced dimensions, and three possible clusters (as defined by the `K` hyperparameter).

```

index=dga sourcetype=dga_domains
| sample 1000 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, class, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian,
ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc,
ut_digit_count, domain_digit_perc
| fit PCA "ut_shannon", "ut_meaning_ratio", "ut_bayesian", "ut_vowel_count",
"ut_consonant_count", "domain_consonant_perc", "domain_vowel_perc", "ut_digit_count",
"domain_digit_perc" k=3 into module2_pca3
| fit KMeans k=3 "PC_1", "PC_2", "PC_3" into module2_kmeans

```

You might want to click back to the **Statistics** panel to see the numbers more easily.

Note that the `cluster` field has been added to each event.

**New Search** Save As Create Table View Close

```

index=dga sourcetype=dga_domains
| sample 1000 by subclass
| 'ut_shannon(domain)'
| 'ut_meaning(domain)'
| 'ut_bayesian(domain)'
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| 'ut_countset(domain, ut_set)'
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| 'ut_countset(domain, ut_set)'
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| 'ut_countset(domain, ut_set)'
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, class, subclass, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count, domain_digit_perc
| fit PCA "ut_shannon", "ut_meaning_ratio", "ut_bayesian", "ut_vowel_count", "ut_consonant_count", "domain_consonant_perc", "domain_vowel_perc", "ut_digit_count", "domain_digit_perc" k=3 into module2_pca3
| fit KMeans k=3 "PC_1", "PC_2", "PC_3" into module2_kmeans

```

✓ 4,000 events (4/1/24 12:00:00.000 AM to 4/24/24 9:28:10.000 PM) No Event Sampling

Events Patterns **Statistics (4,000)** Visualization

20 Per Page Format Preview

sonant_count	ut_digit_count	domain_consonant_perc	domain_vowel_perc	domain_digit_perc	PC_1	PC_2	PC_3	cluster	cluster_distance
9	0	0.5	0.4444444444444444	0.0	-4.2900276281413765	0.8702889750345314	3.708476844216092	1	21.283711512281975
12	11	0.4	0.2	0.36666666666666665	5.79695925634642	-4.666893881856867	1.9988884431764784	2	17.954870622981943
11	0	0.6875	0.25	0.0	-3.2956129480088316	1.4686790720681278	-0.6083250869594297	0	9.098411228701575

2. Run the following search to display the trained model parameters.

| `summary module2_kmeans`

*Note that the parameters are all related to the different fields PCA created.*

**splunk>enterprise** Apps Administrator Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

**New Search** Save As Create Table View Close

| `summary module2_kmeans` All time Find

✓ 4 results (1/1/70 12:00:00.000 AM to 4/24/24 9:32:27.000 PM) No Event Sampling

Events (0) Patterns **Statistics (4)** Visualization

20 Per Page Format Preview

PC_1	PC_2	PC_3	cluster	inertia
-0.7238897114612793	2.983036851672147	-0.17086498745806464	0	58424.978968169664
-3.097374858221189	-1.608020913233632	0.004523418429541276	1	
6.164235447137466	-0.8488227854737425	0.19824635613537847	2	

3. Run the search below, then view the results in a **3D Scatterplot** to view the clusters, colored by subclass.

```

index=dga sourcetype=dga_domains
| sample 1000 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module2_pca3
| apply module2_kmeans
| rename cluster as clusterId
| rename PC_1 as x, PC_2 as y, PC_3 as z
| table clusterId x y z

```

*Using the 3D Scatterplot visualization like this is a good way to view the clusters.*

*You may want to zoom in by scrolling and rotate the graph by clicking and dragging.*



In this module you created new features based off of the DGA dataset, and in the next module you'll be training classification models to predict whether a given domain name is algorithmically-generated, or legitimate, as well as which algorithm generated the domain.

## Module 3: Predictive Analysis

In this module, you will be using the same dataset and features you engineered from the previous module, but instead of clustering you will be training models to identify domain-generated algorithms. This can be a great way to identify hosts or users who are communicating with potentially malicious domains.

At first, you'll be training models to predict which algorithm generated the domain, but later on be training models to predict whether the domain is legitimate or algorithmically-generated. You will even be testing the prediction model against domains created by algorithms that aren't part of the training dataset!

### Task 1: Review Data and Engineered Features

In this task, you'll simply be reviewing the datasets and features engineered from Module 2.

1. In the browser tab open to the `Search & Reporting` app, run the following search to view the features that the models will be training against.

```
index=dga sourcetype=dga_domains
| sample 500 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, class, ut_shannon, ut_meaning_ratio, ut_bayesian,
ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc,
ut_digit_count, domain_digit_perc
```

*The `ut_shannon`, `ut_meaning_ratio`, `ut_bayesian`, `ut_vowel_count`, `ut_consonant_count`, `domain_consonant_perc`, `domain_vowel_perc`, `ut_digit_count`, `domain_digit_perc` are the features the models will be training from.*



splunk>enterprise Apps Administrator Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

### New Search

Save As Create Table View Close

```

index=dga sourcetype=dga_domains
| sample 500 by subclass
| 'ut_shannon(domain)'
| 'ut_meaning(domain)'
| 'ut_bayesian(domain)'
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| 'ut_countset(domain, ut_set)'
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| 'ut_countset(domain, ut_set)'
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| 'ut_countset(domain, ut_set)'
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| table domain, subclass, class, ut_shannon, ut_meaning_ratio, ut_bayesian, ut_vowel_count, ut_consonant_count, domain_consonant_perc, domain_vowel_perc, ut_digit_count, domain_digit_perc

```

2,000 events (4/1/24 12:00:00.000 AM to 5/2/24 8:53:51.000 PM) No Event Sampling

Events Patterns Statistics (2,000) Visualization

20 Per Page Format Preview

class	ut_shannon	ut_meaning_ratio	ut_bayesian	ut_vowel_count	ut_consonant_count	domain_consonant_perc	domain_vowel_perc	ut_digit_count	domain_digit_perc
dga	3.189898095464287	0.13333333333333333	0.8395760469213758	3	11	0.7333333333333333	0.2	0	0
dga	3.821928094887362	0.15	0.9986883054190594	3	12	0.6	0.15	4	0.2
dga	3.6818808028034047	0.0	0.999995946287398	1	10	0.5263157894736842	0.05263157894736842	7	0.3684210526315789
dga	3.418295834054489	0.08333333333333333	0.9961644682943221	3	8	0.6666666666666666	0.25	0	0
dga	3.690116517593666	0.17647058823529413	0.99638795296121	5	11	0.6470588235294118	0.29411764705882354	0	0

- Run the search below to identify the counts of domains generated in each of the datasets (training and test, set as sourcetype).

```

index=dga sourcetype=dga_domains OR sourcetype=dga_test
| stats count by sourcetype, subclass
| rename sourcetype as dataset, subclass as algorithm

```

*Note how there are more algorithms listed in the testing dataset than the training dataset. This will be important in Task 3.*



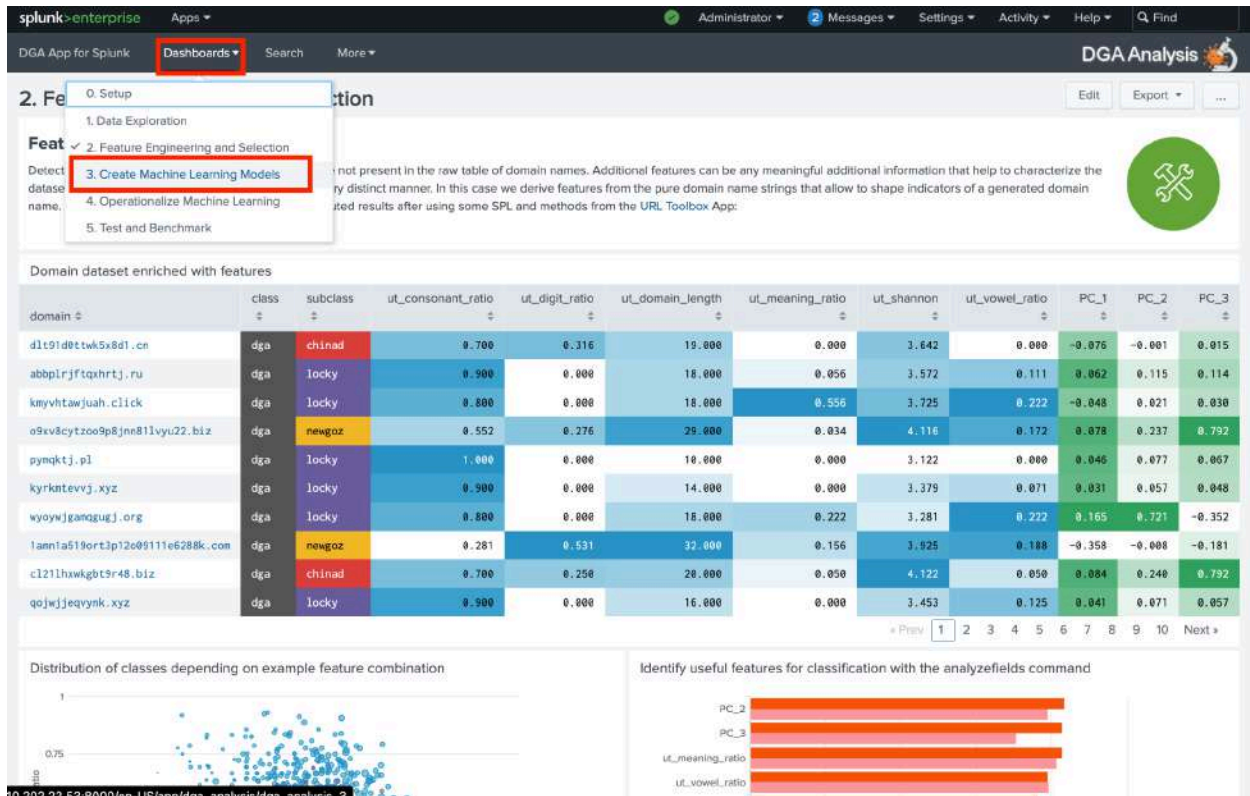
dataset	algorithm	count
dga_domains	chinad	18016
dga_domains	legit	50000
dga_domains	locky	13325
dga_domains	newgoz	18659
dga_test	chinad	30720
dga_test	corebot	120000
dga_test	dircrypt	30000
dga_test	dnscchanger	30000
dga_test	fobber	30000
dga_test	gozi	48000
dga_test	kraken_v1	48000
dga_test	kraken_v2	80000
dga_test	legit	1000000
dga_test	locky_v2	50000
dga_test	locky_v3	24000
dga_test	murofet_v1	30000
dga_test	murofet_v2	30000
dga_test	murofet_v3	30000
dga_test	necurs	28672
dga_test	newgoz	38000

## Task 2: Model Training, Testing, and Application for Algorithm Prediction

In this task, you will be training four different ML models against the training dataset to predict which algorithm generated the domain name (the `subclass` feature in the training and testing datasets). You will be training a random forest (RF), a support vector machine (SVM), a decision tree classifier (DCT), and a logistic regression (LR) model. It's often difficult to predict which machine learning algorithm will work best, which is why it can make sense to train multiple models at the same time then move forward with whichever performs the best.

1. In the browser tab open to the **DGA App for Splunk**, click **Dashboards** and select the **3. Create Machine Learning Models** dashboard. In the app, this is the dashboard that is used to train the models.

*If you receive an error about unsafe searches, feel free to ignore it.*



2. In the browser tab open to the Search & Reporting app, run the following search to train all four models at once.

```
index=dga sourcetype=dga_domains
| sample 3000 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| fit RandomForestClassifier subclass from ut_vowel_count, ut_consonant_count,
ut_digit_count, ut_meaning_ratio, ut_shannon, ut_bayesian into module3_rf
| fit SVM subclass from ut_vowel_count, ut_consonant_count, ut_digit_count,
ut_meaning_ratio, ut_shannon, ut_bayesian into module3_svm
```

```
| fit DecisionTreeClassifier subclass from ut_vowel_count, ut_consonant_count,  
ut_digit_count, ut_meaning_ratio, ut_shannon, ut_bayesian into module3_dtc  
| fit LogisticRegression subclass from ut_vowel_count, ut_consonant_count,  
ut_digit_count, ut_meaning_ratio, ut_shannon, ut_bayesian into module3_lr
```

*In our testing, this search took about 75s to complete.*

*Feel free to ignore any search warnings such as those related to the 10,000 event limit.*

*Note how in each of the `fit` commands, the search lists the subclass field as the feature to predict.*

*Notice how all of the features that were engineered in Module 2 are being used to train the models.*

*Also, notice how this search is training four different models in the same search.*

3. Run the following searches, each in turn to see the parameters for the models

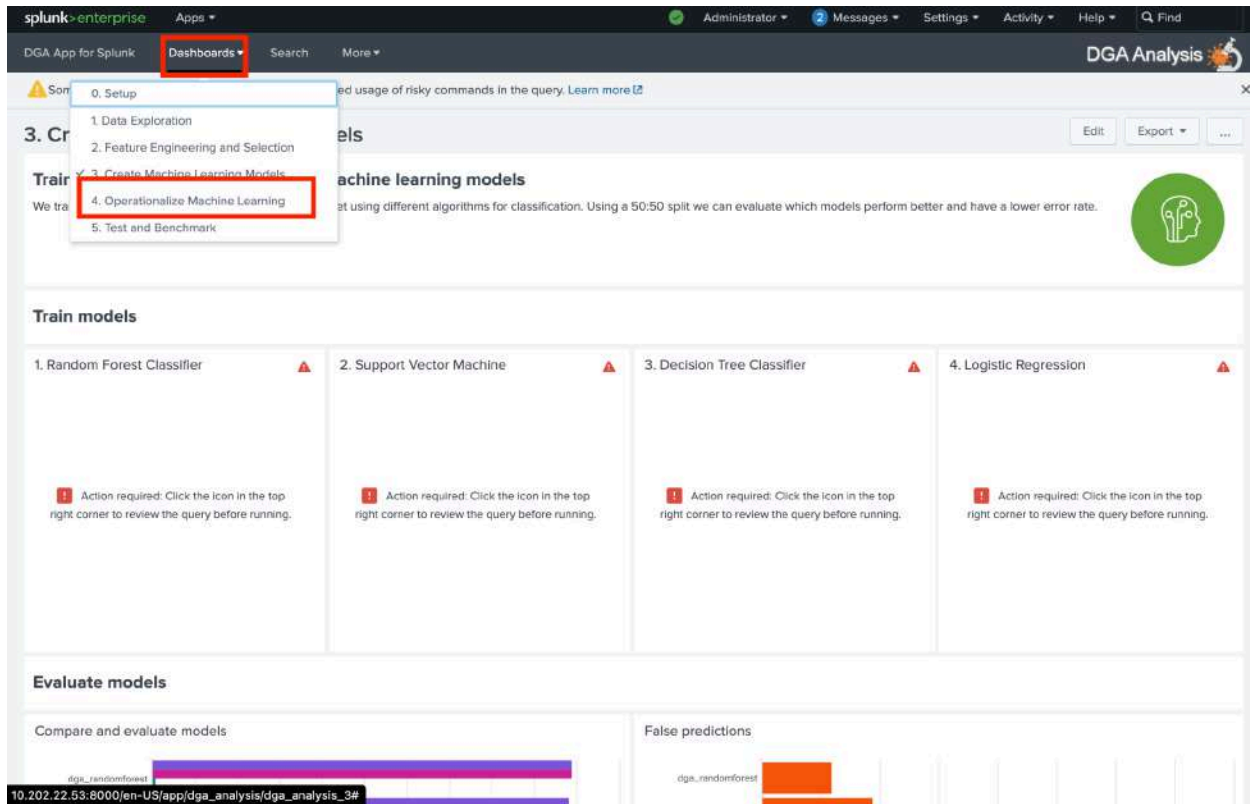
```
| summary module3_rf
```

```
| summary module3_dtc
```

```
| summary module3_lr
```

*Note that the SVM model does not support summary, which is why it wasn't listed above as one to view the parameters for.*

4. In the browser tab open to the **DGA App for Splunk**, click **Dashboards** and select the **4. Operationalize Machine Learning** dashboard. In the app, this is the dashboard that is used to test and operationalize the models predicting the algorithm generating the domain names.



5. Run the following search to test the four models that were trained earlier in this task

```
index=dga sourcetype=dga_test subclass=legit OR subclass=newgoz OR subclass=chinad OR
subclass=locky_v2 OR subclass=locky_v3 earliest=1711929600 latest=1711933200
| eval subclass=if(subclass="locky_v2" OR subclass="locky_v3", "locky", subclass)
| sample 100 by subclass
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc =
ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf
| rename "predicted(subclass)" as "dga_rf_predicted_subclass"
| apply module3_svm
| rename "predicted(subclass)" as "dga_svm_predicted_subclass"
```

```

| apply module3_dtc
| rename "predicted(subclass)" as "dga_dct_predicted_subclass"
| apply module3_lr
| rename "predicted(subclass)" as "dga_lr_predicted_subclass"
| eval rf_correct=if(dga_rf_predicted_subclass = subclass, 1, 0)
| eval svm_correct=if(dga_svm_predicted_subclass = subclass, 1, 0)
| eval dct_correct=if(dga_dct_predicted_subclass = subclass, 1, 0)
| eval lr_correct=if(dga_lr_predicted_subclass = subclass, 1, 0)
| stats sum(rf_correct) as rf_correct, sum(svm_correct) as svm_correct, sum(dct_correct) as
dct_correct, sum(lr_correct) as lr_correct
| eval rf_correct_pct = round(100 * rf_correct / 400, 0)."%", svm_correct_pct = round(100 *
svm_correct / 400, 0)."%", dct_correct_pct = round(100 * dct_correct / 400, 0)."%",
lr_correct_pct = round(100 * lr_correct / 400, 0)."%
| fields rf_correct_pct, svm_correct_pct, dct_correct_pct, lr_correct_pct

```

Each column in the Statistics table shows the percentage of correct predictions each model made, when predicting which algorithm generated the domain (the subclass feature).

Not bad for a first attempt!

The screenshot shows a Splunk search interface. The top part displays a KQL query in a text area, which is a more complex version of the one shown in the previous code block. Below the query, the search results are shown, including a 'Statistics (1)' table. This table has four columns: rf\_correct\_pct, svm\_correct\_pct, dct\_correct\_pct, and lr\_correct\_pct. The values for these columns are 91%, 89%, 88%, and 89% respectively. The table is highlighted with a blue border.

rf_correct_pct	svm_correct_pct	dct_correct_pct	lr_correct_pct
91%	89%	88%	89%

- These models can be operationalized by using a search like the one below, to predict which algorithms generated a given domain or whether the domain is legitimate. This could even be created as an alert or ES correlation search to help identify devices communicating with domain-generated algorithms.

```

index=dga sourcetype=dga_test earliest=1712102400 latest=1712106000
| `ut_shannon(domain)`
| `ut_meaning(domain)`

```

```

| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf
| rename "predicted(subclass)" as "dga_rf_predicted_subclass"
| search dga_rf_predicted_subclass!=legit

```

*In our testing, this search took about 45s to run.*

*This search is essentially finding any domain that isn't predicted to have a subclass of "legit", meaning that all results returned are predicted to be algorithmically-generated.*

The screenshot shows the Splunk Search & Reporting interface. The 'New Search' panel displays the following search query:

```

index=dga sourcetype=dga_test earliest=1712102400 latest=1712106000
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf
| rename "predicted(subclass)" as "dga_rf_predicted_subclass"
| search dga_rf_predicted_subclass!=legit

```

The search results are displayed in a table with the following columns: Time, Event, host, source, and sourcetype. The results are filtered to show only domains predicted to be algorithmically-generated (DGA).

Time	Event	host	source	sourcetype
4/3/24 12:59:59.000 AM	1712105999,legit,"vs-us.vuemix.com",legit	host = so1	source = /opt/splunk/etc/system/local/dga_test.csv	dga_test
4/3/24 12:59:58.000 AM	1712105998,dga,"mhkigxqricphvg.tw",necurs	host = so1	source = /opt/splunk/etc/system/local/dga_test.csv	dga_test
4/3/24 12:59:58.000 AM	1712105998,dga,"ef0feqklv7g616p.ru",chinad	host = so1	source = /opt/splunk/etc/system/local/dga_test.csv	dga_test

What if you wanted to predict just whether a domain was algorithmically-generated, and not specifically which algorithm generated the domain?



### Task 3: Model Training, Testing, and Application for Class Prediction

Similar to the previous task, you'll be training, testing, and applying four different models in a set of searches. However, this time you will be training models only to predict whether the domain is legitimate or algorithmically-generated (the `class` feature in the testing and training datasets), not which algorithm generated the domain.

You will even be testing these models against algorithms that the models haven't been trained against, just to see how accurate they are. This is useful because in the real world, algorithms generating new domains may appear at any time and you won't have labeled datasets to train from.

1. Run the following search to train models to predict whether a given domain is legitimate (`legit`) or domain generated (`dga`).

```
index=dga sourcetype=dga_test earliest=1712102400 latest=1712106000
| sample 5000 by class
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| fit RandomForestClassifier class from ut_vowel_count, ut_consonant_count,
ut_digit_count, ut_meaning_ratio, ut_shannon, ut_bayesian into module3_rf2
| fit SVM class from ut_vowel_count, ut_consonant_count, ut_digit_count,
ut_meaning_ratio, ut_shannon, ut_bayesian into module3_svm2
| fit DecisionTreeClassifier class from ut_vowel_count, ut_consonant_count,
ut_digit_count, ut_meaning_ratio, ut_shannon, ut_bayesian into module3_dtc2
| fit LogisticRegression class from ut_vowel_count, ut_consonant_count,
ut_digit_count, ut_meaning_ratio, ut_shannon, ut_bayesian into module3_lr2
```

*In our testing, this search took about 65s to run.*

*Note how in each of the `fit` commands, the search lists the `class` field as the one to predict.*

2. In the browser tab open to the **DGA App for Splunk**, click **Dashboards** and select the **5. Test and Benchmark** dashboard. In the app, this is the dashboard that is used to test the models against many algorithms including many not in the testing dataset.
3. Run the following search to measure how well the four models do against all of the algorithms in the testing dataset

```

index=dga sourcetype=dga_test earliest=1712188800 latest=1712189700
| sample 1000 by class
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2
| rename "predicted(class)" as "dga_rf_predicted_class"
| apply module3_svm2
| rename "predicted(class)" as "dga_svm_predicted_class"
| apply module3_dtc2
| rename "predicted(class)" as "dga_dct_predicted_class"
| apply module3_lr2
| rename "predicted(class)" as "dga_lr_predicted_class"
| eval rf_correct=if(dga_rf_predicted_class = class, 1, 0)
| eval svm_correct=if(dga_svm_predicted_class = class, 1, 0)
| eval dct_correct=if(dga_dct_predicted_class = class, 1, 0)
| eval lr_correct=if(dga_lr_predicted_class = class, 1, 0)
| stats sum(rf_correct) as rf_correct, sum(svm_correct) as svm_correct,
sum(dct_correct) as dct_correct, sum(lr_correct) as lr_correct
| eval rf_correct_pct = round(100 * rf_correct / 2000, 0)."%", svm_correct_pct =
round(100 * svm_correct / 2000, 0)."%", dct_correct_pct = round(100 * dct_correct /
2000, 0)."%", lr_correct_pct = round(100 * lr_correct / 2000, 0).%"
| fields rf_correct_pct, svm_correct_pct, dct_correct_pct, lr_correct_pct

```

*Notice how the correctly guessed percentages have dropped from about 90% to about 80%. This is likely because there are domains in this training dataset the algorithms have never seen before.*

New Search

```

index=dga sourcetype=dga_test earliest=1712188800 latest=1712189700
| sample 1000 by class
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2
| rename "predicted(class)" as "dga_rf_predicted_class"
| apply module3_svm2
| rename "predicted(class)" as "dga_svm_predicted_class"
| apply module3_dtc2
| rename "predicted(class)" as "dga_dct_predicted_class"
| apply module3_lr2
| rename "predicted(class)" as "dga_lr_predicted_class"
| eval rf_correct=if(dga_rf_predicted_class = class, 1, 0)
| eval svm_correct=if(dga_svm_predicted_class = class, 1, 0)
| eval dct_correct=if(dga_dct_predicted_class = class, 1, 0)
| eval lr_correct=if(dga_lr_predicted_class = class, 1, 0)
| stats sum(rf_correct) as rf_correct, sum(svm_correct) as svm_correct, sum(dct_correct) as dct_correct, sum(lr_correct) as lr_correct
| eval rf_correct_pct = round(100 * rf_correct / 2000, 0) "%", svm_correct_pct = round(100 * svm_correct / 2000, 0) "%", dct_correct_pct = round(100 * dct_correct / 2000, 0) "%", lr_correct_pct = round(100 * lr_correct / 2000, 0) "%"
| fields rf_correct_pct, svm_correct_pct, dct_correct_pct, lr_correct_pct

```

2,000 events (4/4/24 12:00:00.000 AM to 4/4/24 12:15:00.000 AM) No Event Sampling

Events Patterns Statistics (1) Visualization

20 Per Page Format Preview

rf_correct_pct %	svm_correct_pct %	dct_correct_pct %	lr_correct_pct %
82%	72%	78%	76%

#### 4. Run the search below to create a confusion matrix for just the random forest model

```

index=dga sourcetype=dga_test earliest=1712188800 latest=1712189700
| sample 1000 by class
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2
| rename "predicted(class)" as "dga_rf_predicted_class"
| apply module3_svm2
| rename "predicted(class)" as "dga_svm_predicted_class"
| apply module3_dtc2

```

```

| rename "predicted(class)" as "dga_dct_predicted_class"
| apply module3_lr2
| rename "predicted(class)" as "dga_lr_predicted_class"
| eval tp = if(class="dga" AND dga_rf_predicted_class="dga", 1, 0)
| eval fp = if(class="legit" AND dga_rf_predicted_class="dga", 1, 0)
| eval tn = if(class="legit" AND dga_rf_predicted_class="legit", 1, 0)
| eval fn = if(class="dga" AND dga_rf_predicted_class="legit", 1, 0)
| stats sum(tp) as true_positives, sum(fp) as false_positives, sum(tn) as
true_negatives, sum(fn) as false_negatives

```

The screenshot shows the Splunk Search interface. The search bar contains the following query:

```

index=dga sourcetype=dga_test earliest=1712188800 latest=1712189700
| sample 1000 by class
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2_01
| rename "predicted(class)" as "dga_rf_predicted_class"
| eval tp = if(class="dga" AND dga_rf_predicted_class="dga", 1, 0)
| eval fp = if(class="legit" AND dga_rf_predicted_class="dga", 1, 0)
| eval tn = if(class="legit" AND dga_rf_predicted_class="legit", 1, 0)
| eval fn = if(class="dga" AND dga_rf_predicted_class="legit", 1, 0)
| stats sum(tp) as true_positives, sum(fp) as false_positives, sum(tn) as true_negatives, sum(fn) as false_negatives

```

The results table shows the following statistics:

true_positives	false_positives	true_negatives	false_negatives
733	97	903	267

- Run the search below to see the number of correct and incorrect predictions made by the random forest model stratified by which algorithm generated the domain (`subclass` feature) in the testing dataset. Use the **Bar Chart** visualization, and **Format** the visualization so that the bars are **stacked**. This is effectively re-creating the left panel in the second row from the top on the 5. Test and Benchmark dashboard in the DGA App for Splunk.

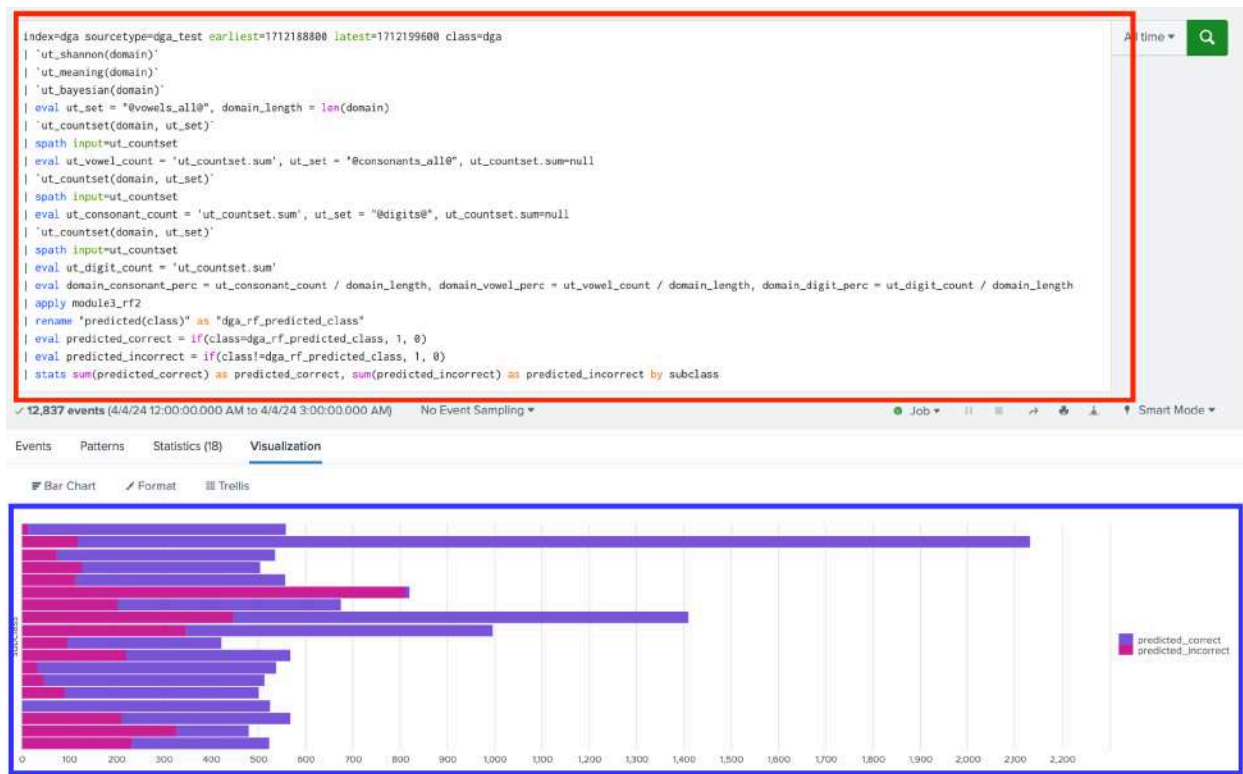
```

index=dga sourcetype=dga_test earliest=1712188800 latest=1712189700 class=dga
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null

```

```
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2
| rename "predicted(class)" as "dga_rf_predicted_class"
| eval predicted_correct = if(class=dga_rf_predicted_class, 1, 0)
| eval predicted_incorrect = if(class!=dga_rf_predicted_class, 1, 0)
| stats sum(predicted_correct) as predicted_correct, sum(predicted_incorrect) as
predicted_incorrect by subclass
```

*Note how the model was not very successful in identifying domains generated by some algorithms it didn't train against, like the gozi algorithm, but it was reasonably successful in predicting domains generated by other algorithms it has not been trained against like proslikefan.*



- This model to only identify whether a domain is algorithmically-generated can be operationalized in a search like the one below. Just like with the previous task this search could be created as an alert or ES correlation search to help identify devices communicating with domain-generated algorithms.

```
index=dga sourcetype=dga_test earliest=1712188800 latest=1712199600 class=dga
| `ut_shannon(domain)`
| `ut_meaning(domain)`
| `ut_bayesian(domain)`
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
```

```
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@",
ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc
= ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2
| rename "predicted(class)" as "dga_rf_predicted_class"
| search dga_rf_predicted_class=dga
```

*This search is looking for any domains where the predicted class is dga, meaning any domain generated by an algorithm.*

The screenshot shows the Splunk Enterprise web interface. At the top, there's a navigation bar with 'Search' selected. Below it, a 'New Search' panel is visible. The search query is pasted into the search bar and is highlighted with a red box. The query is as follows:

```
index=dga sourcetype=dga_test earliest=1712188800 latest=1712199600 class=dga
| 'ut_shannon(domain)'
| 'ut_meaning(domain)'
| 'ut_bayesian(domain)'
| eval ut_set = "@vowels_all@", domain_length = len(domain)
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_vowel_count = 'ut_countset.sum', ut_set = "@consonants_all@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_consonant_count = 'ut_countset.sum', ut_set = "@digits@", ut_countset.sum=null
| `ut_countset(domain, ut_set)`
| spath input=ut_countset
| eval ut_digit_count = 'ut_countset.sum'
| eval domain_consonant_perc = ut_consonant_count / domain_length, domain_vowel_perc = ut_vowel_count / domain_length, domain_digit_perc = ut_digit_count / domain_length
| apply module3_rf2
| rename "predicted(class)" as "dga_rf_predicted_class"
| search dga_rf_predicted_class=dga
```

Below the search bar, the results are displayed. The top section shows a bar chart representing the event distribution over time. Below the chart, there's a table of results. The table has columns for 'Time' and 'Event'. Two rows are visible, both dated 4/4/24 at 2:59:58.000 AM. The first row shows a domain 'ouiugn.dynserv.com' and the second row shows 'vvvoeqpf.dynserv.com'. Both events are categorized as 'dga\_test'.

i	Time	Event
>	4/4/24 2:59:58.000 AM	1712199598, dga, "ouiugn.dynserv.com", "kraken_v1" host = sof source = /opt/splunk/etc/system/local/data/dga_test.csv sourcetype = dga_test
>	4/4/24 2:59:58.000 AM	1712199598, dga, "vvvoeqpf.dynserv.com", "kraken_v1" host = sof source = /opt/splunk/etc/system/local/data/dga_test.csv sourcetype = dga_test

In this module you trained multiple models to predict not only which algorithm generated a domain, but also to predict whether a domain was generated by an algorithm. You also benchmarked and test against domains generated by algorithms the model had never seen before.

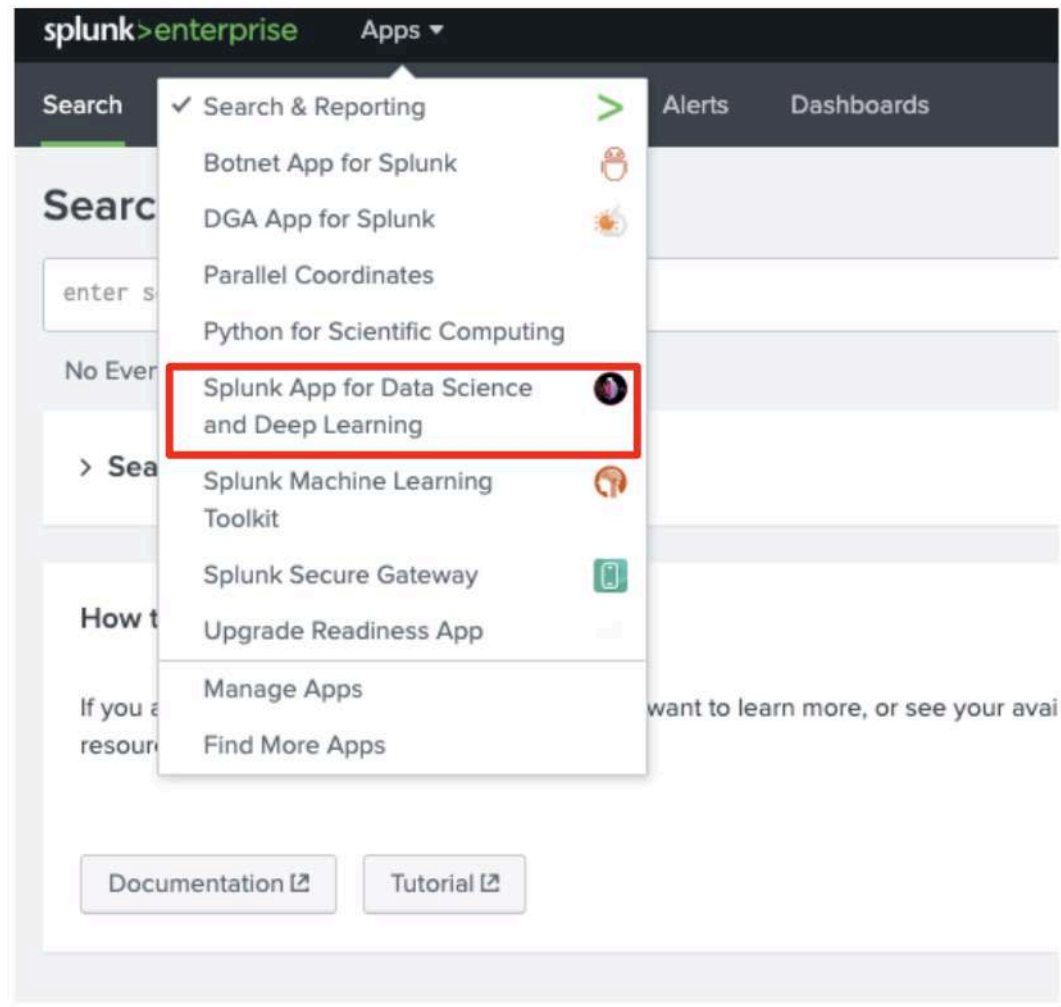


## Module 4: Data Science & Deep Learning

In this module you will further explore Splunk's custom AI capabilities using the Data Science and Deep Learning (DSDL) app which enables deep learning modeling through a containerized deep learning environment and integration with Jupyter notebooks.

### Task 1: Verify DSDL Setup

1. Navigate to the DSDL application



*NOTE: If you receive the message window below at anytime, please click the “Run Query Anyway” button*

## We've identified a potential security risk



The search that you are about to run contains commands that might present a security risk. [Learn more](#)

The flagged commands are:

- fit

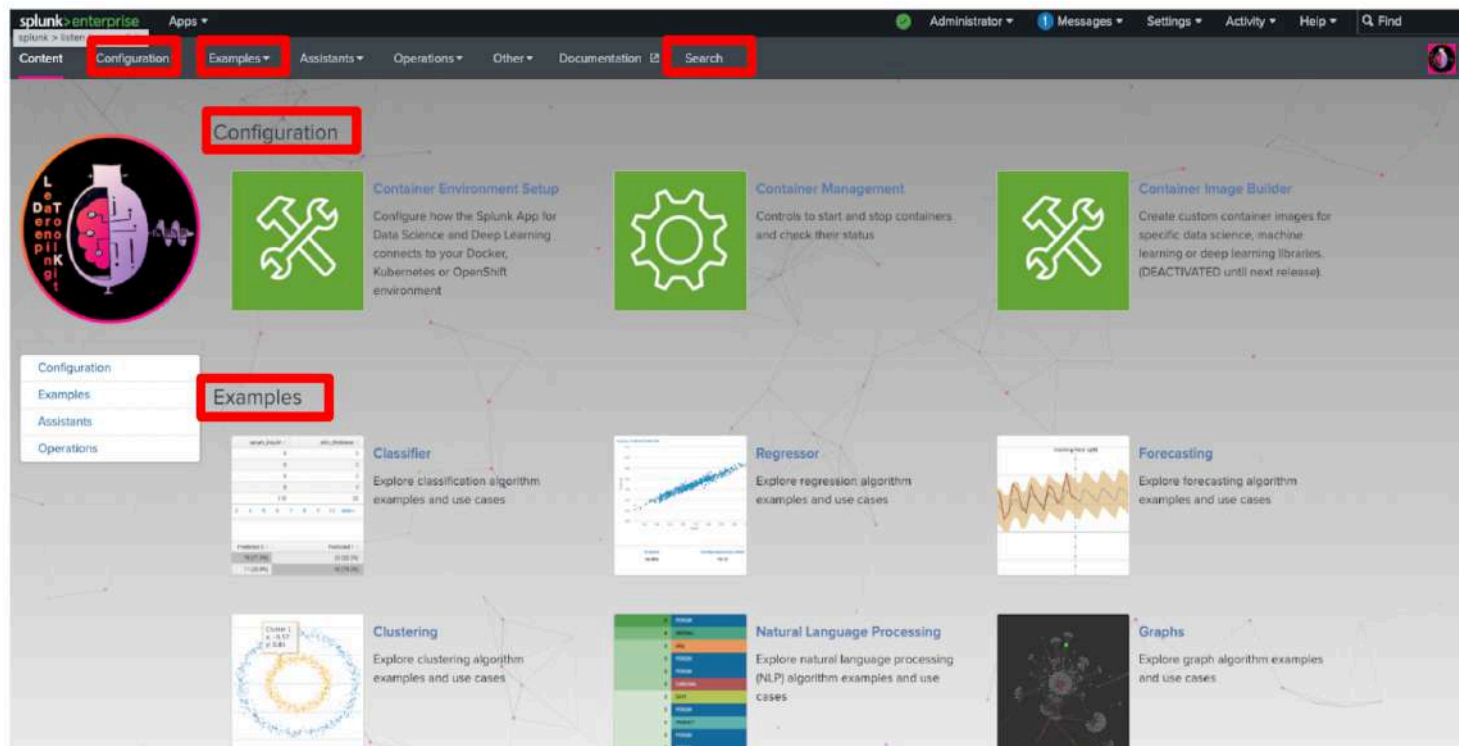
Do you want to investigate the search string?

Cancel

Run Query Anyway

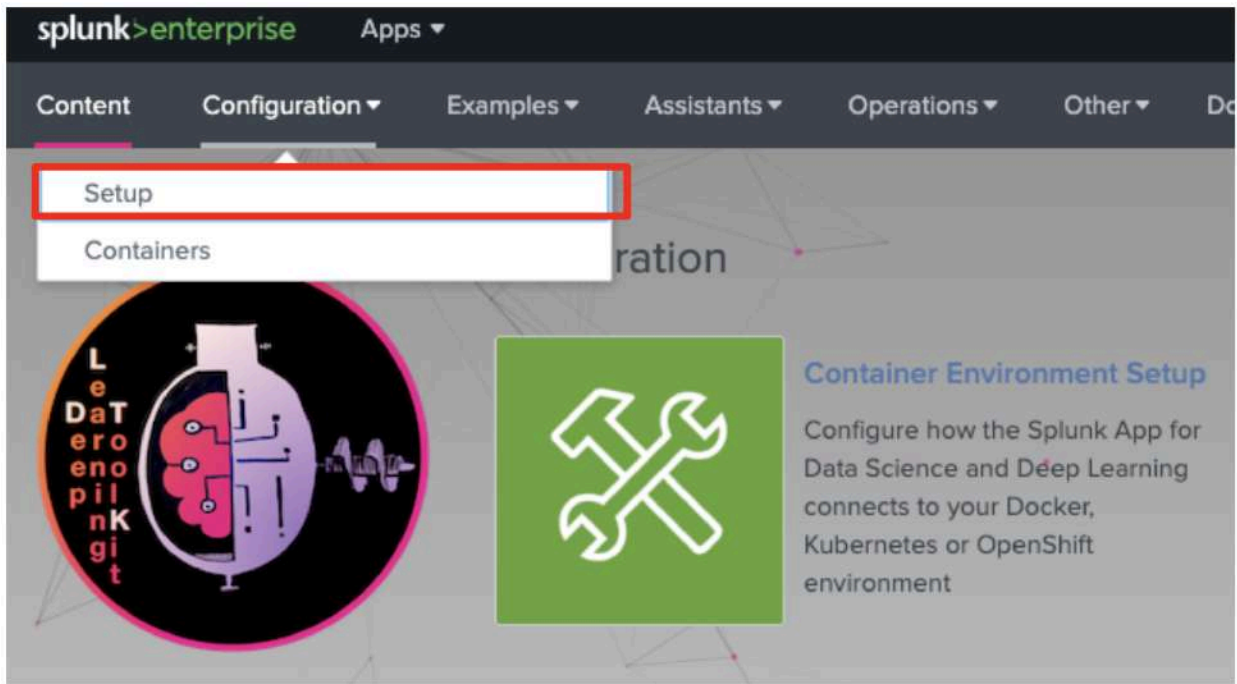
Investigate

2. You should now be at the DSDL home page. Take a moment to familiarize yourself with the tool bar options, especially (**Configuration**, **Examples**, and **Search**). Don't forget to scroll down, to see the full menu of options.

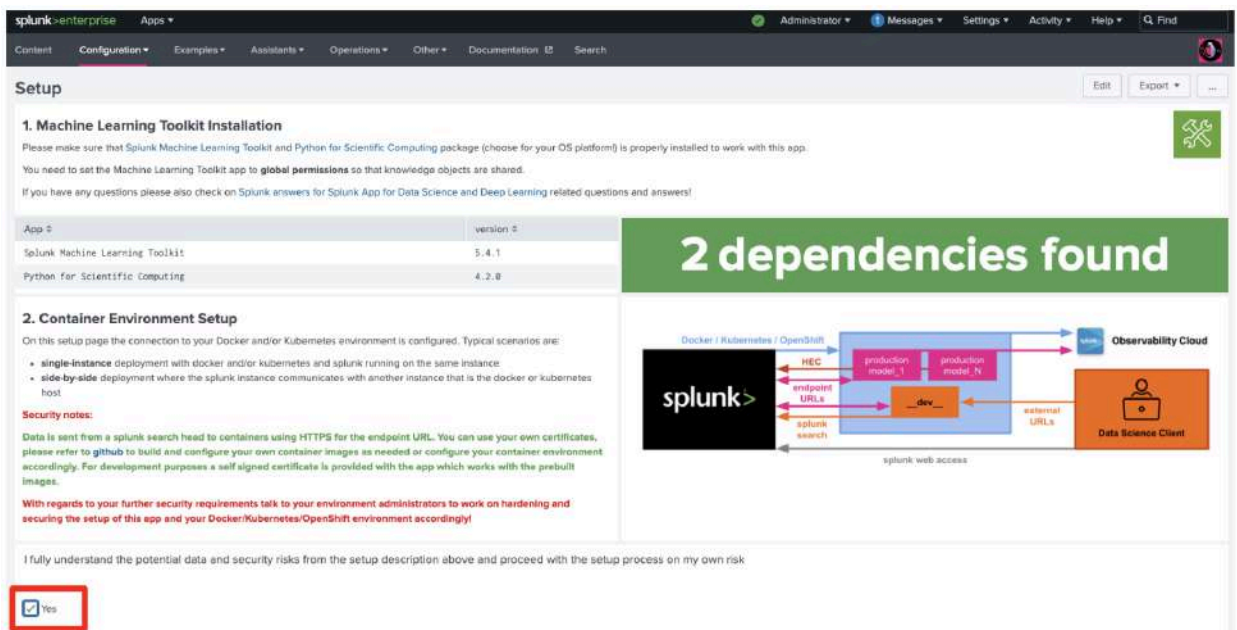


3.

4. Let's look at the container settings: **Configuration > Setup**



5. Now, take a closer look at the container setup. Click the “yes” box scroll down



6. Verify the Docker container setup (*make sure to scroll down*) to ensure that the *Docker* container information, *Password Settings*, and *Splunk Access Settings* are populated similar to the images below, and then click **Test & Save**.

*The DSDL enables you to utilize a containerized compute (CPU or GPU) environment for heavy modeling processing. In a production environment best practice would be to utilize Docker in a Kubernetes environment such as AWS EKS, but in this lab you will be using a Docker container running on your local instance.*

### Docker

Docker Settings

Docker Host:

Endpoint URL:

External URL:

Spunk Docker Logging Endpoint (optional):

Spunk Docker Logging (HEC token) (optional):

Deployment ID	docker host ID	endpoint url ID	internal url ID	host os ID
single-instance	unix:///var/run/docker.sock	localhost	localhost	linux
single-instance	tcp://localhost:2375	localhost	localhost	windows
single-instance	tcp://host.docker.internal:2375	host.docker.internal	host.docker.internal	docker
side-by-side	tcp://monitor-host.com:2375	monitor-host.com	internal-host.com	any

### Kubernetes

Kubernetes Settings

Authentication Mode: ☐ Cert B... ☐ User T... ☐ User L... ☐ AWS... ☐ Service Ac...

Cluster Base URL:

Cluster Certificate Authority:

Client Certificate:

Client Key:

Service Type: ☐ Load Balancer ☐ Node Port ☐ Ingress

Namespace:

Storage Class:

Image Pull Secrets:

In Cluster Mode: ☐ No ☐ Yes

API Tails, i.e. cluster internal, e.g. Splunk running in the same Kubernetes cluster

### Certificate Settings

Endpoint Certificate Settings

The Splunk App for Data Science and Deep Learning (DSDL) connects from the search head to the container endpoints over HTTPS. By default, a self-signed certificate is provided in the prebuilt DSDL containers for development purposes. For production setup, we recommend to enable **hostname** checking initiated by default. For development purposes with self-signed certificates you can disable hostname checking. For more information, please find the source code, build scripts and more details on GitHub.

Check Hostname: ☐ Disabled ☐ Enabled

By default DSDL tries to retrieve the SSL certificate from its container endpoint. In case you want to point the DSDL app to your own certificate or CA chain on your splunk instance then you can enter a path or filename here to use this instead:

Certificate filename or path (optional):

By default DSDL containers use a certificate for all endpoints for HTTPS communication. If you work with an ingress or loadbalancer setup in your container environment like Kubernetes you will likely terminate HTTPS at this point. For those cases you can optionally configure the containers to not use the self-signed certificate, but your own. Note that you must ensure your ingress takes care of all HTTPS and certificate handling. Regardless, all data transfer related communication is forced to HTTPS, so there is no option for unencrypted HTTP traffic.

Enable container certificates (optional setting: disable for HTTPS configured at ingress level, e.g. in Kubernetes)

☐ No ☐ Yes

### Password Settings

Password Settings

For enhanced security you can define a custom token for the container API endpoint and a password for Jupyter Lab.

API Endpoint Token Settings

All container endpoints are protected with a token that can be defined here. Please note that changes will take effect only for newly started containers. If you leave the text box empty a default random token will be used.

Endpoint Token:

Jupyter Lab Password Settings

You can change the default password for Jupyter Lab here:

Jupyter Password:

**Security notes: It is strongly recommended to set a custom password here. Alternatively you can manage this in your container environment by overriding the JUPYTER\_PASSWORD environment variable. Please note that changes will take effect only for newly started containers. If you leave the text box empty the default password will be used.**

### Observability Settings

Generic Instrumentation (optional)

Splunk Observability Cloud provides you with full-fidelity monitoring and troubleshooting across infrastructure, application and users, in real-time and at any scale.

The API of all DSDL containers can be automatically instrumented and traces can be collected to get more insights with Splunk Observability. Please enter your access token, set endpoint and service name and enable the functionality. If you don't have an account with Splunk Observability yet, you can start a trial here.

Enable Observability: ☐ No ☐ Yes

Splunk Observability Access Token:

Open Telemetry Endpoint:

Open Telemetry ServiceName:

### Splunk Access Settings

Splunk Access in Jupyter (optional)

This setting allows you to configure access to your splunk deployment from the container environment. It is useful to pull data with Splunk's Python SDK into your Jupyter Lab Notebooks. To use the new interactive splunk search bar in the barebone\_template notebook in Jupyter you need to setup this. Please generate an access token in your Splunk settings > Tokens. Copy and paste the generated token below and add your splunk host and management port (default: 8089).

Enable Splunk Access: ☐ No ☐ Yes

Splunk Access Token:

Splunk Host Address:

Splunk Management Port:

**Security notes: It is strongly recommended to restrict the Splunk access token to only necessary capabilities to scope permissions down, e.g. to only allow access to selected indexes. Please note that changes will take effect only for newly started containers.**

### Splunk HEC Settings

Splunk HEC in Jupyter (optional)

This setting allows you to actively send back data to your splunk deployment from the container environment using Splunk's HTTP Event Collector (HEC). Please create a HEC data input, paste the token below and set your endpoint URL. It is useful to push generated data or model results back to Splunk as JSON events that get indexed. You can utilize the prebuilt SplunkHEC class available in the barebone\_template notebook example. Please note: when you change the settings below you need to restart your container(s) so that the configuration changes take effect in the container.

Enable Splunk HEC: ☐ No ☐ Yes

Splunk HEC Token:

Splunk HEC Endpoint URL:

### 3. Test the connection and save the configuration

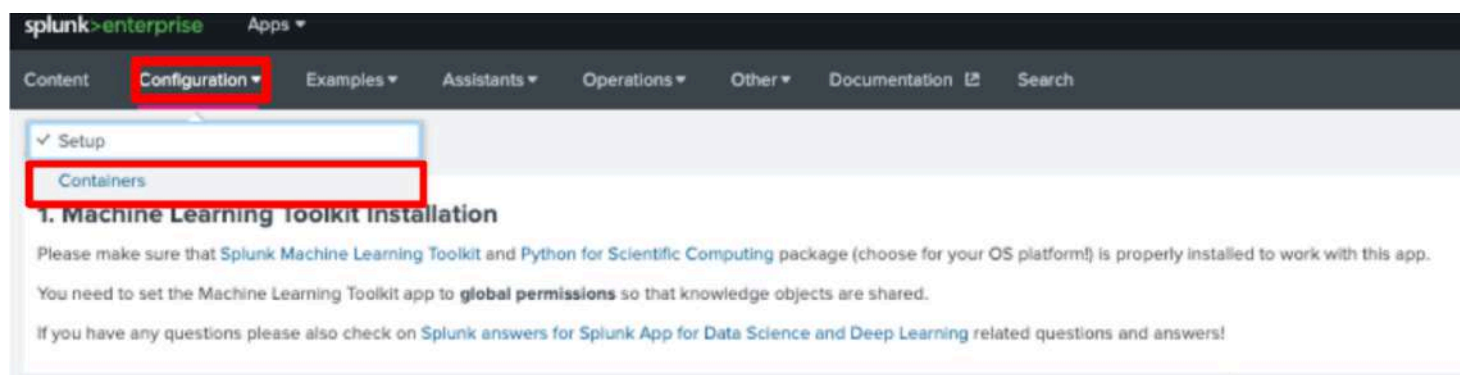
7. Wait for the *setup completed* window to appear, and then click **OK**

## Setup Completed

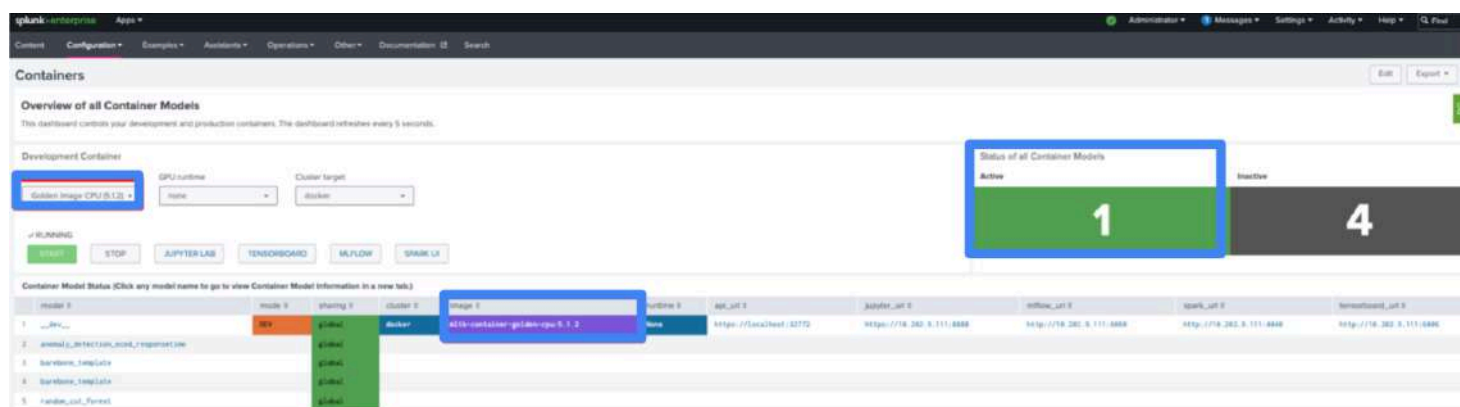
Successfully established connection to container environment.

OK

8. Confirm container is running. Click on **Configuration > Containers**



9. Verify you have the right container running “Golden Image CPU (5.1.2)”. You should see one container showing **Active**.

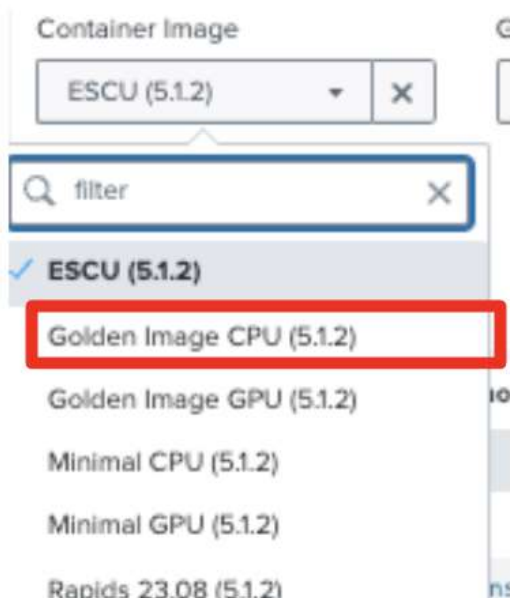


If you do not see the correct image Golden Image CPU (5.1.2), perform steps 9.a - 9.c. Else, skip to 8

- a. Click stop



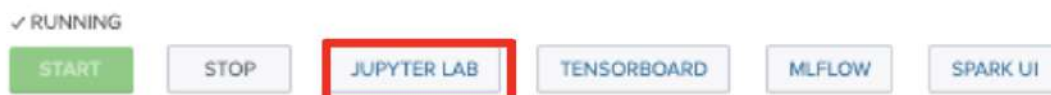
- b. Select the Golden Image CPU (5.1.2)



c. Click start



10. Click on **Jupyter Lab**



a. If you receive this window, click on **Advanced**



### Your connection is not private

Attackers might be trying to steal your information from **10.202.34.118** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR\_CERT\_AUTHORITY\_INVALID

To get Chrome's highest level of security, [turn on enhanced protection](#)



Back to safety

b. Then click the **“Proceed to...”** link





## Your connection is not private

Attackers might be trying to steal your information from **10.202.34.118** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR\_CERT\_AUTHORITY\_INVALID



To get Chrome's highest level of security, [turn on enhanced protection](#)

Hide advanced

Back to safety

This server could not prove that it is **10.202.34.118**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to 10.202.34.118 \(unsafe\)](#)

11. A new tab will open. Enter your Password: Splunk4DeepLearning



Password:

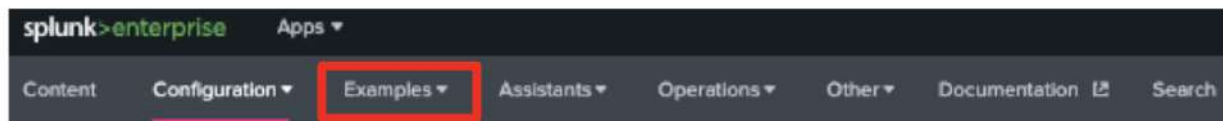
Log in

## Task 2: Review the Example results in the DSDL interface

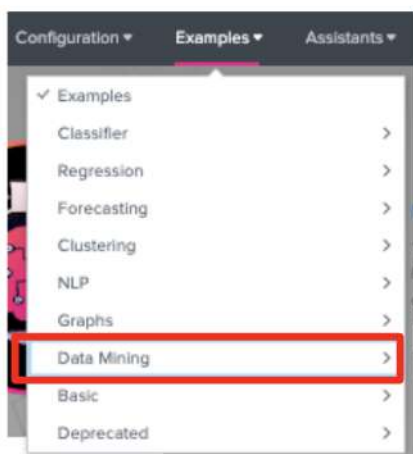
In this task you will review an anomaly detection modeling example in the DSDL. Take a moment to observe the visualization of both the outlier data and the predicted outliers

### Steps

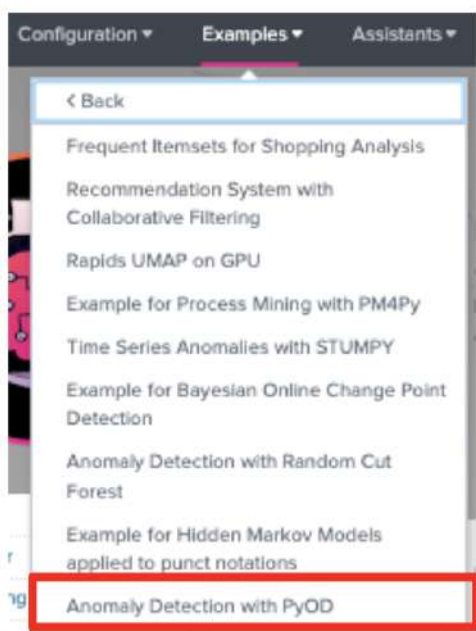
1. Go back to your original Tab and click on “Examples”



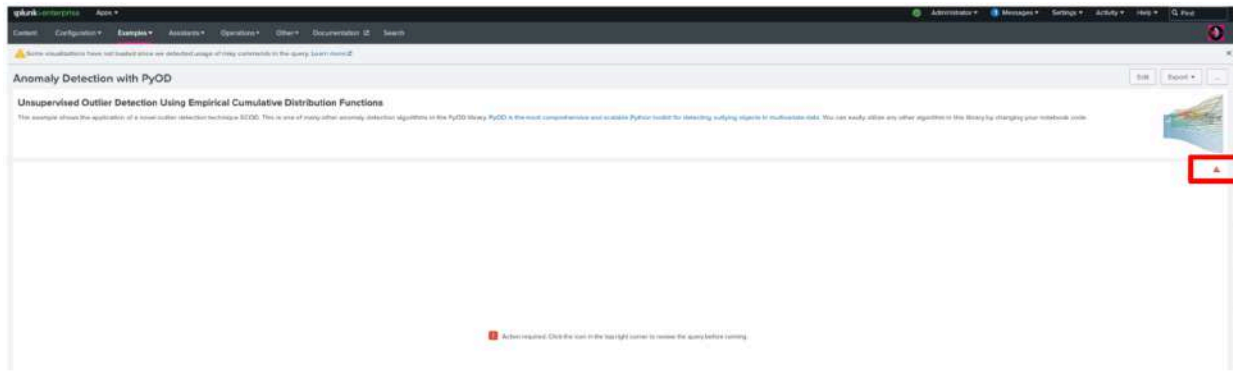
2. Select **Data Mining**



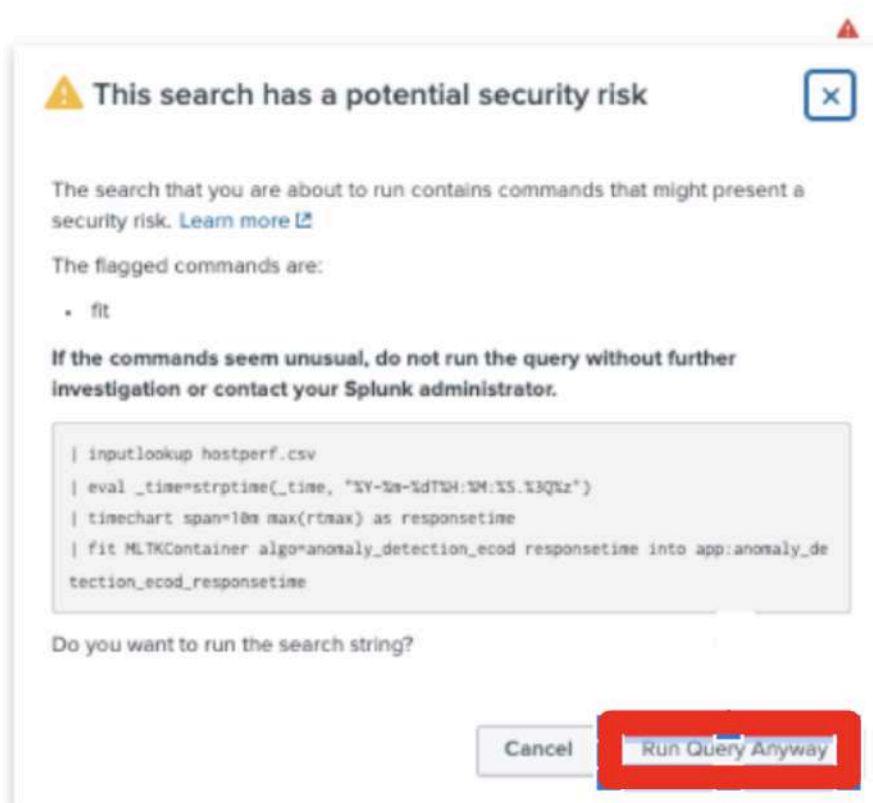
3. Then select **Anomaly Detection with PyOD**



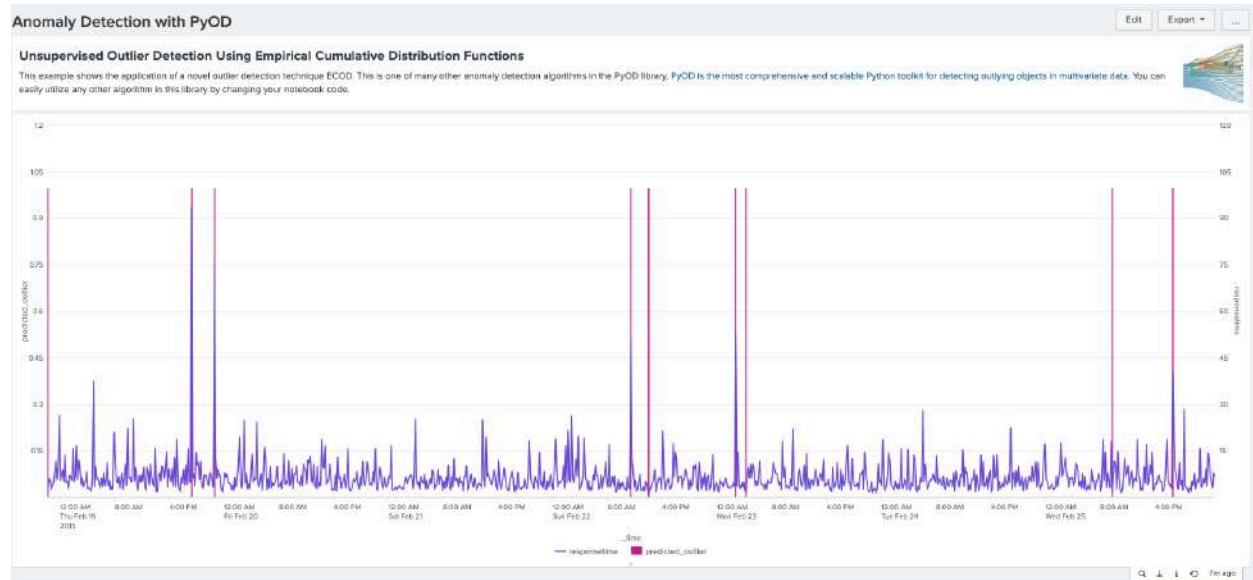
4. Click the warning triangle



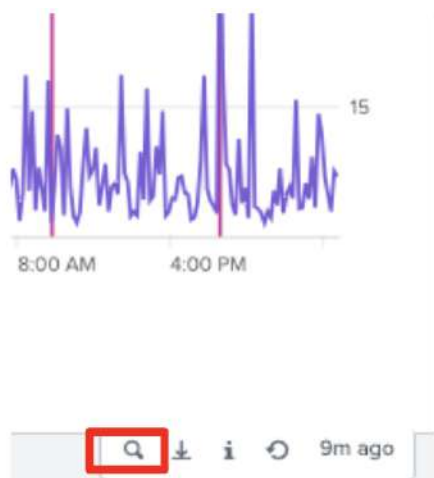
5. Click **Run Query Anyway**



6. We just used the “PyOD” library to perform an unsupervised outlier detection. Notice the blue line is mapping the response time outliers within the dataset and the red lines represent predicted outliers.



- Let's take a closer look at our SPL that generated this output. Click the magnifying glass (*Make sure to hover over the panel for the magnifying glass to appear*)



- A new search tab will open up and you can see the SPL and the results that were graphed in addition to the prediction scores. Notice the SPL contains a **fit** command. This command includes a setting to use your MLTKContainer. You will also notice the algorithm name is *anomaly\_detection\_ecod*



Click on the sort areas in the predicted\_outlier column to see how many outliers were identified.

Events Patterns Statistics (1,024) Visualization			
20 Per Page ▾	Format ▾	Preview ▾	< Prev 1 2 3 4 5 6 7 8 ... Next >
_time ▴	responseTime ▴		predicted_outlier ▴
2015-02-18 22:10:00	1.275		1
2015-02-18 22:20:00	5.933		0
2015-02-18 22:30:00	5.599		0
2015-02-18 22:40:00	2.839		0
2015-02-18 22:50:00	3.782		0
2015-02-18 23:00:00	4.682		0
2015-02-18 23:10:00	8.361		0
2015-02-18 23:20:00	11.885		0
2015-02-18 23:30:00	5.519		0
2015-02-18 23:40:00	10.844		0
2015-02-18 23:50:00	26.55		0

- Next click on the **Job** down arrow. You will see the job has been sent to the container for processing. Remember, one purpose of the DSDL is to offload the modeling process to the containerized compute environment.

Job ▾

MLTKC endpoint: <https://localhost:32773>

MLTKC info: /fit done successfully with parameters {'params': {'algo': 'anomaly\_detection\_ecod'}, 'args': ['responsetime'], 'feature\_variables': ['responsetime'], 'model\_name': 'anomaly\_detection\_ecod\_responsetime', 'algo\_name': 'MLTKContainer', 'mispl\_limits': {'handle\_new\_cat': 'default', 'max\_distinct\_cat\_values': '100', 'max\_distinct\_cat\_values\_for\_classifiers': '100', 'max\_distinct\_cat\_values\_for\_scoring': '100', 'max\_fit\_time': '600', 'max\_inputs': '100000'}}

Edit Job Settings...

Send Job to Background

Inspect Job

Delete Job

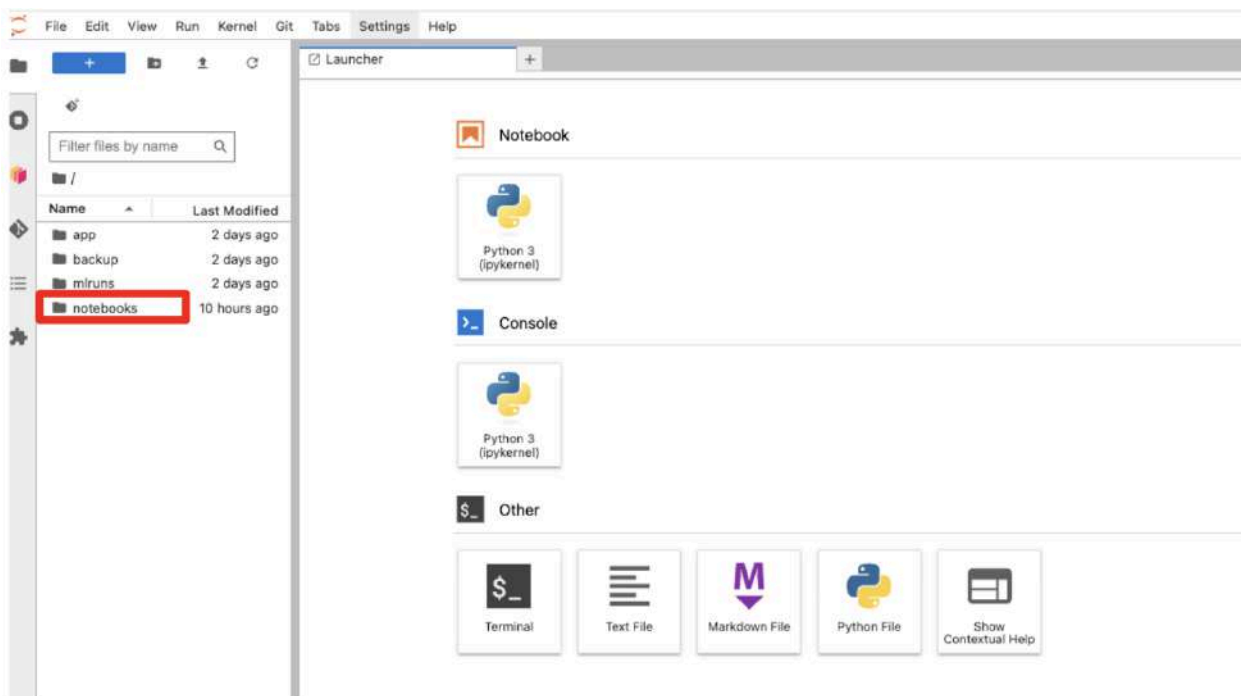
Now that you've seen an example of what is possible, it is now time to see how the example was created. Over the remaining tasks you will recreate what you just viewed in the DSDL example.

### Task 3: Access the Jupyter Lab

In this scenario you are using Splunk's integration with Jupyter Notebooks to create the custom model you are using in your SPL. The Jupyter Lab allows you to perform the data analysis, feature engineering, and model initialization and training with the Jupyter Notebook. Let's take a look at the example Jupyter Notebook.

#### Steps

1. Go to the Jupyter Environment tab you created earlier and double click on **notebooks**



2. Do a search for *anomaly* and you will find the *anomaly\_detection\_ecod* notebook. Double click it.



3. In the following steps, you are going to run each cell of python code in the notebook. When you want to run a cell, click on the cell you want to run and click the run icon





Or press **shift + return** on your keyboard

4. Run each of the following cells in *1.2. Stage 0 - import libraries*:

a. Import your libraries

```
[2]: # this definition exposes all python module imports that should be available in all subsequent commands
import json
import numpy as np
import pandas as pd
from pyod.models.ecod import ECOD
# from pyod.models.iforest import IForest
# ...
# global constants
MODEL_DIRECTORY = "/srv/app/model/data/"
```

b. Verify your library versions

```
[3]: # THIS CELL IS NOT EXPORTED - free notebook cell for testing or development purposes
print("numpy version: " + np.__version__)
print("pandas version: " + pd.__version__)

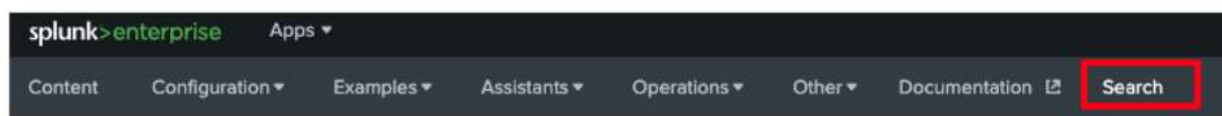
numpy version: 1.26.4
pandas version: 2.2.2
```

c. Verify the algorithm you will use is available and has metadata. Feel free to scroll through the output to familiarize yourself with the data structure used by the model.

```
[4]: dir(ECOD) # Model Metadata

['_str_',
 '_subclasshook_',
 '_weakref_',
 '_abc_impl',
 '_decision_function_parallel',
 '_get_param_names',
 '_predict_rank',
 '_process_decision_scores',
 '_set_n_classes',
 'decision_function',
 'explain_outlier',
 'fit',
 'fit_predict',
 'fit_predict_score',
 'get_params',
 'predict',
 'predict_confidence',
 'predict_proba']
```

5. Before you move forward to the next step, you should familiarize yourself with the dataset we will be working with. Click on your Splunk UI browser tab and open search



6. Next, run the following search to view the entire dataset you will be working with.

| [inputlookup](#) hostperf.csv

Take note of the number of results in the dataset and the abbreviated response time values. You will use Splunk to create a sample from the dataset and create a new field (*responsetime*), which will be the feature used to train your model.

New Search Save As Create Table View Close

`inputlookup hostperf.csv` All time Q

✓ 9,999 results 170 12:00:00.000 AM to 5/23/24 2:06:34.000 PM No Event Sampling Job Smart Mode

Events Patterns **Statistics (9,999)** Visualization

20 Per Page Format Preview 1 2 3 4 5 6 7 8 Next

_time	hoststate	pl	rt	rtmax	rtmin	src_host	timestamp
2015-02-18 14:17:26	UP	0%	1.856	1.191	0.989	Host48	142423786
2015-02-18 14:18:27	UP	0%	1.128	1.275	0.965	Host48	142423796
2015-02-18 14:19:28	UP	0%	1.888	1.254	1.885	Host48	142423799
2015-02-18 14:20:34	UP	0%	1.888	1.499	0.932	Host48	142423800
2015-02-18 14:21:35	UP	0%	1.772	2.182	1.115	Host48	142423808
2015-02-18 14:22:39	UP	0%	1.642	3.816	0.862	Host48	142423811

## Task 4: Test Sample Dataset Creation

Now that you are familiar with the dataset, it is time to create your sample dataset you will use for modeling. You will establish a connection to Splunk and use a search to pull a dataset directly from Splunk for sample dataset creation. You will first adjust the search to generate the sample size you want and then you will add a **fit** command to generate the final sample dataset to be used for modeling.

### Steps

1. Click on the Jupyter notebook browser tab and scroll to 1.3. *Stage 1 - get a data sample from Splunk* and then go to 1.3.1.1. *Option 1.a - Create test sample dataset*. For the next steps you will be using the cells in this section.
2. Run the first cell to import the Splunk library which is needed to communicate with your splunk environment.

```
[6]: #Interact with Splunk to validate your search results. Change span to 30m
from dsdl.support import SplunkSearch as SplunkSearch
```

3. Next, run the next cell to establish the search connection with Splunk

```
search = SplunkSearch.SplunkSearch(search='| inputlookup hostperf.csv | eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z") | timechart span=10m max(rt)
```

4. Once the connection is established, run the next cell to initialize a Splunk search window. The window will appear in the cell as shown below.

```
[2]: search = SplunkSearch.SplunkSearch(search='| inputlookup hostperf.csv \n| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z") \n| timechart span=10m max(rta
```

search	inputlookup hostperf.csv   eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")   timechart span=10m max(rta) as responsetime	Q search
		earliest -15m@m
		latest now

search

results

result info

- Click the search button and wait for the status bars and results info to show completed.  
**IMPORTANT:** Do not click “run” after this has completed.

```
search = SplunkSearch.SplunkSearch(search='| inputlookup hostperf.csv \n| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z") \n| timechart span=10m max(r
```

search	inputlookup hostperf.csv   eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")   timechart span=10m max(rta) as responsetime	Q search
		earliest -15m@m
		latest now

search done

loading done

result info

- Next, you will ensure the dataframe has been populated with the search results.

Click on the next cell and then click **run**.

*Note: The previous cell must still show completed status bars in order for the dataframe to be populated with results.*

```
[15]: df = search.as_df()
df
#make sure to run this in proper sequence.
```

	_time	responsetime	_span
0	2015-02-18T22:10:00.000+00:00	1.275	600
1	2015-02-18T22:20:00.000+00:00	5.933	600
2	2015-02-18T22:30:00.000+00:00	5.599	600
3	2015-02-18T22:40:00.000+00:00	2.839	600
4	2015-02-18T22:50:00.000+00:00	3.702	600
...	...	...	...
1020	2015-02-26T00:10:00.000+00:00	4.894	600
1021	2015-02-26T00:20:00.000+00:00	2.898	600
1022	2015-02-26T00:30:00.000+00:00	7.564	600
1023	2015-02-26T00:40:00.000+00:00	6.938	600
1024	2024-05-17T18:40:00.000+00:00	NaN	600

1025 rows x 3 columns

\*If the dataframe does not return results, rerun the search and ensure the results are showing successful as pictured above in **Task 4: step 5**. Then rerun the dataframe cell.

Notice the number of results we have generated for our sample dataset. You will also notice you have enriched your data with a new field (*responsetime*) that will be used for modeling results.

7. You can change the span parameter as needed to adjust the size of the sample dataset. *Sample size can impact the accuracy of your modeling results. It is best to try multiple sample sizes to optimize your modeling results.*

Before you do, make sure to rerun the “SplunkSearch” import step first.

```
#interact with Splunk to validate your search results. Change span to 30m
from dsdlsupport import SplunkSearch as SplunkSearch
```

This ensures your dataframe is populated with your new search results.

8. Now, try a few span iterations and observe how they impact your sample size. The example below shows a change from **span=10m** to **span=20m** *For this step, please set span to 2m or greater when testing.*

h.SplunkSearch(search='| inputlookup hostperf.csv | eval \_time=strptime(\_time, "%Y-%m-%dT%H:%M:%S.%3Q%z") | timechart span=20m max(rtmax) as responsetime'

search | inputlookup hostperf.csv  
| eval \_time=strptime(\_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")  
| timechart span=20m max(rtmax) as responsetime

search done

loading done

result info search completed with 514 results.["Successfully read lookup file '/opt/splunk/etc/apps/Splunk\_ML\_Toolkit/lookups/hostperf.csv'."][info: ["Successfully read lookup file '/opt/splunk/etc/apps/Splunk\_M

```
df = search.as_df()
```

```
df
```

```
#make sure to run this in proper sequence.
```

	_time	responsetime	_span
0	2015-02-18T22:00:00.000+00:00	1.275	1200
1	2015-02-18T22:20:00.000+00:00	5.933	1200
2	2015-02-18T22:40:00.000+00:00	3.702	1200
3	2015-02-18T23:00:00.000+00:00	8.361	1200
4	2015-02-18T23:20:00.000+00:00	11.885	1200
...	...	...	...
509	2015-02-25T23:40:00.000+00:00	14.206	1200
510	2015-02-26T00:00:00.000+00:00	6.892	1200
511	2015-02-26T00:20:00.000+00:00	7.564	1200
512	2015-02-26T00:40:00.000+00:00	6.938	1200
513	2024-05-20T13:00:00.000+00:00	NaN	1200

514 rows x 3 columns

Notice the change in the sample size. You can repeat these steps until you have the sample size you want. When you have decided which span setting you will use, move on to Task 5. The remaining steps will be using **span=10m**

## Task 5: Final Sample Dataset Creation

It is time to create the dataset you will use for modeling. Now that you have the search that will be used to create your dataset, you will need to collect the features(fields) you will be modeling for and send this data over to the container for modeling. To accomplish this you will use the **fit** command.

The **fit** command is used to produce a machine learning model based on the behavior of a set of events and applies the machine learning model to the current set of search results in the search pipeline. Running the fit command by default updates the model, which is what we don't want you to do in this example. You will be using a special setting (**mode=stage**) with the **fit** command to accomplish this.

The **mode=stage** setting does not actually fit the data or modify the model. This setting indicates that you want to transfer a dataset over to the container to be worked on in the JupyterLab environment.

Notice the command below uses the **mode=stage**. It designates the algorithm(*anomaly\_detection\_ecod*) to use and the feature(*responsetime*) to model.

*Note: You created the field 'responsetime' using the eval command in your SPL search*

```
Unset ▼  
| fit MLTKContainer mode=stage algo=anomaly_detection_ecod responsetime into  
app:anomaly_detection_ecod_responsetime
```

## Steps

1. Scroll to *Option 1.b - Create final sample dataset*

```
#### Option 1.b - Create final sample dataset
```

2. Run the first cell to re-Initialize your "SplunkSearch" connection..

```
#interact with Splunk to validate your search results. Create final sample dataset  
from dsdlsupport import SplunkSearch as SplunkSearch
```

3. Copy the new search with the **fit** command from the code block below

```
search = SplunkSearch.SplunkSearch(search='| inputlookup hostperf.csv \n| eval  
_time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z") \n| timechart span=10m max(rtmax) as  
responsetime \n| fit MLTKContainer mode=stage algo=anomaly_detection_ecod  
responsetime into app:anomaly_detection_ecod_responsetime')
```

...and click on the next cell, as shown below, and paste the new search into the cell. Then click **run**.

```
#add new search here
```

*Note: The **span** in this example has been changed back to 10m for this final sample dataset which will produce a sample dataset with 1026 rows, as you will see below. Depending on which span definition you use, your sample dataset size may contain a different number of rows.*

4. Once the new search window is initialized, click **search** to create the final dataset.





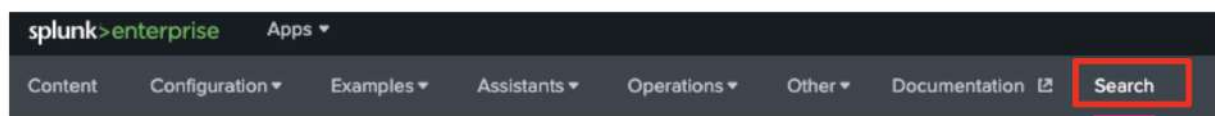
- Next, run the dataframe cell to ensure the dataframe is populated. *Remember the status bars above most show completed.*

```
df = search.as_df() ...
```

	_time	responsetime	_span
0	2015-02-18T22:10:00.000+00:00	1.275	600
1	2015-02-18T22:20:00.000+00:00	5.933	600
2	2015-02-18T22:30:00.000+00:00	5.599	600
3	2015-02-18T22:40:00.000+00:00	2.839	600
4	2015-02-18T22:50:00.000+00:00	3.702	600
...	...	...	...
1021	2015-02-26T00:20:00.000+00:00	2.898	600
1022	2015-02-26T00:30:00.000+00:00	7.564	600
1023	2015-02-26T00:40:00.000+00:00	6.938	600
1024	2024-05-20T13:30:00.000+00:00	NaN	600
1025	2024-05-20T13:40:00.000+00:00	NaN	600

1026 rows x 3 columns

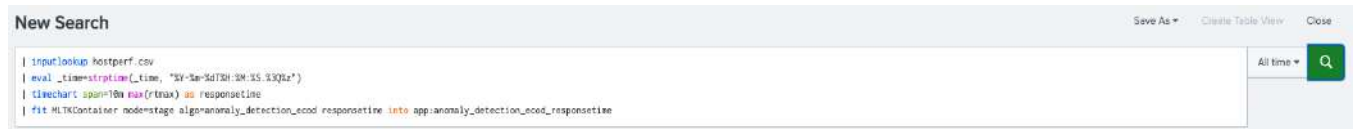
- To support use cases where dataset generation needs to be initiated directly from Splunk and not from a Jupyter Notebook(e.g., operationalized dataset generation by roles with no direct Jupyter Notebooks access), you can also push the data to the Jupyter Notebook from Splunk. Click on your Splunk tab in the browser. Then click on “Search”



- Using the same search, we used in the Jupyter notebook to create the sample dataset

```
| inputlookup hostperf.csv
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")
| timechart span=10m max(rtm) as responsetime
| fit MLTKContainer mode=stage algo=anomaly_detection_ecod responsetime into
app:anomaly_detection_ecod_responsetime
```

Run the search directly in Splunk



8. Your results will look similar

_time	responsetime
2015-02-18 23:18:00	1.275
2015-02-18 23:20:00	5.833
2015-02-18 23:22:00	5.599
2015-02-18 23:24:00	2.839
2015-02-18 23:26:00	3.782
2015-02-18 23:28:00	4.682
2015-02-18 23:30:00	8.351
2015-02-18 23:32:00	11.885
2015-02-18 23:34:00	5.519
2015-02-18 23:36:00	18.844
2015-02-18 23:38:00	26.55
2015-02-19 00:00:00	5.273
2015-02-19 00:10:00	6.981

9. Once completed, go back to your Jupyter Lab tab in your browser

10. Scroll to section 1.3.2. Option 2 - push data from Splunk, and run the first two cells

```
# this cell is not executed from MLTK and should only be used for staging data into the notebook environment
def stage(name):
    with open("data/"+name+".csv", 'r') as f:
        df = pd.read_csv(f)
    with open("data/"+name+".json", 'r') as f:
        param = json.load(f)
    return df, param

# THIS CELL IS NOT EXPORTED - free notebook cell for testing or development purposes
df, param = stage("anomaly_detection_ecod_responsetime")
```

These cells save your dataset to a .csv file and your metadata to a json file on your MLTKC container. It will then populate a dataframe and a param variable with the dataset and metadata respectively. These will be used within the Jupyter Notebook for the following modeling tasks.

11. Now, run the next two cells. You should see similar results.

```
df
```

```
[20]:
```

	responsetime
0	1.275
1	5.933
2	5.599
3	2.839
4	3.702
...	...
1019	6.892
1020	4.894
1021	2.898
1022	7.564
1023	6.938

1024 rows x 1 columns

```
[21]: param

[21]: {'options': {'params': {'mode': 'stage', 'algo': 'anomaly_detection_ecod'},
  'args': ['responsetime'],
  'feature_variables': ['responsetime'],
  'model_name': 'anomaly_detection_ecod_responsetime',
  'algo_name': 'MLTKContainer',
  'mlspl_limits': {'handle_new_cat': 'default',
    'max_distinct_cat_values': '100',
    'max_distinct_cat_values_for_classifiers': '100',
    'max_distinct_cat_values_for_scoring': '100',
    'max_fit_time': '600',
    'max_inputs': '100000',
    'max_memory_usage_mb': '4000',
    'max_model_size_mb': '30',
    'max_score_time': '600',
    'use_sampling': 'true'},
  'kfold_cv': None},
  'feature_variables': ['responsetime']}
```

## Task 6: Modeling

Now that you have created a final sample data set,, it is time to create your model. Remember modeling processing is offloaded to the containerized environment.

In this step we will initialize your model, train, test, and save your model.

*Note: In a normal model creation process, you would perform an iterative process of training, inspecting, validating, testing, and retraining your model to improve your model via code enhancement or new data sets. Since this is an interactive workshop, we will bypass those steps.*

### Steps

1. Scroll to the 1.4. Stage 2 - create and initialize a model section
2. Run the following two cells to initialize your model with the dataset and metadata.

```
# initialize your model
# available inputs: data and parameters
# returns the model object which will be used as a reference to call fit, apply and summary subsequently
def init(df,param):
    model = ECOD(contamination=0.01)
    # parallization options for ECOD:
    # ECOD(n_jobs=2)
    # most of other PyOD models would work similar, e.g. replace with Isolation Forest:
    # model = IForest()

    return model
```

```
# THIS CELL IS NOT EXPORTED - free notebook cell for testing or development purposes
model = init(df,param)
print(model)
```

```
ECOD(contamination=0.01, n_jobs=1)
```

- Now that your model is initialized, it is time for you to train the model using the **fit** command. Scroll to section 1.5. Stage 3 - fit the model, and run the first cell, which uses python code to train your model.

```
# train your model
# returns a fit info json object and may modify the model object
def fit(model,df,param):
    X = df[param['feature_variables']][0]
    X_train = np.reshape(X.to_numpy(), (len(X), 1))

    # contamination = 0.01
    model.fit(X_train)

    info = {"message": "model trained"}
    return info
```

```
# THIS CELL IS NOT EXPORTED - free notebook cell for testing or development purposes
print(fit(model,df,param))
```

```
{'message': 'model trained'}
```

Once complete, make sure you see the message “model trained”

*Note: The benefit is you can utilize the full capabilities of python to train your customized models which includes libraries and algorithms that aren't included in the Splunk MTLK. These models can be operationalized directly from Splunk searches or exported as pretrained ONNX models and imported into the MLTK for further operationalization..*

- Now that your model is trained. It is time to use the **apply** command to run the model with your test dataset and get the results.

The **apply** command is used to apply the machine learning model that was learned using the **fit** command. It repeats a selection of the **fit** command steps

*Note: this is an anomaly detection model, so you will look for outlier values in the results.*

Scroll to the 1.6. Stage 4 - apply the model section.

5. Run the cell to perform the **apply** command using your trained model and your dataset

```
# apply your model
# returns the calculated results
def apply(model,df,param):
    X = df[param['feature_variables']][0]
    X_apply = np.reshape(X.to_numpy(), (len(X), 1))

    y_hat = model.predict(X_apply) # outlier labels (0 or 1)
    y_scores = model.decision_function(X_apply) # outlier scores

    result = pd.DataFrame(y_hat, columns=['outlier'])
    return result
```

6. Run the next cell to view the results of the model

```
# THIS CELL IS NOT EXPORTED - free notebook cell for testing or development purposes
print(apply(model,df,param))
```

Notice the results include an outlier column

	outlier
0	1
1	0
2	0
3	0
4	0
...	...
1019	0
1020	0
1021	0
1022	0
1023	0

[1024 rows x 1 columns]

## 7. Proceed to run the remaining cells to save, load, and summarize the model

### 1.7. Stage 5 - save the model

```
# save model to name in expected convention "<algo_name>-<model_name>"
# for model persistence you can save the model. This is optional
def save(model,name):
    if model is not None:
        if isinstance(model,ECOD):
            from joblib import dump, load
            dump(model, MODEL_DIRECTORY + name + '.joblib') #model directory is on the container side
    return model
```

### 1.8. Stage 6 - load the model

```
# load model from name in expected convention "<algo_name>-<model_name>"
def load(name):
    model = {}
    from joblib import dump, load
    model = load(model, MODEL_DIRECTORY + name + '.joblib')
    return model
```

### 1.9. Stage 7 - provide a summary of the model

```
# return a model summary
def summary(model=None):
    returns = {"version": {"numpy": np.__version__, "pandas": pd.__version__} }
    return returns
```

## Task 7: Review Results and Visualization Creation

Now that you have created your model, you will perform the next steps to quickly operationalize the model.

Visualizations are essential to operationalizing the results from your modeling process. You can use these visualizations to populate panels in your custom Splunk dashboards which will make your results easily consumable.

In this final task, you will use your model within the Splunk UI to generate search results and create a visualization.

### Steps

1. Now click the Splunk UI browser tab and run the following search. Notice we have removed the **mode=stage** parameter.

```
| inputlookup hostperf.csv
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")
| timechart span=10m max(rtxmax) as responsetime
| fit MLTKContainer algo=anomaly_detection_ecod responsetime into
app:anomaly_detection_ecod_responsetime
```

2. Your results should look similar. Take a moment to sort the *Outlier* column to see how many were identified.



**New Search**

```

| inputlookup hostperf.csv
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")
| timechart span=10m max(rtmax) as responsetime
| fit MLTKContainer algo=anomaly_detection_ecod responsetime into app:anomaly_detection_ecod_responsetime

```

The specified span would result in too many (>50000) rows.

1,024 results (from 5/20/24 12:00:00 AM to 5/24/24 1:30:46.000 AM) No Event Sampling

Events Patterns Statistics (1,024) Visualization

20 Per Page Format Preview

_time	response	predicted_outlier
2015-02-18 22:18:00	1.275	1
2015-02-19 18:18:00	93.47	1
2015-02-19 22:38:00	75.845	1
2015-02-22 11:28:00	51.889	1
2015-02-22 13:58:00	1.584	1
2015-02-22 14:08:00	1.3719999999999999	1
2015-02-23 02:40:00	52.589	1
2015-02-23 04:18:00	1.5719999999999999	1
2015-02-25 09:48:00	1.581	1
2015-02-25 18:38:00	40.417	1
2015-02-18 22:28:00	5.933	8
2015-02-18 22:38:00	5.589	8
2015-02-18 22:48:00	2.819	8

*Note: You may receive a “The specified span would result in too many (>50000) rows.” warning. You can ignore this message for the purpose of the workshop.*

- You will now re-create the example visualization from *Task 2: step 6*.

Click on **visualization**

**New Search**

```

| inputlookup hostperf.csv
| eval _time=strptime(_time, "%Y-%m-%dT%H:%M:%S.%3Q%z")
| timechart span=10m max(rtmax) as responsetime
| fit MLTKContainer algo=anomaly_detection_ecod responsetime into app:anomaly_detection_ecod_responsetime

```

✓ 1,024 results (before 5/20/24 6:51:27.000 PM) No Event Sampling

Events Patterns Statistics (1,024) **Visualization**

- And then **Format**

Events Patterns Statistics (1,024) **Visualization**

Column Chart **Format** Trellis

5. Then click on **Chart Overlay**

The screenshot shows the 'Format' panel for a 'Column Chart'. The 'Chart Overlay' tab is selected and highlighted with a red box. The panel includes the following options:

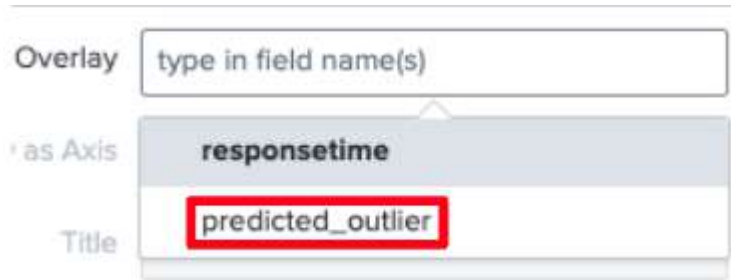
- Stack Mode:** Three radio buttons for different stacking options.
- Multi-series Mode:** Two buttons, 'Yes' and 'No'.
- Show Data Values:** Three buttons, 'Off', 'On', and 'Min/Max'.

6. Next, click inside the **Overlay** input field to activate the selection drop-down

The screenshot shows the 'Format' panel for a 'Column Chart' with the 'Chart Overlay' tab selected. The 'Overlay' input field is highlighted with a red box. The panel includes the following options:

- Overlay:** A text input field with the placeholder text 'type in field name(s)'.
- View as Axis:** Two buttons, 'On' and 'Off'.
- Title:** A dropdown menu with 'Default' selected.
- Scale:** Three buttons, 'Inherit', 'Linear', and 'Log'.
- Interval:** A text input field.
- Min Value:** A text input field.
- Max Value:** A text input field.
- Number Abbreviations:** Two buttons, 'Off' and 'On'.

7. ...and select **predicted\_outlier**



8. Next click the **On** button for **View as Axis**



...and close the pop-up window.

9. You have now created a new visualization using the model you've created in the DSDL.



10. Try adjusting the “**span=**”. We recommend nothing less than 2m for this dataset.



In this module, you were introduced to some of the capabilities of the Splunk Data Science and Deep Learning (DSDL) app (e.g., Containerization, Jupyter/Splunk integration, custom model training & deployment, and model operationalization via dashboard visualization).

Remember, the Data Science and Machine Learning (DSDL) app extends the Splunk Machine Learning Toolkit (MLTK) with prebuilt Docker containers for TensorFlow, PyTorch, and a collection of data science, NLP, and classical machine learning libraries. By using predefined workflows for rapid development with Jupyter Lab Notebooks, the DSDL enables you to build, test, and operationalize your custom models with Splunk. For your process heavy modeling requirements, the DSDL enables you to leverage GPUs for compute intense training tasks and flexibly deploy models on CPU or GPU enabled containers. Use the DSDL to perform custom AI modeling or to create pretrained ONNX models to upload into the MLTK for further operationalization.