



# Introduction to HPCC Systems, ECL, and ECL Machine Learning

Dan S. Camper (Sr. Architect) and Lili Xu (Software Engineer III)  
HPCC Solutions Lab

January 17, 2019

# Agenda

- Who and What
  - About LexisNexis Risk Solutions
  - HPCC Systems Intro
- Enterprise Control Language
  - What is it?
  - Basic, But Important, Stuff
  - Common Data Types
  - Important Minutia
- Stock Data
  - Step 0: Raw Data
  - Step 1: Profile the Data
  - Step 2: Enhance the Data
  - Step 3: Analytics
- Setting Up Your Workstation
  - Browser Bookmark
  - Source Code and IDE Installation Instructions
- Stock Data: Code
  - Code -> Purpose
  - Let's Play With The Code

# About LexisNexis Risk Solutions

*We believe in the power of data and advanced analytics for better risk management.*



## Data, Analytics and Technology

LexisNexis Risk Solutions leverages its industry-leading Big Data computing platform with vast data assets and a proprietary fast-linking technology to enable businesses of all sizes to better analyze and understand data at scale, improving time-to-results and decisions.



## Solutions We Provide

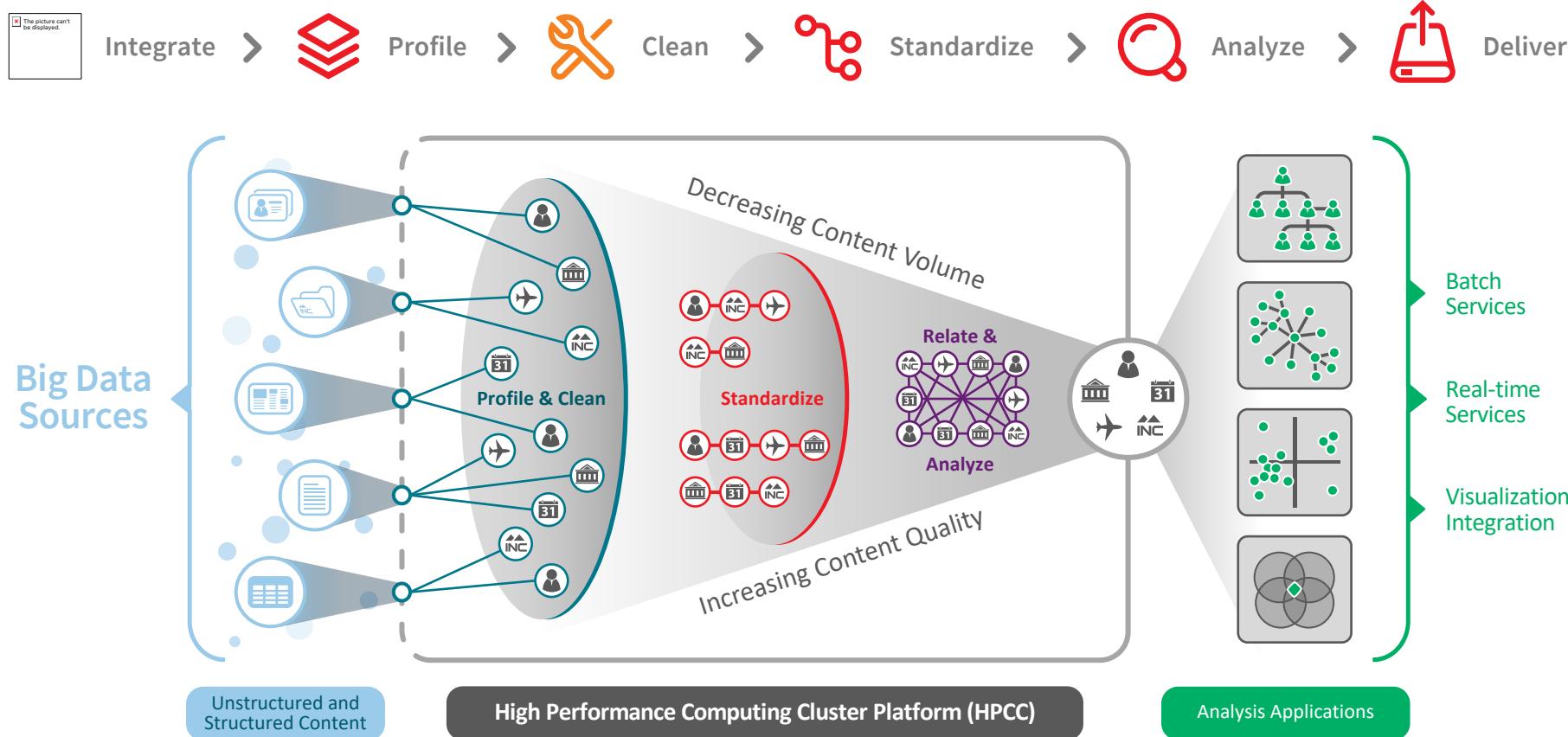
With our solutions, our customers transform their risk decision making and are empowered to make better decisions easier. We help them with business challenges like fighting fraud, facilitating compliance, streamlining workflows and increasing efficiencies, improving health outcomes and keeping communities safe by providing timely insights for business decisions.



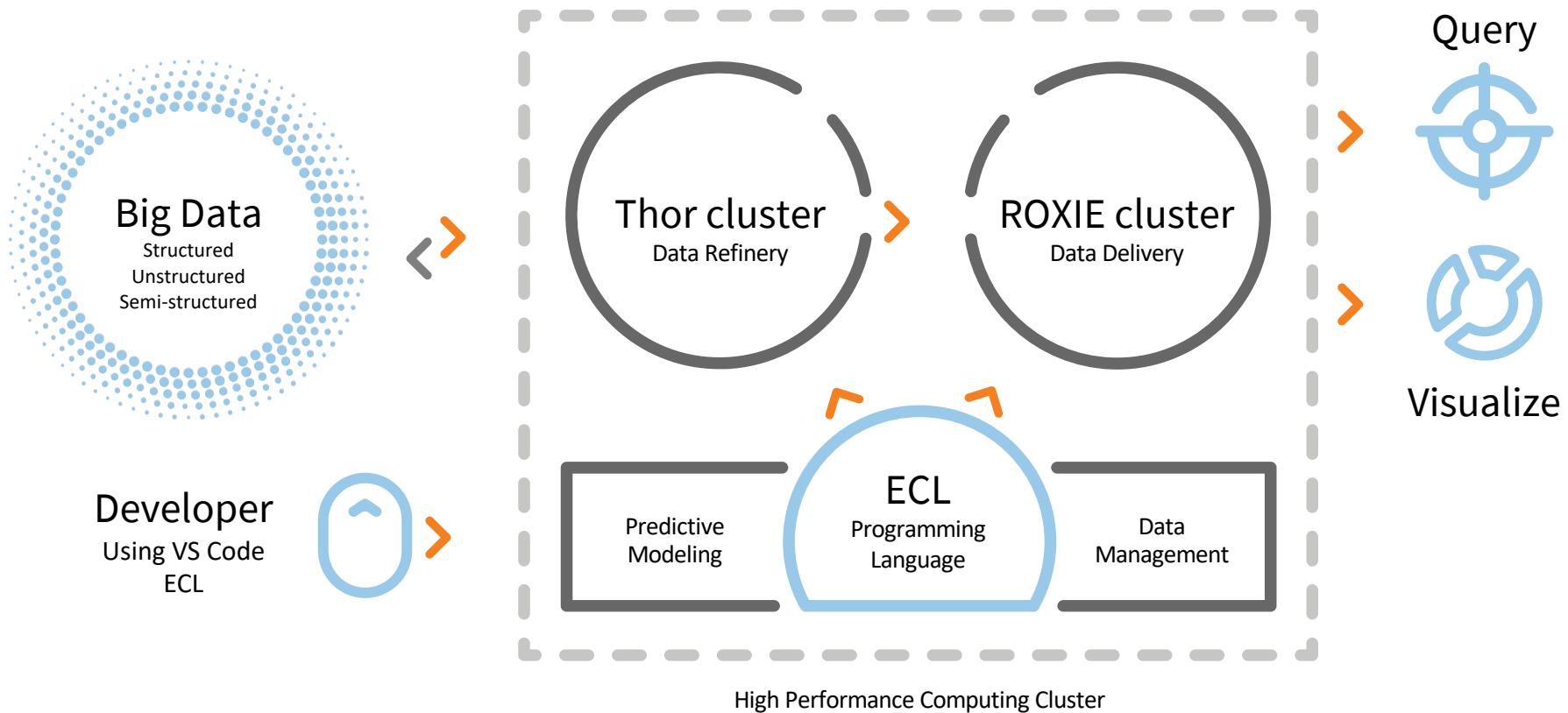
## Markets We Serve

LexisNexis Risk Solutions provides solutions across multiple industries, including Insurance, Financial Services, Collections and Recovery, Retail, Health Care and Communications. We also work with all levels of local, state, and federal governments and their agencies. We serve customers in more than 100 countries.

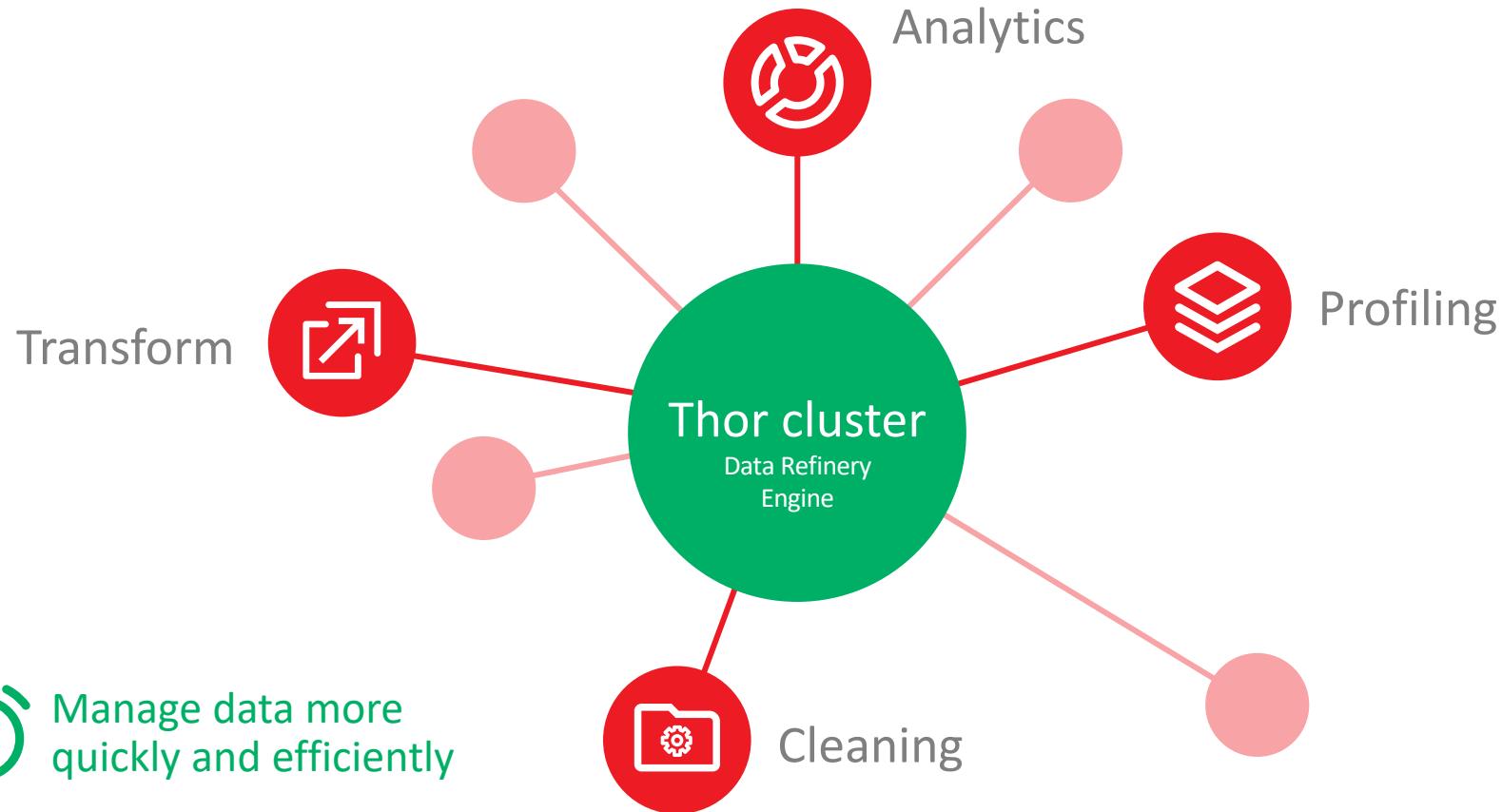
# HPCC Systems (Small to Big Data) ETL



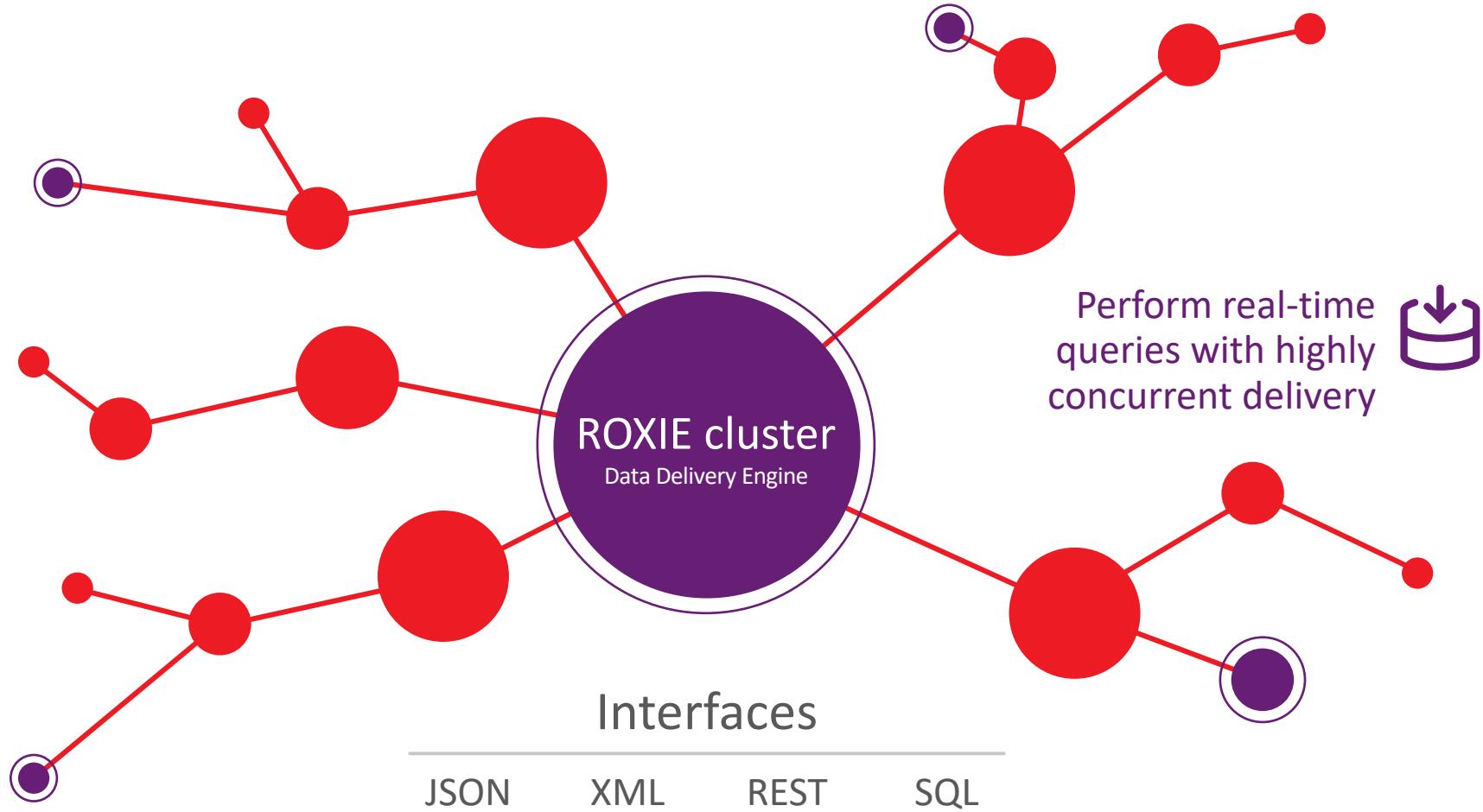
# The HPCC Systems Components



# An Intro to Thor



# An Intro to ROXIE

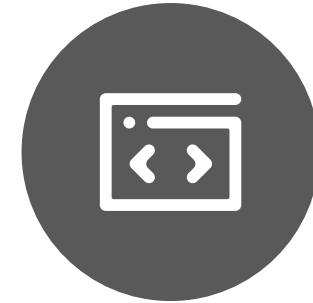


# An Intro to ECL



```
1 IMPORT $ AS XHRGlobal;
2 IMPORT XpertHR;
3 IMPORT STD;
4
5 EXPORT Util := MODULE
6
7   EXPORT BatchId := FUNCTION
8     currentSeconds := STD.Date.CurrentSeconds() : INDEPENDENT;
9     RETURN STD.Date.SecondsToString(currentSeconds);
10    END;
11
12  EXPORT TransactionalSuperFileName(XHRGlobal.Interfaces.IProfile profile, XpertHR.Types.PL)
13    RETURN '~' + profile.CustomerFilePrefix + '::imported:' + platformType + '::Transact';
14  END;
15
16  EXPORT TransactionalFileName(XHRGlobal.Interfaces.IProfile profile, XpertHR.Types.PlatformType)
17    RETURN TransactionalSuperFileName(profile, platformType) + '::' + batchId;
18  END;
19
20  EXPORT TransactionalBaseFileName(XHRGlobal.Interfaces.IProfile profile, XpertHR.Types.PlatformType)
21    RETURN TransactionalSuperFileName(profile, platformType) + '::Base';
22  END;
23
24  EXPORT CreateTransactionalFile(incoming, profile, platformType) := FUNCTIONMACRO
25    LOCAL fileName := Util.TransactionalFileName(profile, platformType, Util.BatchId);
26    LOCAL superFileName := Util.TransactionalsuperFileName(profile, platformType);
27
28    LOCAL createFile := OUTPUT(incoming,,fileName, COMPRESSED);
29
30    RETURN SEQUENTIAL()
31      Std.File.CreateSuperFile(superFileName, allowExist := TRUE),
32      createFile,
33      Std.File.StartSuperFileTransaction(),
34      Std.File.AddSuperFile(superFileName, fileName),
35      Std.File.FinishSuperFileTransaction()
36    ];

```



Powerful language  
built for big data

How to do it



vs.



What to do

# The ECL Programming Language

- Designed to allow data operations to be specified in a manner which is easy to optimize and parallelize
- Declarative in nature (“what you want done, rather than how to do it”)
- Extremely succinct
- Data centric, extensible and internally abstract
  - Minimizes data movement
  - Does work once
  - Keeps all nodes equally busy
- Implicitly parallel
- Compared to MapReduce:
  - MAP -> PROJECT/TRANSFORM
  - SHUFFLE (phase 1) -> DISTRIBUTE(,HASH(KeyValue))
  - SHUFFLE (phase 2) -> SORT(,LOCAL)
  - REDUCE -> ROLLUP(,Key,LOCAL)

Return list of actors appearing most often in movies

```
FO := IMDB.File_actors;  
CountActors := RECORD  
    FO.ActorName;  
    Unsigned C := COUNT(GROUP)  
END
```

```
MoviesIn := TABLE(FO, CountActors, ActorName);
```

```
OUTPUT (TOPN(MoviesIn, 100, -C));
```

# SQL vs. ECL (table and dataset)

SQL

```
CREATE TABLE Products (
    productCode VARCHAR(15) NOT NULL,
    productName VARCHAR(70) NOT NULL,
    productLine VARCHAR(50) NOT NULL,
    productScale VARCHAR(10) NOT NULL,
    productVendor VARCHAR(50) NOT NULL,
    productDescription TEXT NOT NULL,
    quantityInStock SMALLINT NOT NULL,
    buyPrice DOUBLE NOT NULL,
    MSRP DOUBLE NOT NULL,
    PRIMARY KEY (productCode)
);
```

ECL

```
Product := RECORD
    STRING productCode;
    STRING productName;
    STRING productLine;
    STRING productScale;
    STRING productVendor;
    STRING productDescription;
    INTEGER quantityInStock;
    DECIMAL7_2 buyPrice;
    DECIMAL7_2 MSRP;
END;

products := DATASET('~/cm/:products', Product, csv);
```

# SQL vs. ECL (sort, count, group)

SQL

```
select * from products  
order by productName
```

```
select COUNT(*) from products  
where productLine='Vintage Cars'
```

```
select * from products  
group by productLine
```

ECL

```
SORT(products, productName);
```

```
COUNT(  
products(productLine='Vintage Cars'));
```

```
GROUP(  
SORT(products, productLine),  
productLine);
```

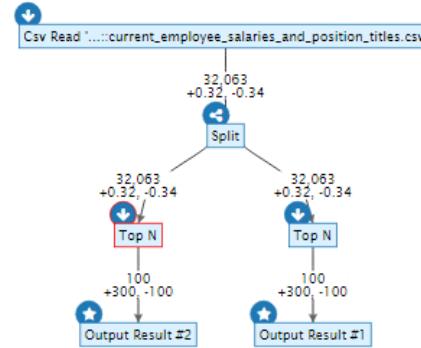
# ECL: A Powerful Data Flow Language

## How you code

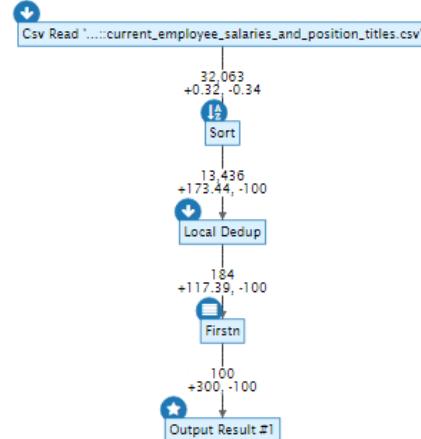
```
dsEmpSalary := DATASET(...);  
OUTPUT(SORT(dsEmpSalary, position));  
OUTPUT(SORT(dsEmpSalary, department));
```



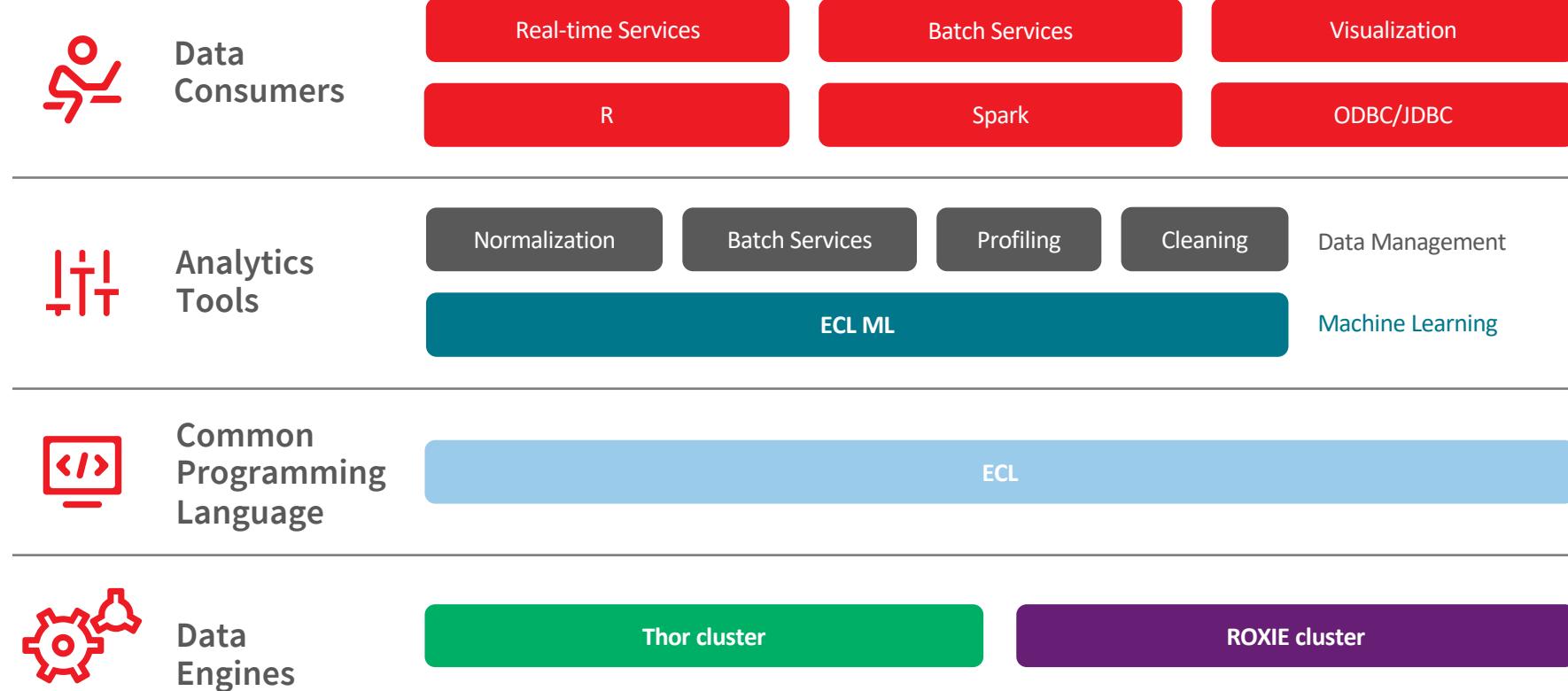
## How the system executes it



```
dsEmpSalary := DATASET(...);  
dsSortedEmp:=SORT(dsEmpSalary, position);  
OUTPUT(DEDUP(dsSortedEmp, position));
```



# Technology — Layer View



# Enterprise Control Language (ECL)



# ECL: What is it?

- Declarative Programming Language
  - “... a programming paradigm ... that expresses the logic of a computation without describing its control flow.” – *Wikipedia*
- Designed For Big Data Scenarios
- Any Cluster Size
- Source-to-source compiler
  - ECL code translated to C++ that is compiled to shared libraries and executed within a custom framework
- Can use external libraries and embedded code in certain other languages
  - C/C++, Python, R, Javascript
- Useful Documentation
  - Language Reference
    - <https://hpccsystems.com/training/documentation/ecl-language-reference/html>
  - Standard Library
    - <https://hpccsystems.com/training/documentation/standard-library-reference/html>

# ECL: Basic, But Important, Stuff

- Two Statement Types
  - *Definition*
    - Assign an expression to an attribute
    - Example: `foo := bar * 2;`
  - *Action*
    - Do something that affects the outside world
      - Create a file, talk to a service, etc.
    - Example: `OUTPUT(foo, NAMED('foo_value'));`
- Plot Twist
  - You can define an attribute as an action
  - Example: `outputFoo := OUTPUT(foo, NAMED('foo_value'));`

# ECL: Common Data Types

- Character
  - STRING[n]
  - UTF8
  - UNICODE[\_locale][n]
- Numeric
  - INTEGER[n]
  - UNSIGNED[n]
  - REAL[n]
  - DECIMAL<n>[\_y]
  - UDECIMAL<n>[\_y]
- Other
  - BOOLEAN
  - SET OF <type>
  - RECORD
  - DATASET

# ECL: Important Minutia

- Case-insensitive
- Whitespace insensitive (within reason)
- String literals are quoted with apostrophes
- Semicolon terminator
- C++/Java style commenting
- Definition (assignment) operator is :=
- Equality test operator is =
- Attributes can be defined only once
- Single-pass code parser
  - Only previously-defined attributes can be referenced
- Only those definitions that contribute to a result are actually used
- There are no loops

# Stock Data



## Step 0: Raw Data

- 3 U.S. stock exchanges
    - NYSE
    - NASDAQ
    - AMEX
  - 16 years of data
  - 1GB of data in tab-delimited format
  - ~21M rows of data
- Data fields:
    - trade\_date
    - exchange\_code
    - stock\_symbol
    - opening\_price
    - high\_price
    - low\_price
    - closing\_price
    - shares\_traded
    - share\_value

# Step 1: Profile the Data

attribute	given_attribute_type	best_attribute_type	rec_count	fill_count	fill_rate	cardinality	modes		min_length	max_length	ave_length	popular_patterns	data_pattern	rec_count	example
							value	rec_count							
trade_date	unsigned4	unsigned4	20959177	20959177	100	4274	20020101	6605	8	8	8	99999999		20959177	20020101
exchange_code	string1	string1	20959177	20959177	100	3	0	9794864	1	1	1	A		20959177	0
stock_symbol	string9	string9	20959177	20959177	100	12400	ORCL	7171	1	9	3	AAA		9957675	AAA
												AAAA		9223864	AAAE
												AAAAA		860033	AAIIIE
												AA		830431	AA
												A		74939	A
												AAA-A		8795	CMG-B
												A-AAA		2727	I-COMP
												AAA-A-A		544	OIBR-C
												AA-A		168	UA-C
												AAA-AAA9		1	BLDP-OI
opening_price	decimal19_2	decimal19_2	20959177	20596033	98.267375	73073	0.01	144026	1	9	4	99.99		10193305	10.01
												9.99		5891451	0.01
												99.9		2133440	10.1
												9.9		1072363	0.1
												999.99		495726	100.01
												99		477761	10
												9		157129	1
												999.9		100827	100.1
												999		49137	100
												9999.99		8141	1001.91
												9999		7744	1000
												9999.9		2216	1001.6

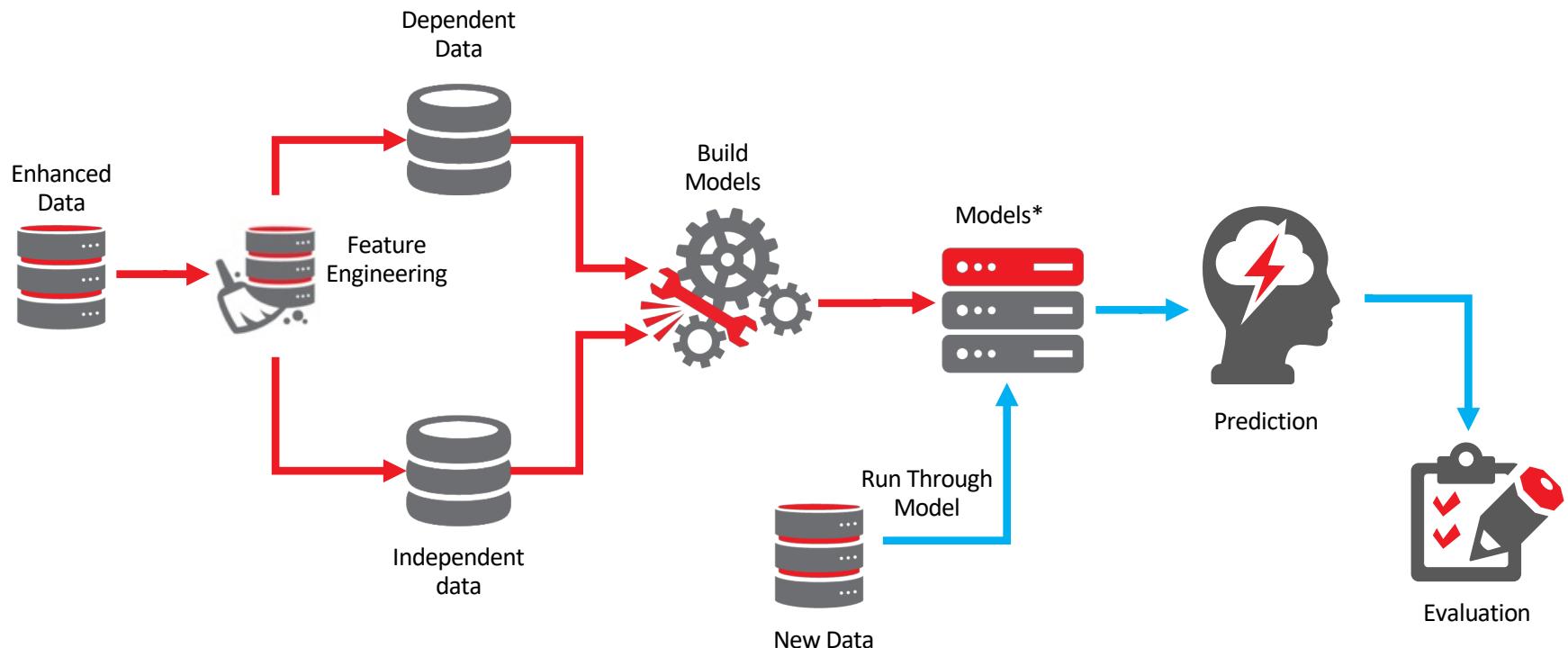
1 - 9 of 9 results

« < ] > » 50

## Step 2: Enhance the Data

- Original Fields:
  - trade\_date
  - exchange\_code
  - stock\_symbol
  - opening\_price
  - high\_price
  - low\_price
  - closing\_price
  - shares\_traded
  - share\_value
- Additional Fields:
  - symbol (exchange:stock\_symbol)
  - trade\_year
  - trade\_month
  - trade\_day
  - trade\_day\_of\_week
  - trade\_quarter
  - trade\_day\_of\_year
  - trade\_day\_of\_quarter
  - opening\_price\_change
  - closing\_price\_change
  - shares\_traded\_change
  - moving\_ave\_opening\_price
  - moving\_ave\_high\_price
  - moving\_ave\_low\_price
  - moving\_ave\_closing\_price
  - shares\_traded\_change\_rate
  - direction

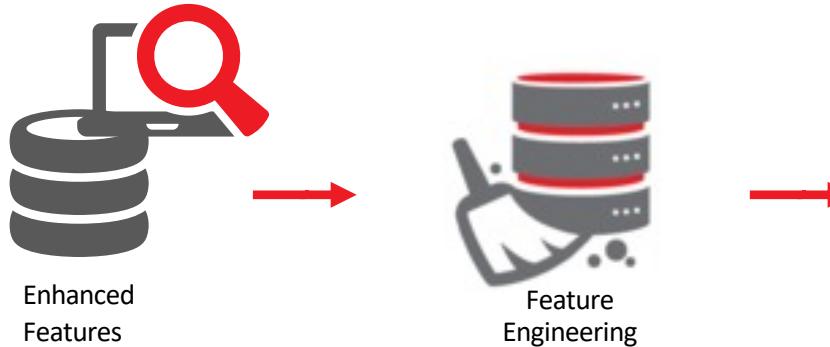
## Step 3: Analytics



# Step 3: Machine Learning on HPCC Systems Platform



## Step 3: Analytics Cont.,

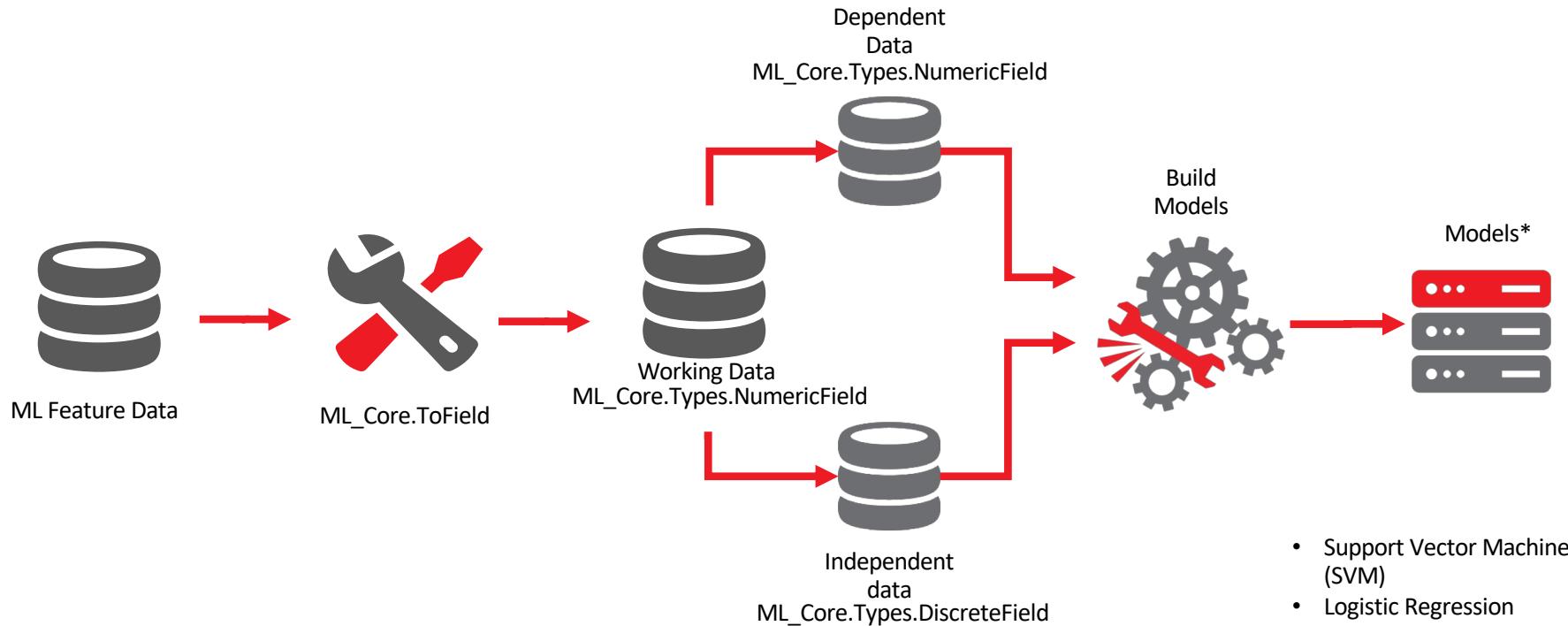


Direction	Description
0	Closing price is decreasing
1	Closing price is increasing

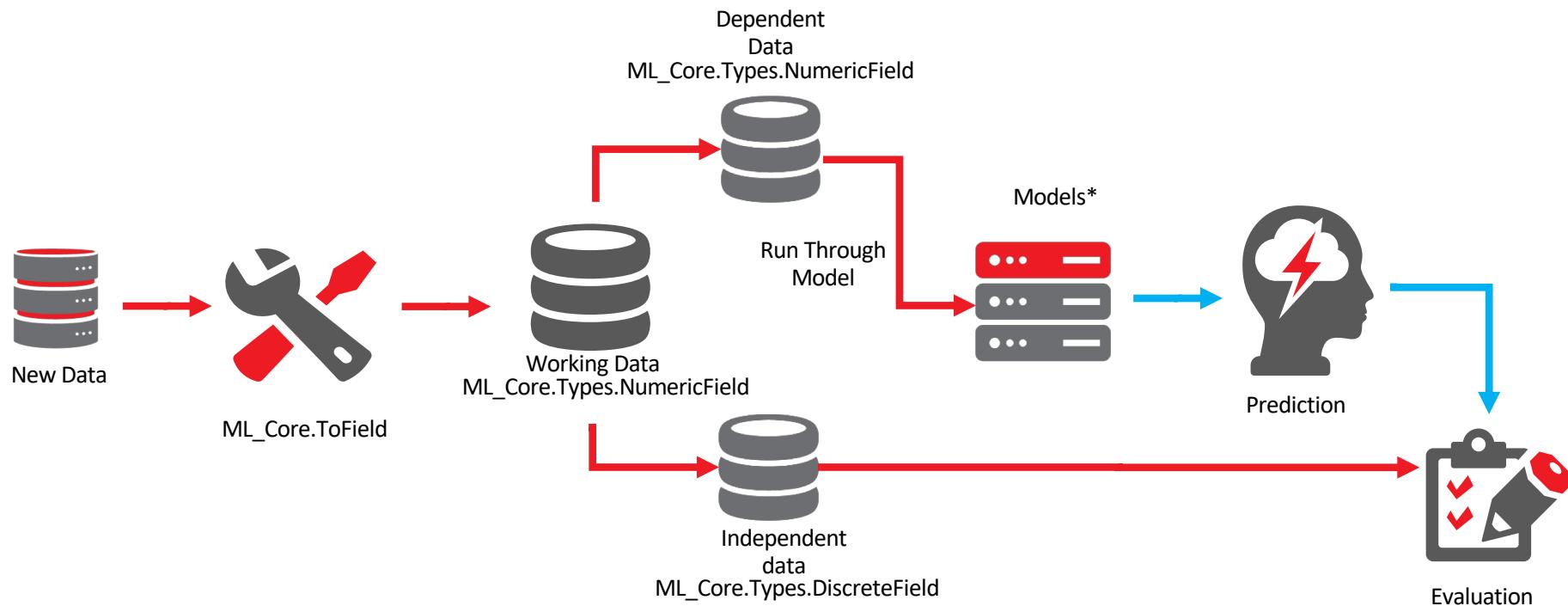
- ML Feature Fields:

- symbol (exchange:stock\_symbol)
- trade\_date
- opening\_price\_change
- closing\_price\_change
- shares\_traded\_change
- moving\_ave\_opening\_price
- moving\_ave\_high\_price
- moving\_ave\_low\_price
- moving\_ave\_closing\_price
- shares\_traded\_change\_rate
- direction

## Step 3: Analytics Cont.,



## Step 3: Analytics Cont.,



- Support Vector Machine (SVM)
- Logistic Regression

# Setting Up Your Workstation



# Setting Up Your Workstation – Browser Bookmark

<http://18.216.233.32:8010/>

The screenshot shows the ECL Watch web application interface. At the top, there is a blue header bar with the title "ECL Watch" and several icons. Below the header is a navigation bar with tabs: "Activity" (which is selected), "Event Scheduler", and "Search Results". The main content area is titled "Activity" and contains a table with the following data:

<input type="checkbox"/>	Target/Wuid	Graph	State	Owner	Job Name
<input type="checkbox"/>	hthor				
<input type="checkbox"/>	thor				
<input type="checkbox"/>	thor_roxie				
<input type="checkbox"/>	roxie				
<input type="checkbox"/>	hthor.eclserver,thor.eclserver,rox...				
<input type="checkbox"/>	dfuserver_queue				

# Source Code and IDE Installation Instructions

<https://github.com/dcamper/StockTrade>



The screenshot shows the GitHub repository page for 'dcamper / StockTrade'. The page displays the following information:

- Repository name: dcamper / StockTrade
- Owner: dcamper
- Watchers: 2
- Stars: 0
- Forks: 1
- Issues: 0
- Pull requests: 0
- Projects: 0
- Wiki
- Insights
- Settings

For January 2019 Workshop

Manage topics

5 commits, 1 branch, 0 releases, 2 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

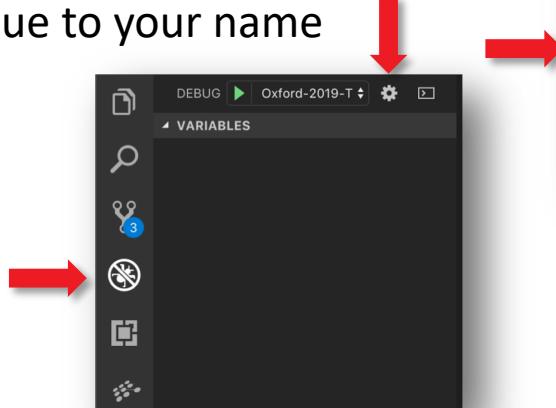
File / Commit	Description	Time
vscode	Initial commit	28 days ago
DataPatterns	Initial commit	28 days ago
GLM	Initial commit	28 days ago
LearningTrees	Initial commit	28 days ago
LinearRegression	Initial commit	28 days ago
LogisticRegression	Initial commit	28 days ago
ML_Core	Initial commit	28 days ago
PBblas	Initial commit	28 days ago
StockData	added SVM to predict stock market trend	21 hours ago
SupportVectorMachines	Initial commit	28 days ago
.gitignore	Initial commit	28 days ago
vscode_hpcc_install.pdf	Updated for client tools v7.0.4	22 days ago

Help people interested in this repository understand your project by adding a README.

Add a README

# Source Code and IDE Installation Instructions

- Clone GitHub repo
  - <https://github.com/dcamper/StockTrade>
- Follow VS Code installation instructions
  - Do not install optional bundles
- Within VS Code, open the cloned StockTrade directory
- Modify the launch.json file
  - Change “user” value to your name
  - Save and close

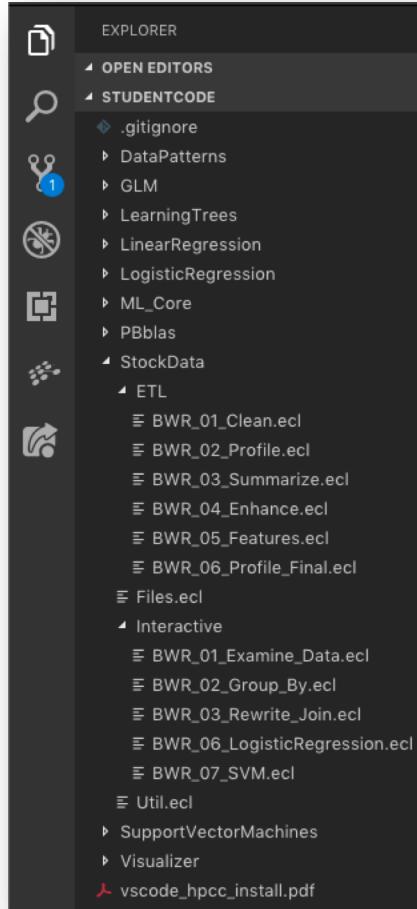


```
{} launch.json x
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Oxford-2019-Thor-Submit",
6       "type": "ecl",
7       "request": "launch",
8       "mode": "submit",
9       "workspace": "${workspaceRoot}",
10      "program": "${file}",
11      "protocol": "http",
12      "serverAddress": "18.216.233.32",
13      "port": 8010,
14      "rejectUnauthorized": false,
15      "targetCluster": "thor",
16      "eclccPath": "${config:ecl.eclccPath}",
17      "eclccArgs": [],
18      "includeFolders": "${config:ecl.includeFolders}",
19      "legacyMode": "${config:ecl.legacyMode}",
20      "resultLimit": 100,
21      "user": "dcamper",
22      "password": ""
23    }
24  ]
25 }
```

## Stock Data: Code



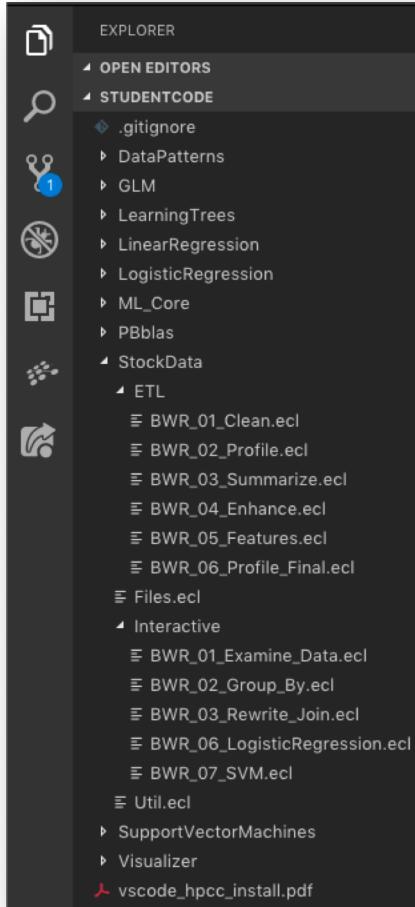
# Code -> Purpose



← Code used to clean, profile and enhance  
*Provided for review; please don't execute!*

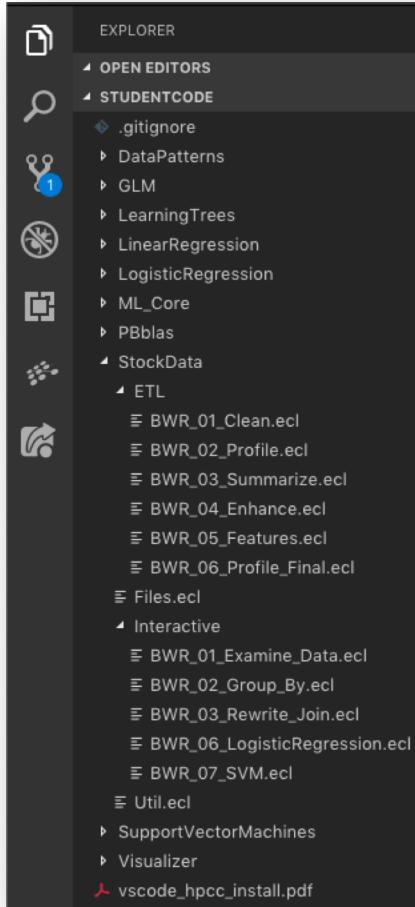
← Example code you can run, modify, and play with

# Code: ETL Directory Contents



- BWR\_01\_Clean.ecl
  - *Simple rewrite of the raw data (where all fields were STRING) as a fully datatyped dataset*
- BWR\_02\_Profile.ecl
  - *Perform data profiling against the original data that has has datatyped fields*
- BWR\_03\_Summarize.ecl
  - *Create some per-symbol statistics and write the results to a separate file*
- BWR\_04\_Enhance.ecl
  - *Pull apart the data in certain fields, like date, and create new fields containing features*
- BWR\_05\_Features.ecl
  - *Append some other features that will be used for machine learning*
- BWR\_06\_Profile\_File.ecl
  - *Perform data profiling against the final dataset, which includes appended fields*

# Code: Interactive Directory Contents



- **BWR\_01\_Examine\_Data.ecl**
  - *Output a sample of the data and some minor statistics about it*
- **BWR\_02\_Group\_By.ecl**
  - *Demonstrate the TABLE() function by computing the average number of shares traded for each day of the week*
- **BWR\_03\_Rewrite\_Join.ecl**
  - *Append a new field that shows how the closing price compares against that year's median value*
- **BWR\_04\_LogisticRegression.ecl**
  - *Perform a logistic regression against the movement of Apple's closing price*
- **BWR\_05\_SVM.ecl**
  - *A different model for predicting the movement of Apple's closing price*

# Let's Play With The Code



# Useful Links

- [Open Source HPCC Systems Platform: Home Page](#)
- [Internship Program](#)
- [Online Training](#)
- [Download Page](#)
- [Our GitHub portal](#)
- [Community Forums](#)
- [Getting Started with ECL](#)
- [Advanced ECL](#)
- [Latest Release and Documentation](#)
- [Supported plugins, connectors, third party modules and bundles](#)
- [Machine Learning on HPCC Systems](#)



Join our Community

Help us make HPCC Systems better. Register on our community portal.

# *Questions?*





CONFIDENTIAL