Data Science with Groovy

Presented by
Dr Paul King
Dr Paul King

OCI Groovy Lead
V.P. and PMC Chair Apache Groovy

https://speakerdeck.com/paulk/groovy-data-science
https://github.com/paulk-asert/groovy-data-science
@paulk_asert
WE ARE SOFTWARE ENGINEERS.

We deliver mission-critical software solutions that accelerate innovation within your organization and stand up to the evolving demands of your business.
WE ARE SOFTWARE ENGINEERS.

We deliver mission-critical software solutions that accelerate innovation within your organization and stand up to the evolving demands of your business.

AREAS OF EXPERTISE

**INDUSTRIAL IoT**
We equip industrial environments with seamless connectivity and real-time analytics that reduce operating costs and deliver on customer demands.

**MACHINE LEARNING**
We can modernize your legacy applications and enable scalable, integrated AI capabilities tailored to your unique sets of data.

**BLOCKCHAIN CONSULTING**
We are at the forefront of blockchain technology, and we have practical, real-world experience with its implementation.

**CLOUD SOLUTIONS**
We combine software engineering expertise with cloud-native architecture to accelerate innovation within your organization.
What is Groovy?

It’s like a super version of Java:

- Supports most Java syntax but allows simpler syntax for many constructs
- Supports all Java libraries but provides many extensions and its own productivity libraries
- Has both a static and dynamic nature
- Extensible language and tooling
Groovy has a low learning curve, particularly for Java developers.
Groovy is Java's friend

Seamless integration
• IDEs provide cross-language compile, navigation, and refactoring
• Arbitrarily mix source language
• Drop-in replace any class
• Overloaded methods
• Syntax alignment
• Shared data types
import java.util.List;
import java.util.ArrayList;

class Main {
    private List keepShorterThan(List strings, int length) {
        List result = new ArrayList();
        for (int i = 0; i < strings.size(); i++) {
            String s = (String) strings.get(i);
            if (s.length() < length) {
                result.add(s);
            }
        }
        return result;
    }

    public static void main(String[] args) {
        List names = new ArrayList();
        names.add("Ted"); names.add("Fred");
        names.add("Jed"); names.add("Ned");
        System.out.println(names);
        Main m = new Main();
        List shortNames = m.keepShorterThan(names, 4);
        System.out.println(shortNames.size());
        for (int i = 0; i < shortNames.size(); i++) {
            String s = (String) shortNames.get(i);
            System.out.println(s);
        }
    }
}
Concise syntax including DSL friendly

```java
import java.util.List;
import java.util.ArrayList;

class Main {
    private List keepShorterThan(List strings, int length) {
        List result = new ArrayList();
        for (int i = 0; i < strings.size(); i++) {
            String s = (String) strings.get(i);
            if (s.length() < length) {
                result.add(s);
            }
        }
        return result;
    }

    public static void main(String[] args) {
        List names = new ArrayList();
        names.add("Ted"); names.add("Fred");
        names.add("Jed"); names.add("Ned");
        System.out.println(names);
        Main m = new Main();
        List shortNames = m.keepShorterThan(names, 4);
        System.out.println(shortNames.size());
        for (int i = 0; i < shortNames.size(); i++) {
            String s = (String) shortNames.get(i);
            System.out.println(s);
        }
    }
}
```

given the names "Ted", "Fred", "Jed" and "Ned"
display all the names
display the number of names having size less than 4
display the names having size less than 4
Metaprogramming

```java
@Nonnull
public final class Person {
    private final String first;
    private final String last;

    public String getFirst() {
        return first;
    }

    public String getLastName() {
        return last;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((first == null) ? 0 : first.hashCode());
        result = prime * result + ((last == null) ? 0 : last.hashCode());
        return result;
    }

    public Person(String first, String last) {
        this.first = first;
        this.last = last;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        Person other = (Person) obj;
        if (first == null) {
            if (other.first == null) return true;
        } else if (!first.equals(other.first)) return false;
        if (last == null) {
            if (other.last == null) return true;
        } else if (!last.equals(other.last)) return false;
        return true;
    }

    @Override
    public String toString() {
        return "Person(first: " + first + ", last: " + last + ")";
    }
}
```

Groovy's metaprogramming capabilities allow complex design patterns to be encapsulated within simple constructs.
Matrix manipulation example

```java
import org.apache.commons.math3.linear.*;

public class MatrixMain {
    public static void main(String[] args) {
        double[][] matrixData = {{1d, 2d, 3d}, {2d, 5d, 3d}};
        RealMatrix m = MatrixUtils.createRealMatrix(matrixData);

        double[][] matrixData2 = {{1d, 2d}, {2d, 5d}, {1d, 7d}};
        RealMatrix n = new Array2DRowRealMatrix(matrixData2);
        RealMatrix o = m.multiply(n);

        // Invert p, using LU decomposition
        RealMatrix oInverse = new LUDecomposition(o).getSolver().getInverse();
        RealMatrix p = oInverse.scalarAdd(1d).scalarMultiply(2d);
        RealMatrix q = o.add(p.power(2));
        System.out.println(q);
    }
}
```

```java
Array2DRowRealMatrix{{15.1379501385,40.488531856},{21.4354570637,59.5951246537}}
```
Operator overloading and extensible tools

```python
def m = [[1d, 2d, 3d], [2d, 5d, 3d]] as Matrix
def n = [[1d, 2d], [2d, 5d], [1d, 7d]] as Matrix
def o = m * n
def p = (~o + 1) * 2
o + p ** 2
```

Result: 

\[
\begin{pmatrix}
15.137950138504156 & 40.48853185595568 \\
21.43545706371191 & 59.59512465373961
\end{pmatrix}
\]
showing @Grab, power asserts and natural language processing
REPL for Groovy (groovysh)

The groovysh REPL is where you can program interactively.
Text file? No it’s a statically typed Groovy program which won’t compile if the animal names aren’t valid.
A constraint programming solver is used to find the solution.
cranes have 2 legs
tortoises have 4 legs
millipedes have 1000 legs
there are 8 animals
there are 1020 legs

display solution

Cranes 4
Tortoises 3
Millipedes 1
Groovy frameworks plus entire Java ecosystem
Groovy frameworks plus entire Java ecosystem
Groovy frameworks plus entire Java ecosystem

plugins {
    id 'groovy'
}

repositories {
    jcenter()
}

dependencies {
    implementation 'org.codehaus.groovy:groovy-all:2.5.7'
    testImplementation 'org.spockframework:spock-core:1.2-groovy-2.5'
}
Groovy frameworks plus entire Java ecosystem

<table>
<thead>
<tr>
<th>Data Parallelism: Fork/Join Map/Reduce</th>
<th>Fixed coordination (for collections)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Explicit coordination</td>
</tr>
<tr>
<td>Safe Agents</td>
<td>Delegated coordination</td>
</tr>
<tr>
<td>Dataflow</td>
<td>Implicit coordination</td>
</tr>
</tbody>
</table>

Rich ecosystem.

Source: ReGinA – Groovy in Action, Manning, 2nd edition
Data Science Process

- Research Goals
- Obtain Data
- Data Preparation
- Data Exploration
- Data Modeling
- Visualization
Data Science Process

Research Goals

- Obtain Data
- Data Preparation
- Data Exploration
- Data Modeling
- Visualization

Supporting Components:
- Data processing platforms
- Graphics processing
- Integration
- Data ingestion
- Data storage
- Math libraries
- Modeling algorithms
- Deployment
Libraries/Tools/Frameworks for Data Science - R

- **dplyr**: powerful library for data wrangling, works with local data frames and remote database tables, precise and simple command syntax
- **data.table**: quick aggregation of large data, laconic flexible syntax and a wide suite of useful functions, friendly file reader and parallel file writer
- **lubridate**: a set of functions to work with date and time format, easy and fast parsing of date-time data, expanded mathematical operations on time data
- **jsonlite**: robust and quick parsing JSON objects in R, great tool for interacting with web APIs and building pipelines, functions to stream, validate, and prettyfy JSON data
- **ggplot**: powerful implementation of the grammar of graphics visualization, developed static graphics system, takes care of plot specifications, abilities to visualize correlation matrices and confidence intervals, contains algorithms to do smooth rendering, flexible appearance details settings
- **lattice**: high-level visualization system, emphasis on multivariate data, efficiently copes with nonstandard requirements
- **ggvis**: rich features and plenty of available charts, web-based interface for building visualizations, abilities to make ggplot2 graphics interactive
- **DT**: implementation of an interactive grammar of graphic, incorporates interactive programming model and dplyr grammar of data transformation
- **DataTables**: displays R matrices and data frames as interactive HTML tables, creates sortable tables with a minimum of code, many useful features and styling options for tables
- **plotly**: interactive 5 charts from R, tools for creation, customization, and sharing

---

Libraries/Tools/Frameworks for Data Science - Python

Top 10 Data Science Libraries
by GitHub Contributors, Commits and Size (size of the circle)

Obtain data

Identify sources
• Internal/External (data providers)
• Databases, files, events, IoT
• Unstructured/structured
• Text, XML, JSON, YAML, CSV/spreadsheets, audio, images, video, web services
• Apache Tika
• JExcel
• Apache POI

Clarify ownership
Check quality

```python
def jsonSlurper = new JsonSlurper()
def object = jsonSlurper.parseText('{ "myList": [4, 8, 15, 16, 23, 42] }')
assert object instanceof Map
assert object.myList instanceof List
assert object.myList == [4, 8, 15, 16, 23, 42]

def response = new XmlSlurper().parseText(books)
def authorResult = response.value.books.book[0].author
assert authorResult.text() == 'Miguel de Cervantes'

def qry = 'SELECT * FROM Author'
assert sql.rows(qry, 1, 3).*firstname == ['Dierk', 'Paul', 'Guillaume']
assert sql.rows(qry, 4, 3).*firstname == ['Hamlet', 'Cedric', 'Erik']
assert sql.rows(qry, 7, 3).*firstname == ['Jon']
```
Data preparation

Collections:
• Java: lists, maps, sets
• Groovy: literal notation, GDK methods, GPath
• Libraries
  • Google Guava https://github.com/google/guava
  • Apache Common Collections https://commons.apache.org/collections/

DataFrames:
• Joinery https://cardillo.github.io/joinery/
  A data frame implementation in the spirit of Pandas or R data frames with show/plot
• Tablesaw https://jtablesaw.github.io/tablesaw/
  Java dataframe and visualization library
• Apache Spark DataFrames https://spark.apache.org/docs/latest/sql-programming-guide.html
  Equivalent to a table in a relational database or a data frame in R/Python
• Paleo https://github.com/netzwerg/paleo
  Immutable Java 8 data frames with typed columns (including primitives)
**Data exploration - DEX**

DEX : The Data Explorer

**Dex**

Dex is a powerful tool for data science. It is written in Groovy and Java on top of JavaFX and offers the ability to:

- Read in data from a variety of sources such as files, programs and a variety of databases.
- Transform the data in a powerful ways.
- Apply powerful machine learning to the data via SMILE and R integration.
- Visualize the data in over 50 distinct ways.
- Output the data to a variety of databases and file formats.
- Extend Dex from within via templates and internal scripting. Save the extensions to reuse later.
Data exploration - Weka

More info: https://www.youtube.com/watch?v=7quZv6WCTQc
Visualization

- Open-Source plotting libraries: GRAL, JFreeChart, Xchart, JMathPlot, Jzy3d, JavaFX, GroovyFX
- Types: Scatter plot, Line plot, Area plot, Pie plot, Donut plot, Horizontal bars, Vertical bars, Bubble plot, Radar plot, Box plot, Raster plot, Contour plot, Gauge plot, Step plot, Gantt plot, Histogram
- Features: Multiple axes, Logarithmic axes, Line smoothing, Data filtering/aggregation
Notebooks supporting Groovy

- Jupyter/beakerx
- Apache Zeppelin
- GroovyLab
- Seco
Notebooks supporting Groovy

- Jupyter/beakerx
- Apache Zeppelin
- GroovyLab
- Seco

What is Apache Zeppelin?

Multi-purpose notebook which supports
20+ language backends

- Data Ingestion
- Data Discovery
- Data Analytics
- Data Visualization & Collaboration

Diagram showing various components of Apache Zeppelin, including Groovy compiler, GroovySci domain specific language, Java compiler, Groovy shell, integrated Java libraries for graphics and numerical analysis, toolboxes of class code, application specific wizards, import wizard, and computer algebra subsystem.
Data preparation

IO:
• Java in-built
• Groovy enhancements
• Libraries
  • Apache Commons IO https://commons.apache.org/io/
  • AOL Cyclops-React https://github.com/aol/cyclops-react

Web scraping:
• GEB

Batch processing/ETL:
• Spring Batch http://projects.spring.io/spring-batch/
• Apache Camel
• Apache Gobblin
Math libraries

Math and stats libraries

• Java: log, exp, other basic methods.

Libraries:

• Apache Commons Math http://commons.apache.org/math/
  statistics, optimization, and linear algebra

• Apache Mahout http://mahout.apache.org/
  linear algebra, plus distributed linear algebra and machine learning

• JBlas http://jblas.org/ optimized and very fast linear algebra package
  that uses the BLAS library

Also, many of the machine learning libraries come with some math
functionality, often linear algebra, stats, and optimization.
Data storage

Database:
• Java’s JDBC
• Groovy JDBC enhancements and Datasets
• Other NOSQL
  • Apache Cassandra
  • MongoDB
  • Apache CouchDB
  • Neo4j

```java
import com.gmongo.GMongo

def db = new GMongo().getDB('athletes')
db.athletes.drop()
db.athletes << [
    first: 'Paul',
    last: 'Tergat',
    dob: '1969-06-17',
    runs: [[
        distance: 42195,
        time: 2 * 60 * 60 + 4 * 60 + 55,
        venue: 'Berlin',
        when: '2003-09-28'
    ]]]

println "World records following $marathon3.venue $marathon3.when:"
def t = new Traversal()
for (Path p in t.description().breadthFirst().relationships(MyRelationshipTypes.supercedes).
    evaluator(Evaluators.fromDepth(1)).
    uniqueness(Uniqueness.NONE).
    traverse(marathon3)) {
    def newRecord = p.endNode()
    println "$newRecord.venue $newRecord.when"
}

Graph g = new Neo4jGraph(graphDb)
def pretty = { it.collect { "$it.venue $it.when" }.join(', ', ) }
def results = []
g.V('venue', 'London').fill(results)
println 'London world records: ' + pretty(results)
results = []
g.V('venue', 'London').in('supercedes').fill(results)
println 'World records after London: ' + pretty(results)
```
Machine learning and data mining

Machine learning and data mining libraries:

• Weka http://www.cs.waikato.ac.nz/ml/weka/
• JavaML http://java-ml.sourceforge.net/
  old and reliable ML library but not actively updated
• Smile http://haifengl.github.io/smile/
  ML library under active development
• JSAT https://github.com/EdwardRaff/JSAT
  contains many machine learning algorithms
• H2O http://www.h2o.ai/
  distributed Java ML framework also supports Scala, R and Python
• Apache Mahout http://mahout.apache.org/
  distributed linear algebra framework
• Apache Spark ml and mllib
  https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples
Neural networks and text processing

Neural networks:
• Encog http://www.heatonresearch.com/encog/
• DeepLearning4j http://deeplearning4j.org/
  natively supports Keras models

Text processing
• Java: StringTokenizer, the java.text package, regular expressions
• Groovy: Regex enhancements, templates, String enhancements
• Libraries:
  • Apache Lucene https://lucene.apache.org/
  • Stanford CoreNLP http://stanfordnlp.github.io/CoreNLP/
  • Apache OpenNLP https://opennlp.apache.org/
  • LingPipe http://alias-i.com/lingpipe/
  • GATE https://gate.ac.uk/
  • MALLET http://mallet.cs.umass.edu/
  • Smile http://haifengl.github.io/smile/
Scaling up

Microservice frameworks:
• Micronaut

Distributed data processing:
• Apache Hadoop
• Apache Spark
• Apache Flink
• Apache Samza
• Apache Kafka
• Apache Storm
• Apache Apex
• Apache Beam
• Apache Ignite
• Apache Nifi
Data Science Algorithms

Deep Learning:
- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)
- Convolutional Neural Network (CNN)
- Stacked Auto-Encoders

Ensemble:
- Random Forest
- Gradient Boosting Machines (GBM)
- Boosting
- Bootstrapped Aggregation (Bagging)
- AdaBoost
- Stacked Generalization (Blending)
- Gradient Boosted Regression Trees (GBRT)

Neural Networks:
- Radial Basis Function Network (RBFN)
- Perceptron
- Back-Propagation
- Hopfield Network
- Least Absolute Shrinkage and Selection Operator (LASSO)
- Elastic Net
- Least Angle Regression (LARS)

Regularization:
- Cubist
- One Rule (Onet)
- Zero Rule (Zerot)
- Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

Regression:
- Linear Regression
- Ordinary Least Squares Regression (OLS)
- Stepwise Regression
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)
- Logistic Regression

Decision Tree:
- Classification and Regression Tree (CART)
- Iterative Dichotomiser 3 (ID3)
- C4.5
- C5.0
- Chi-squared Automatic Interaction Detection (CHAID)
- Decision Stump
- Conditional Decision Trees
- MS

Bayesian:
- Naive Bayes
- Averaged One-Dependence Estimators (ADDE)
- Bayesian Belief Network (BBN)
- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Bayesian Network (BN)

Dimensionality Reduction:
- Principal Component Analysis (PCA)
- Partial Least Squares Regression (PLSR)
- Sammon Mapping
- Multidimensional Scaling (MDS)
- Projection Pursuit
- Principal Component Regression (PCR)
- Partial Least Squares Discriminant Analysis
- Mixture Discriminant Analysis (MDA)
- Quadratic Discriminant Analysis (QDA)
- Regularized Discriminant Analysis (RDA)
- Flexible Discriminant Analysis (FDA)
- Linear Discriminant Analysis (LDA)

Instance Based:
- k-Nearest Neighbour (kNN)
- Learning Vector Quantization (LVQ)
- Self-Organizing Map (SOM)
- Locally Weighted Learning (LWL)

Clustering:
- k-Means
- k-Medians
- Expectation Maximization
- Hierarchical Clustering

House price predictions

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.

It's a great dataset for evaluating simple regression models.

Columns

- # id  a notation for a house
- date Date house was sold
- # price  Price is prediction target
- # bedrooms  Number of Bedrooms/House
- # bathrooms  Number of bathrooms/House
- # sqft_living  square footage of the home
- # sqft_lot  square footage of the lot
- # floors  Total floors (levels) in house
- # waterfront  House which has a view to a waterfront
- # view  Has been viewed
- # condition  How good the condition is ( Overall )
- grade  overall grade given to the housing unit, based on King County grading system
- # sqft_above  square footage of house apart from basement
- # sqft_basement  square footage of the basement
- # yr_built  Built Year
- # yr_renovated  Year when house was renovated
- # zipcode  zip
- # lat  Latitude coordinate
- # long  Longitude coordinate
- # sqft_living15  Living room area in 2015(implies-- some renovations)
  This might or might not have affected the lotsize area
- # sqft_lot15  lotSize area in 2015(implies-- some renovations)
import org.apache.commons.math3.random.EmpiricalDistribution
import static groovyx.javafx.GroovyFX.start
import static org.apache.commons.csv.CSVFormat.RFC4180 as CSV

def file = 'kc_house_data.csv' as File
def csv = CSV.withFirstRecordAsHeader().parse(new FileReader(file))
def all = csv.collect { it.bedrooms.toInteger() }
def dist = new EmpiricalDistribution(all.max()).tap{ load(all as double[])}
def bins = dist.binStats.withIndex().collectMany { v, i -> [i.toString(), v.n] }

start {
  stage(title: 'Number of bedrooms histogram', show: true, width: 800, height: 600) {
    scene {
      barChart(title: 'Bedroom count', barGap: 0, categoryGap: 2) {
        series(name: 'Number of properties', data: bins)
      }
    }
  }
}
House price predictions – view data graphically
import static org.apache.commons.csv.CSVFormat.RFC4180 as CSV

def file = 'kc_house_data.csv' as File
def csv = CSV.withFirstRecordAsHeader().parse(new FileReader(file))
def all = csv.collect { it.bedrooms.toInteger() }

def stats = new SummaryStatistics()
all.each{ stats.addValue(it as double) }
println stats.summary

n: 21613
min: 0.0
max: 33.0
mean: 3.370841623097218
std dev: 0.930061831147451
variance: 0.8650150097573497
sum: 72854.0
House price predictions – investigate outliers

```java
import org.apache.commons.csv.CSVFormat.RFC4180 as CSV

def file = 'kc_house_data.csv' as File
def csv = CSV.withFirstRecordAsHeader().parse(new FileReader(file))
csv.findAll{ it.bedrooms.toInteger() > 10 }.each{ println it.toMap() as TreeMap }
```

```java
```
import com.opencsv.bean.*
import groovy.transform.ToString

@ToString(includeNames = true)
class House {
    @CsvBindByName
    Integer bedrooms

    @CsvBindByName
    String bathrooms

    @CsvBindByName(column = 'sqft_lot')
    Integer area_lot
}

def file = 'kc_house_data.csv' as File
def builder = new CsvToBeanBuilder(new FileReader(file)).build().parse()
def records = builder.withType(House).build().parse()

records.findAll{ it.bedrooms > 10 }.each{ println it }
```java
import org.apache.commons.math3.random.EmpiricalDistribution
import static groovyx.javafx.GroovyFX.start
import static org.apache.commons.csv.CSVFormat.RFC4180 as CSV

def file = 'kc_house_data.csv' as File
def csv  = CSV.withFirstRecordAsHeader().parse(new FileReader(file))
def all  = csv.collect { it.bedrooms.toInteger() }.findAll{ it < 30 }
def dist = new EmpiricalDistribution(all.max()).tap{ load(all as double[]) }
def bins = dist.binStats.withIndex().collectMany { v, i -> [i.toString(), v.n] }

start {
  stage(title: 'Number of bedrooms histogram', show: true, width: 800, height: 600) {
    scene {
      barChart(title: 'Bedroom count', barGap: 0, categoryGap: 2) {
        series(name: 'Number of properties', data: bins)
      }
    }
  }
}
```
House price predictions – remove outlier and view
import org.apache.commons.math3.random.EmpiricalDistribution
import static groovyx.javafx.GroovyFX.start
import static org.apache.commons.csv.CSVFormat.RFC4180 as CSV

def file = 'kc_house_data.csv' as File
def csv = CSV.withFirstRecordAsHeader().parse(new FileReader(file))
def all = csv.findAll { it.bedrooms.toInteger() < 30 }.collect { it.price.toDouble() }
def info = new SummaryStatistics(); all.each(info::addValue)
def head = "Price percentile (min=${info.min}, mean=${info.mean as int}, max=${info.max})"
def dist = new EmpiricalDistribution(100).tap{ load(all as double[]) }
def bins = dist.binStats.withIndex().collectMany { v, i -> [i.toString(), v.n] }

start {
    stage(title: 'Price histogram', show: true, width: 800, height: 600) {
        scene {
            barChart(title: head, barGap: 0, categoryGap: 0) {
                series(name: 'Number of properties', data: bins)
            }
        }
    }
}
House price predictions – remove outlier and view

Price histogram

Price percentile (min=$75000.0, mean=$540083, max=$7700000.0)
```java
import org.apache.commons.math3.stat.regression.SimpleRegression
import static groovy.javafx.GroovyFX.start
import static org.apache.commons.csv.CSVFormat.RFC4180 as CSV

def file = 'kc_house_data.csv' as File
def csv = CSV.withFirstRecordAsHeader().parse(new FileReader(file))
def all = csv.collect { [it.bedrooms.toDouble(), it.price.toDouble()] }.findAll{ it[0] < 30 }

def reg = new SimpleRegression().tap{ addData(all as double[][]) }
def (minB, maxB) = [all.min{ it[0] }][0], all.max{ it[0] }[0]
def predicted = [[minB, reg.predict(minB)], [maxB, reg.predict(maxB)]]

start {
    stage(title: 'Price vs Number of bedrooms', show: true, width: 800, height: 600) {
        scene {
            scatterChart {
                series(name: 'Actual', data: all)
            }
            lineChart {
                series(name: 'Predicted', data: predicted)
            }
          }
    }
}
```
House price predictions – linear regression
House price predictions - Weka
House price predictions
House price predictions
House price predictions
House price predictions
```python
def full = getClass().classLoader.getResource('kc_house_data.csv').file as File
def parent = full.parentFile
def lines = full.readLine()

# Split dataset

def (trainLines, testLines) = lines.chop(lines.size() * 0.8 as int, -1)

def train = new File(parent, 'house_train.csv')
train.text = trainLines.join('\n')

def test = new File(parent, 'house_test.csv')
test.delete()
test << lines[0] << '\n' << testLines.join('\n')
```
Scaling up

House price predictions – linear regression

```
Ignite ignite = Ignition.start("/path/to/example-ignite.xml")
ignite.withCloseable {
  println ">>> Ignite grid started."
  def dataCache = new SandboxMLCache(ignite).fillCacheWith(BOSTON_HOUSE_PRICES)
  def trainer = new LinearRegressionLSQRTrainer()
    .withEnvironmentBuilder(defaultBuilder().withRNGSeed(0))
  def vectorizer = new DummyVectorizer().labeled(Vectorizer.LabelCoordinate.FIRST)
  def split = new TrainTestDatasetSplitter().split(0.8)
  def mdl = trainer.fit(ignite, dataCache, split.trainFilter, vectorizer)
  def metric = new RegressionMetrics().withMetric{ it.r2() }
  def score = Evaluator.evaluate(dataCache, split.testFilter, mdl, vectorizer, metric)
  println ">>> Model: " + toString(mdl)
  println ">>> R^2 score: " + score
  dataCache.destroy()
}
```

```
static toString(mdl) {
  def sign = { val -> val < 0 ? '- ' : ' + ' }
  def valIdx = { idx, val -> sprintf '%.2f*f%d', val, idx }
  def valIdxSign = { idx, val -> sign(val) + valIdx(idx, Math.abs(val)) }
  def valSign = { val -> sign(val) + sprintf('% .2f', Math.abs(val)) }
  def w = mdl.weights
  def i = mdl.intercept
  def result = [valIdx(0, w.get(0)), *(1..<w.size()).collect{ valIdxSign(it, w.get(it)) }, valSign(i)]
  result.join(' ')
```

[09:17:59] Ignite node started OK (id=0b9592e1)
[09:17:59] Topology snapshot [ver=1, locNode=0b9592e1, servers=1, clients=0, state=ACTIVE, CPUs=12, offheap=6.4GB, heap=7.1GB]

```
>>> Ignite grid started.
>>> Model: 0.04*f0 - 0.05*f1 - 0.79*f2 - 10.01*f3 + 0.28*f4 + 0.01*f5 - 0.89*f6 + 0.54*f7 - 0.00*f8 - 0.18*f9 - 0.01*f10 + 0.09*f11 - 0.15*f12 + 14.10
```

[09:17:59] R^2 score: 0.412878309612252

[09:17:59] Ignite node stopped OK [uptime=00:00:00.626]
Data Science Algorithms

Deep Learning
- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)
- Convolutional Neural Network (CNN)
- Stacked Auto-Encoders
- Random Forest
- Gradient Boosting Machines (GBM)
- Boosting
- Bootstrapped Aggregation (Bagging)
- AdaBoost
- Stacked Generalization (Blending)
- Gradient Boosted Regression Trees (GBRT)
- Radial Basis Function Network (RBFN)

Bayesian
- Naive Bayes
- Averaged One-Dependence Estimators (ACDE)
- Bayesian Belief Network (BBN)
- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Bayesian Network (BN)
- Classification and Regression Tree (CART)
- Iterative Dichotomiser 3 (ID3)
- C4.5
- C5.0
- Chi-squared Automatic Interaction Detection (CHAID)
- Decision Stump
- Conditional Decision Trees
- M5

Decision Tree
- Principal Component Analysis (PCA)
- Partial Least Squares Regression (PLSR)
- Sammon Mapping
- Multidimensional Scaling (MDS)
- Projection Pursuit
- Principal Component Regression (PCR)
- Partial Least Squares Discriminant Analysis
- Mixture Discriminant Analysis (MDA)
- Quadratic Discriminant Analysis (QDA)
- Regularized Discriminant Analysis (RDA)
- Flexible Discriminant Analysis (FDA)
- Linear Discriminant Analysis (LDA)

Neural Networks
- Perceptron
- Back-Propagation
- Hopfield Network
- Least Absolute Shrinkage and Selection Operator (LASSO)
- Elastic Net
- Least Angle Regression (LARS)
- Cubist
- One Rule (OneR)
- Zero Rule (ZeroR)
- Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
- Linear Regression
- Ordinary Least Squares Regression (OLS)
- Stepwise Regression
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)
- Logistic Regression

Regularization
- Ridge Regression
- Regularized Regression

Dimensionality Reduction
- Principal Component Analysis (PCA)
- Partial Least Squares Regression (PLSR)
- Sammon Mapping
- Multidimensional Scaling (MDS)
- Projection Pursuit
- Principal Component Regression (PCR)
- Partial Least Squares Discriminant Analysis
- Mixture Discriminant Analysis (MDA)
- Quadratic Discriminant Analysis (QDA)
- Regularized Discriminant Analysis (RDA)
- Flexible Discriminant Analysis (FDA)
- Linear Discriminant Analysis (LDA)

Regression
- Least Absolute Shrinkage and Selection Operator (LASSO)
- Elastic Net
- Least Angle Regression (LARS)
- Cubist
- One Rule (OneR)
- Zero Rule (ZeroR)
- Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
- Linear Regression
- Ordinary Least Squares Regression (OLS)
- Stepwise Regression
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)
- Logistic Regression

Instance Based
- k-Nearest Neighbour (kNN)
- Learning Vector Quantization (LVQ)
- Self-Organizing Map (SOM)
- Locally Weighted Learning (LWL)

Clustering
- k-Means
- k-Medians
- Expectation Maximization
- Hierarchical Clustering

Data Science Algorithms: Classification

https://commons.apache.org/proper/commons-math/userguide/ml.html
Computational problems in Computer Science

- Kinds: decision problems, search problems, counting problems, optimization problems
- Paradigms: brute force, divide & conquer, search & enumerate, randomized, complexity reduction, recursive, back tracking, graph
- Optimization approaches: linear programming, dynamic programming, greedy, heuristics
Constraint Programming

Where is it used?

• Factory/assembly line scheduling
• Workforce management/scheduling (call centres, aircraft crew, etc)
• Vehicle routing/traffic planning/traffic management
• Packing problems
• Timetabling (exams, lectures, trains, gate arrival at airports)
• Configuration and design
• Supertree construction (bioinformatics)
• Network design (telecommunications)
• Military logistics (air cover for naval fleet)
• Aircraft maintenance schedules
• Recruitment (personnel selection)
• Financial trading systems
• Language detection
• Operations research
**Constraint Programming**

Typical domains:

- boolean domains, where only true/false constraints apply (SAT problem)
- integer domains
- rational domains
- interval domains, in particular for scheduling problems
- linear domains, where only linear functions are described and analyzed (although approaches to non-linear problems do exist)
- finite domains, where constraints are defined over finite sets
- mixed domains, involving two or more of the above
Cryptarithmetic

\[
\begin{array}{c}
S \\
E \\
N \\
D \\
\hline
+ \\
M \\
O \\
R \\
E \\
\hline
= \\
M \\
O \\
N \\
E \\
Y
\end{array}
\]

Replace the letters with decimal digits to make a valid arithmetic sum.
def solutions():
    # letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
    all_solutions = list()
    for s in range(1, 10):
        for e in range(0, 10):
            for n in range(0, 10):
                for d in range(0, 10):
                    for m in range(1, 10):
                        for o in range(0, 10):
                            for r in range(0, 10):
                                for y in range(0, 10):
                                    if len({s, e, n, d, m, o, r, y}) == 8:
                                        send = 1000 * s + 100 * e + 10 * n + d
                                        more = 1000 * m + 100 * o + 10 * r + e
                                        money = 10000 * m + 1000 * o + 100 * n + 10 * e + y
                                        if send + more == money:
                                            all_solutions.append((send, more, money))
    return all_solutions

print(solutions())

[[9567, 1085, 10652]]
Cryptarithmetic: Brute force

```python
def solutions() {
    // letters = ['s', 'e', 'n', 'd', 'm', 'o', 'r', 'y']
def all_solutions = []
    for (s in 1..<10)
        for (e in 0..9)
            for (n in 0..9)
                for (d in 0..9)
                    for (m in 1..9)
                        for (o in 0..9)
                            for (r in 0..9)
                                for (y in 0..9)
                                    if ([s, e, n, d, m, o, r, y].toSet().size() == 8) {
                                        def send = 1000 * s + 100 * e + 10 * n + d
                                        def more = 1000 * m + 100 * o + 10 * r + e
                                        def money = 10000 * m + 1000 * o + 100 * n + 10 * e + y
                                        if (send + more == money)
                                            all_solutions.add([send, more, money])
                                    }
    return all_solutions
}

print(solutions())
```

```
[[9567, 1085, 10652]]
```
Cryptarithmetic: Brute force

```python
from itertools import permutations

def solution2():
    letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
    digits = range(10)
    for perm in permutations(digits, len(letters)):
        sol = dict(zip(letters, perm))
        if sol['s'] == 0 or sol['m'] == 0:
            continue
        send = 1000 * sol['s'] + 100 * sol['e'] + 10 * sol['n'] + sol['d']
        more = 1000 * sol['m'] + 100 * sol['o'] + 10 * sol['r'] + sol['e']
        money = 10000 * sol['m'] + 1000 * sol['o'] + 100 * sol['n'] + 10 * sol['e'] + sol['y']
        if send + more == money:
            return send, more, money

print(solution2())
```

\[
\begin{align*}
\text{S} & \quad \text{E} \quad \text{N} \quad \text{D} \\
+ & \quad \text{M} \quad \text{O} \quad \text{R} \quad \text{E} \\
= & \quad \text{M} \quad \text{O} \quad \text{N} \quad \text{E} \quad \text{Y}
\end{align*}
\]

\[(9567, 1085, 10652)\]
def solution2() {
    def digits = 0..9
    for (p in digits.permutations()) {
        if (p[-1] < p[-2]) continue
        def (s, e, n, d, m, o, r, y) = p
        if (s == 0 || m == 0) continue
        def send = 1000 * s + 100 * e + 10 * n + d
        def more = 1000 * m + 100 * o + 10 * r + e
        def money = 10000 * m + 1000 * o + 100 * n + 10 * e + y
        if (send + more == money)
            return [send, more, money]
    }
}

print(solution2())

[9567, 1085, 10652]
from csp import Constraint, CSP
from typing import Dict, List, Optional

class SendMoreMoneyConstraint(Constraint[str, int]):
    def __init__(self, letters: List[str]) -> None:
        super().__init__(letters)
        self.letters: List[str] = letters

    def satisfied(self, assignment: Dict[str, int]) -> bool:
        # not a solution if duplicate values
        if len(set(assignment.values())) < len(assignment):
            return False
        # if all vars assigned, check if correct
        if len(assignment) == len(self.letters):
            s: int = assignment['S']
            e: int = assignment['E']
            n: int = assignment['N']
            d: int = assignment['D']
            m: int = assignment['M']
            o: int = assignment['O']
            r: int = assignment['R']
            y: int = assignment['Y']
            send: int = s * 1000 + e * 100 + n * 10 + d
            more: int = m * 1000 + o * 100 + r * 10 + e
            money: int = m * 10000 + o * 1000 + n * 100 + e * 10 + y
            return send + more == money
        return True

letters = ['S', 'E', 'N', 'D', 'M', 'O', 'R', 'Y']
possible_digits = {}
for letter in letters:
    possible_digits[letter] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
possible_digits['M'] = [1]  # can't start with 0 constraint
csp: CSP[str, int] = CSP(letters, possible_digits)
csp.add_constraint(SendMoreMoneyConstraint(letters))
solution: Optional[Dict[str, int]] = csp.backtracking_search()
if solution is None:
    print("No solution found!")
else:
    print(solution)

{'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0, 'R': 8, 'Y': 2}

Adapted from: https://freecontent.manning.com/constraint-satisfaction-problems-in-python/
Cryptarithmetic: Constraint programming

```python
@Grab('org.choco-solver:choco-solver:4.10.0')
import org.chocosolver.solver.Model
import org.chocosolver.solver.variables.IntVar

def model = new Model("SEND+MORE=MONEY")
def S = model.intVar("S", 1, 9)
def E = model.intVar("E", 0, 9)
def N = model.intVar("N", 0, 9)
def D = model.intVar("D", 0, 9)
def M = model.intVar("M", 1, 9)
def O = model.intVar("O", 0, 9)
def R = model.intVar("R", 0, 9)
def Y = model.intVar("Y", 0, 9)

model.allDifferent(S, E, N, D, M, O, R, Y).post()
```

Solution: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2,
## Dietary restrictions

Minimise cost of diet given:
- Must be at least 300 calories
- Not more than 10 grams of protein
- Not less than 10 grams of carbohydrates
- Not less than 8 grams of fat
- At least 0.5 units of fish
- No more than 1 unit of milk

<table>
<thead>
<tr>
<th></th>
<th>Bread</th>
<th>Milk</th>
<th>Cheese</th>
<th>Potato</th>
<th>Fish</th>
<th>Yogurt</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td>2.0</td>
<td>3.5</td>
<td>8.0</td>
<td>1.5</td>
<td>11.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Protein (g)</strong></td>
<td>4.0</td>
<td>8.0</td>
<td>7.0</td>
<td>1.3</td>
<td>8.0</td>
<td>9.2</td>
</tr>
<tr>
<td><strong>Fat (g)</strong></td>
<td>1.0</td>
<td>5.0</td>
<td>9.0</td>
<td>0.1</td>
<td>7.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Carbohydrates (g)</strong></td>
<td>15.0</td>
<td>11.7</td>
<td>0.4</td>
<td>22.6</td>
<td>0.0</td>
<td>17.0</td>
</tr>
<tr>
<td><strong>Calories</strong></td>
<td>90</td>
<td>120</td>
<td>106</td>
<td>97</td>
<td>130</td>
<td>180</td>
</tr>
</tbody>
</table>

[Learn more about linear programming](http://support.sas.com/documentation/cdl/en/ormpug/63352/HTML/default/viewer.htm#ormpug_lpsolver_sect018.htm)
<table>
<thead>
<tr>
<th></th>
<th>Bread</th>
<th>Milk</th>
<th>Cheese</th>
<th>Potato</th>
<th>Fish</th>
<th>Yogurt</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td>2</td>
<td>3.5</td>
<td>8</td>
<td>1.5</td>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Protein</strong></td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>1.3</td>
<td>8</td>
<td>9.2</td>
<td>0</td>
</tr>
<tr>
<td><strong>Fat</strong></td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>0.1</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Carbs</strong></td>
<td>15</td>
<td>11.7</td>
<td>0.4</td>
<td>22.6</td>
<td>0</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td><strong>Calories</strong></td>
<td>90</td>
<td>120</td>
<td>105</td>
<td>97</td>
<td>130</td>
<td>180</td>
<td>0</td>
</tr>
</tbody>
</table>
Dietary restrictions (Excel)

<table>
<thead>
<tr>
<th></th>
<th>Bread</th>
<th>Milk</th>
<th>Cheese</th>
<th>Potato</th>
<th>Fish</th>
<th>Yogurt</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>2</td>
<td>3.5</td>
<td>8</td>
<td>1.5</td>
<td>11</td>
<td>1</td>
<td>12.08134</td>
</tr>
<tr>
<td>Protein</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>1.3</td>
<td>8</td>
<td>9.2</td>
<td>10</td>
</tr>
<tr>
<td>Fat</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>0.1</td>
<td>7</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Carbs</td>
<td>15</td>
<td>11.7</td>
<td>0.4</td>
<td>22.6</td>
<td>0</td>
<td>17</td>
<td>42.9597</td>
</tr>
<tr>
<td>Calories</td>
<td>90</td>
<td>120</td>
<td>106</td>
<td>97</td>
<td>130</td>
<td>180</td>
<td>300</td>
</tr>
</tbody>
</table>

Solver Parameters:
- Set Objective: $O14
- To: Max
- By Changing Variable Cells: $B$6:$E$13
- Subject to the Constraints:
  - $F9 \leq 1$
  - $F6 \leq 0.5$
  - $O5 \leq 10$
  - $O6 \geq 8$
  - $O7 \geq 10$
  - $O8 \geq 300$

Make Unconstrained Variables Non-Negative
- Select a Solving Method: GRG Nonlinear
Dietary restrictions (Google sheets)

<table>
<thead>
<tr>
<th></th>
<th>Bread</th>
<th>Milk</th>
<th>Cheese</th>
<th>Potato</th>
<th>Fish</th>
<th>Yogurt</th>
<th>Total</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>2</td>
<td>3.5</td>
<td>8</td>
<td>1.5</td>
<td>11</td>
<td>1</td>
<td>12.08133788</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Protein</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>1.3</td>
<td>8</td>
<td>9.2</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Fat</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>0.1</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Carbs</td>
<td>15</td>
<td>11.7</td>
<td>0.4</td>
<td>22.6</td>
<td>0</td>
<td>17</td>
<td>42.959069651</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Calories</td>
<td>90</td>
<td>120</td>
<td>106</td>
<td>97</td>
<td>130</td>
<td>180</td>
<td>300</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>

Min: 0.5
Max: 1
Diet problem (ojalgo)

```python
def model = new ExpressionsBasedModel()

def bread = model.addVariable("Bread").lower(0)
def milk = model.addVariable("Milk").lower(0).upper(1)
def cheese = model.addVariable("Cheese").lower(0)
def potato = model.addVariable("Potato").lower(0)
def fish = model.addVariable("Fish").lower(0)
def yogurt = model.addVariable("Yogurt").lower(0)

def cost = model.addExpression("Cost")
cost.set(bread, 2.0).set(milk, 3.5).set(cheese, 8.0).set(potato, 1.5).set(fish, 11.0).set(yogurt, 1.0)

def protein = model.addExpression("Protein").upper(10)
protein.set(bread, 4.0).set(milk, 8.0).set(cheese, 7.0).set(potato, 1.3).set(fish, 8.0).set(yogurt, 9.2)

def fat = model.addExpression("Fat").lower(8)
fat.set(bread, 1.0).set(milk, 5.0).set(cheese, 9.0).set(potato, 0.1).set(fish, 7.0).set(yogurt, 1.0)

def carbs = model.addExpression("Carbohydrates").lower(10)
carbs.set(bread, 15.0).set(milk, 11.7).set(cheese, 0.4).set(potato, 22.6).set(fish, 0.0).set(yogurt, 17.0)

def calories = model.addExpression("Calories").lower(300)

def result = model.minimise()
```

// for a variation, see:
// https://www.ojalgo.org/2019/05/the-diet-problem/

OPTIMAL 0.0 @ { 0.0, 0.0, 0.40813898143741034, 1.8538791051880057, 0.5916230366492151, 0.0 }

0 <= Bread: 0
0 <= Milk: 0 <= 1
0 <= Cheese: 0.408139
0 <= Potato: 1.853879
0.5 <= Fish: 0.591623
0 <= Yogurt: 0
10 <= Carbohydrates: 42.060923
8 <= Fat: 8.0
Cost: 12.553784
Protein: 10.0 <= 10
300 <= Calories: 300.0

OPTIMAL 0.0 @ { 0.0, 0.0, 0.40813898143741034, 1.8538791051880057, 0.5916230366492151, 0.0 }

0 <= Bread: 0
0 <= Milk: 0 <= 1
0 <= Cheese: 0.408139
0 <= Potato: 1.853879
0.5 <= Fish: 0.591623
0 <= Yogurt: 0
10 <= Carbohydrates: 42.060923
8 <= Fat: 8.0
Cost: 12.553784
Protein: 10.0 <= 10
300 <= Calories: 300.0
```python
def model = new Model("Diet problem")
def unbounded = 100000

// scale quantities by 10, coefficients by 10, products by 100
def bread = model.intVar("Bread", 0, unbounded)
def milk = model.intVar("Milk", 0, 10)
def cheese = model.intVar("Cheese", 0, unbounded)
def potato = model.intVar("Potato", 0, unbounded)
def fish = model.intVar("Fish", 5, unbounded)
def yogurt = model.intVar("Yogurt", 0, unbounded)
IntVar[] all = [bread, milk, cheese, potato, fish, yogurt]

def cost = model.intVar("Cost", 0, unbounded)
model.scalar(all, [20, 35, 80, 15, 110, 10] as int[], ",=", cost).post()

def protein = model.intVar("Protein", 0, 1000)
model.scalar(all, [40, 80, 70, 13, 80, 92] as int[], ",=", protein).post()

def fat = model.intVar("Fat", 800, unbounded)
model.scalar(all, [10, 50, 90, 1, 70, 10] as int[], ",=", fat).post()

def carbs = model.intVar("Carbohydrates", 1000, unbounded)
model.scalar(all, [150, 117, 4, 226, 0, 170] as int[], ",=", carbs).post()

def calories = model.intVar("Calories", 30000, unbounded)
model.scalar(all, [900, 1200, 1060, 970, 1300, 1800] as int[], ",=", calories).post()
model.setObjective(Model.MINIMIZE, cost)

def found = model.solver.findSolution()
if (found) {
    all.each { println "$it.name: ${it.value / 10}" }
    [carbs, fat, protein, calories, cost].each { println "$it.name: ${it.value / 100}" }
} else {
    println "No solution"
}
```

Choco supports `RealVar` but doesn't have as rich a set of possible constraints for such vars.
Diet problem (Choco)

```python
def model = new Model("Diet problem")
def unbounded = 1000.0d
def precision = 0.00001d

// scale quantities by 10, coefficients by 10, products by 100
def bread = model.realVar("Bread", 0.0, unbounded, precision)
def milk = model.realVar("Milk", 0.0, 1.0, precision)
def cheese = model.realVar("Cheese", 0.0, unbounded, precision)
def potato = model.realVar("Potato", 0.0, unbounded, precision)
def fish = model.realVar("Fish", 0.5, unbounded, precision)
def yogurt = model.realVar("Yogurt", 0.0, unbounded, precision)
RealVar[] all = [bread, milk, cheese, potato, fish, yogurt]

def scalarIbex = { coeffs, var ->
    def (a, b, c, d, e, f) = coeffs
    model.realIbexGenericConstraint("a*{0}+b*{1}+c*{2}+d*{3}+e*{4}+f*{5}={6}",
        ["a", var] as RealVar[]).post();
}

def cost = model.realVar("Cost", 0.0, unbounded, precision)
scalarIbex([2.0, 3.5, 8.0, 1.5, 11.0, 1.0], cost)
def protein = model.realVar("Protein", 0.0, 10.0, precision)
scalarIbex([4.0, 8.0, 7.0, 1.3, 8.0, 9.2], protein)
def fat = model.realVar("Fat", 8.0, unbounded, precision)
scalarIbex([1.0, 5.0, 9.0, 0.1, 7.0, 1.0], fat)
def carbs = model.realVar("Carbohydrates", 10.0, unbounded, precision)
scalarIbex([15.0, 11.7, 0.4, 22.6, 0.0, 17.0], carbs)
def calories = model.realVar("Calories", 300, unbounded, precision)
scalarIbex([90, 120, 106, 97, 130, 180], calories)
model.setObjective(Model.MINIMIZE, cost)

def found = model.solver.findSolution()
```

Choco does have a plugin (via JNI) for the Ibex C++ constraint processing library which does handle real numbers.
import org.apache.commons.math3.optim.linear.*
import org.apache.commons.math3.optim.nonlinear.scalar.GoalType
import static org.apache.commons.math3.optim.linear.Relationship.*

def cost = new LinearObjectiveFunction([2.0, 3.5, 8.0, 1.5, 11.0, 1.0] as double[], 0)

static scalar(coeffs, rel, val) { new LinearConstraint(coeffs as double[], rel, val) }

def bread_min = scalar([1, 0, 0, 0, 0, 0], GEQ, 0)
def milk_min = scalar([0, 1, 0, 0, 0, 0], GEQ, 0)
def milk_max = scalar([0, 1, 0, 0, 0, 0], LEQ, 1)
def cheese_min = scalar([0, 0, 1, 0, 0, 0], GEQ, 0)
def potato_min = scalar([0, 0, 0, 1, 0, 0], GEQ, 0)
def fish_min = scalar([0, 0, 0, 1, 0, 0], GEQ, 0.5)
def yogurt_min = scalar([0, 0, 0, 0, 0, 1], GEQ, 0)
def protein = scalar([4.0, 8.0, 7.0, 1.3, 8.0, 9.2], LEQ, 10)
def fat = scalar([1.0, 5.0, 9.0, 0.1, 7.0, 1.0], GEQ, 8)
def carbs = scalar([15.0, 11.7, 0.4, 22.6, 0.0, 17.0], GEQ, 10)
def calories = scalar([90, 120, 106, 97, 130, 180], GEQ, 300)

LinearConstraintSet constraints = [bread_min, milk_min, milk_max, fish_min, cheese_min, potato_min, yogurt_min, protein, fat, carbs, calories]

def solution = new SimplexSolver().optimize(cost, constraints, GoalType.MAXIMIZE)

if (solution != null) {
    println "Opt: $solution.value"
    println solution.point.collect{ printf ' %.2f', it }.join(', ')
}

Opt: 12.553783912422674
-0.00, -0.00, 0.41, 1.85, 0.59, 0.00
Apache Spark

Open Source Ecosystem

Applications

Spark SQL
Spark Streaming
MLlib
GraphX

Apache Spark

Programming Guides:
- Quick Start: a quick introduction to the Spark API; start here!
- RDD Programming Guide: overview of Spark basics - RDDs
- Spark SQL, Datasets, and DataFrames: processing structured
- Structured Streaming: processing structured data streams
- Spark Streaming: processing data streams using DStreams
- MLlib: applying machine learning algorithms
- GraphX: processing graphs

API Docs:
- Spark Scala API (Scaladoc)
- Spark Java API (Javadoc)
- Spark Python API (Sphinx)
- Spark R API (Roxygen2)
- Spark SQL, Built-In Functions (MkDocs)

import scala.Tuple2
import org.apache.spark.sql.SparkSession

def SPACE = ~" "
def spark = SparkSession.builder()
    .config("spark.master", "local")
    .appName("JavaWordCount")
    .getOrCreate()

def lines = spark.read().textFile('/path/to/peppers.txt').javaRDD()
def words = lines.flatMap(s -> SPACE.split(s).iterator())
def ones = words.mapToPair(s -> new Tuple2<>(s, 1))
def counts = ones.reduceByKey{ i1, i2 -> i1 + i2 }
def output = counts.collect()

for (tup in output) {
    println tup._1() + ": " + tup._2()
}

spark.stop()
Apache Beam Overview

Apache Beam is an open source, unified model for defining both batch and streaming data-parallel processing pipelines.

Apache Beam Pipeline Runners

The Beam Pipeline Runners translate the data processing pipeline you define with your Beam program into the API compatible with the distributed processing back-end of your choice. When you run your Beam program, you’ll need to specify an appropriate runner for the back-end where you want to execute your pipeline.

Beam currently supports Runners that work with the following distributed processing back-ends:

- Apache Apex
- Apache Flink
- Apache Gearpump (incubating)
- Apache Samza
- Apache Spark
- Google Cloud Dataflow

Note: You can always execute your pipeline locally for testing and debugging purposes.

Apache Beam SDKs

The Beam SDKs provide a unified programming model that can represent and transform data sets of any size, whether the input is a finite data set from a batch data source, or an infinite data set from a streaming data source. The Beam SDKs use the same classes to represent both bounded and unbounded data, and the same transforms to operate on that data. You use the Beam SDK of your choice to build a program that defines your data processing pipeline.

Beam currently supports the following language-specific SDKs:

- Java
- Python
- Go

A Scala interface is also available as Scio.
Apache Beam Overview

Apache Beam is an open source, unified model for defining both batch and streaming data-parallel processing pipelines.

Apache Beam Pipeline Runners

The Beam Pipeline Runners translate the data processing pipeline you define with your Beam program into the API compatible with the distributed processing back-end of your choice. When you run your Beam program, you'll need to specify an appropriate runner for the back-end where you want to execute your pipeline.

Beam currently supports Runners that work with the following distributed processing back-ends:

- Apache Apex
- Apache Flink
- Apache Gearpump (incubating)
- Apache Samza
- Apache Spark
- Google Cloud Dataflow

Note: You can always execute your pipeline locally for testing and debugging purposes.

Apache Beam SDKs

The Beam SDKs provide a unified programming model that can represent and transform data sets of any size, whether the input is a finite data set from a batch data source, or an infinite data set from a streaming data source. The Beam SDKs use the same classes to represent both bounded and unbounded data, and the same transforms to operate on that data. You use the Beam SDK of your choice to build a program that defines your data processing pipeline.

Beam currently supports the following language-specific SDKs:

- Java
- Python
- Go

A Scala interface is also available as Scio.
Apache Beam Java SDK Quickstart

This Quickstart will walk you through executing your first Beam pipeline to run WordCount, written using Beam's Java SDK, on a runner of your choice.

Get the WordCount Code

The easiest way to get a copy of the WordCount pipeline is to use the following command to generate a simple Maven project that contains Beam's WordCount examples and builds against the most recent Beam release:

```
PS> mvn archetype:generate
-D archetypeGroupId=org.apache.beam
-D archetypeArtifactId=beam-sdks-java-maven-archetypes-examples
-D archetypeVersion=2.13.0
-D groupId=org.example
-D artifactId=word-count-beam
-D version="0.1"
-D package=org.apache.beam.examples
-D interactiveMode=false
```

Run WordCount

For Unix shells:

```
$ mvn compile exec:java -Dexec.mainClass=org.apache.beam.examples.WordCount \
-Dexec.args="--inputFile=pom.xml --output=counts" -Pdirect-runner
```
Apache Beam Word Count – Kata

See: https://beam.apache.org/blog/2019/05/30/beam-kata-release.html

Word Count Pipeline

Kata: Create a pipeline that counts the number of words.

Please output the count of each word in the following format:

```java
word count
apple: 1
orange: 2
banana: 3
```

Hint:
Refer to your katas above.
import ...

static PCollection applyTransform(PCollection input) {
    ProcessFunction asWords = line -> line.split(" ").toList()

    def kv2out = new DoFn<KV, String>() {
        @ProcessElement
        void processElement(@Element KV element, OutputReceiver<String> out) {
            out.output(element.key + ":" + element.value)
        }
    }

    return input
        .apply(into(strings()).via(asWords))
        .apply(Count.perElement())
        .apply(ParDo.of(kv2out))
}

def lines = ["apple orange grape banana apple banana",
             "banana orange banana papaya"]

def pipeline = Pipeline.create()
def counts   = pipeline.apply(Create.of(lines))
def output   = applyTransform(counts)

output.apply(Log.ofElements())

pipeline.run()
Why MXNet?

Probably, if you’ve stumbled upon this page, you’ve heard of deep learning. Deep learning denotes the modern incarnation of neural networks and it’s the technology behind recent breakthroughs in self-driving cars, machine translation, speech recognition and more. While widespread interest in deep learning took off in 2012, deep learning has become an indispensable tool for countless industries.

Source: https://mxnet.apache.org_versions/master_faq_why_mxnet.html
import org.apache.mxnet.infer.javaapi.ObjectDetector
import org.apache.mxnet.javaapi.*

static void downloadUrl(String url, String filePath) {
    def destination = filePath as File
    if (!destination.exists()) {
        destination.bytes = new URL(url).bytes
    }
}

static downloadModelImage() {
    String tempDirPath = System.getProperty('java.io.tmpdir')
    println "tempDirPath: \$tempDirPath"
    def imagePath = tempDirPath + '/inputImages/resnetssd/dog-ssd.jpg'
    String imgURL = 'https://s3.amazonaws.com/model-server/inputs/dog-ssd.jpg'
    downloadUrl(imgURL, imagePath)
    def modelPath = tempDirPath + '/resnetssd/resnet50_ssd_model'
    println "Download model files, this can take a while..."
    String modelURL = 'https://s3.amazonaws.com/model-server/models/resnet50_ssd/'
    downloadUrl(modelURL + 'resnet50_ssd_model-symbol.json',
                tempDirPath + '/resnetssd/resnet50_ssd_model-symbol.json')
    downloadUrl(modelURL + 'resnet50_ssd_model-0000.params',
                tempDirPath + '/resnetssd/resnet50_ssd_model-0000.params')
    downloadUrl(modelURL + 'synset.txt',
                tempDirPath + '/resnetssd/synset.txt')
    [imagePath, modelPath]
}
import mxnet as mx

def downloadModelImage()
    # Download model and image
    pass

def main()
    modelPath = "model_path"
    imagePath = "image_path"
    width, height = [512, 512]

    inputShape = new Shape([1, 3, width, height])
    results = detectObjects(modelPath, imagePath, inputShape).sum()

    for r in results
        print "Class: $r.className" with probability: %.3f"n
        printf " Coord: %r.XMin * width, %r.XMax * height, %r.YMin * width, %r.YMax * height"
        println " Coord: $coordinates.join(, )"

    def detectObjects(modelPath, imagePath, inputShape)
        def context = [Context.cpu()]
        def inputDescriptor = new DataDesc("data", inputShape, DType.Float32(), "NCHW")
        def objDet = new ObjectDetector(modelPath, [inputDescriptor], context, 0)
        return objDet.imageObjectDetect(ObjectDetector.loadImageFromFile(imagePath), 3)
Groovy Community Information

- groovy.apache.org
- groovy-lang.org
- github.com/apache/groovy
- groovycommunity.com
- dev@groovy.apache.org
- users@groovy.apache.org
- @ApacheGroovy
- objectcomputing.com/training/catalog/groovy-programming/
THANK YOU

Find me on twitter @paulk_asert
References

- Integer Linear Programming: Graphical Introduction https://www.youtube.com/watch?v=RhHhy-8sz-4
- Integer Linear Programming: Binary Constraints https://www.youtube.com/watch?v=-3my1TkyFiM https://www.youtube.com/watch?v=B3biWsBLeCw https://www.youtube.com/watch?v=MO8uQnIch6I
- Linear Programming: Alternate solutions, Infeasibility, Unboundedness, & Redundancy https://www.youtube.com/watch?v=eMA0LWsRQQ