DATA
SYNDROME

Predictive Analytics with Airflow and PySpark

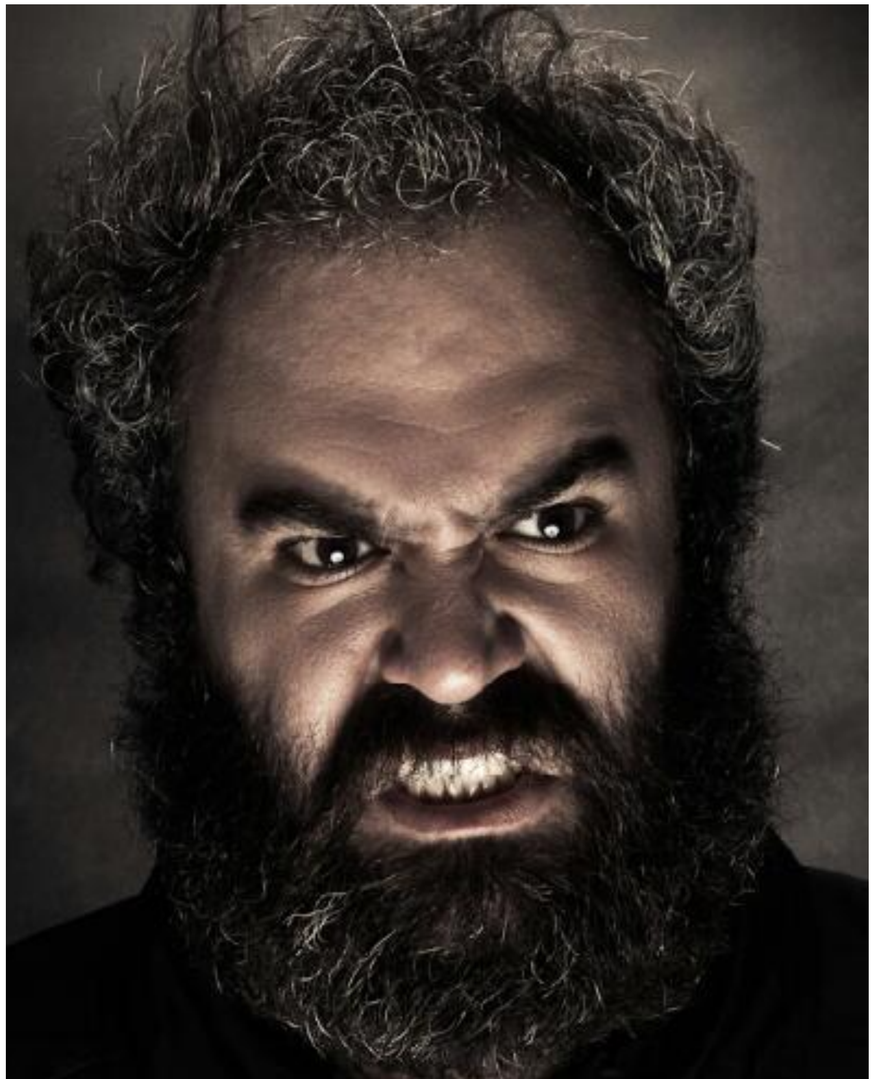
A Beginner's Guide to Building with Airflow

https://www.slideshare.net/rjurney/predictive-analytics-with-airflow-and-pyspark

http://bit.ly/airflow_pyspark

Russell Jurney

Principal Consultant at Data Syndrome



Russell Jurney

Russell Jurney is a veteran data scientist and thought leader. He coined the term Agile Data Science in the book of that name from O'Reilly in 2012, which outlines the first agile development methodology for data science. Russell has constructed numerous full-stack analytics products over the past ten years and now works with clients helping them extract value from their data assets.

Russell Jurney
Principal Consultant
Data Syndrome, LLC
Email : russell.jurney@gmail.com
Web : datasyndrome.com

Skill

- Data Engineer — 85%
- Data Scientist — 85%
- Visualization Software Engineer — 85%
- Writer — 85%
- Teacher — 50%

DATA SYNDROME

Product Consulting

We build analytics products and systems consisting of big data viz, predictions, recommendations, reports and search.

Corporate Training

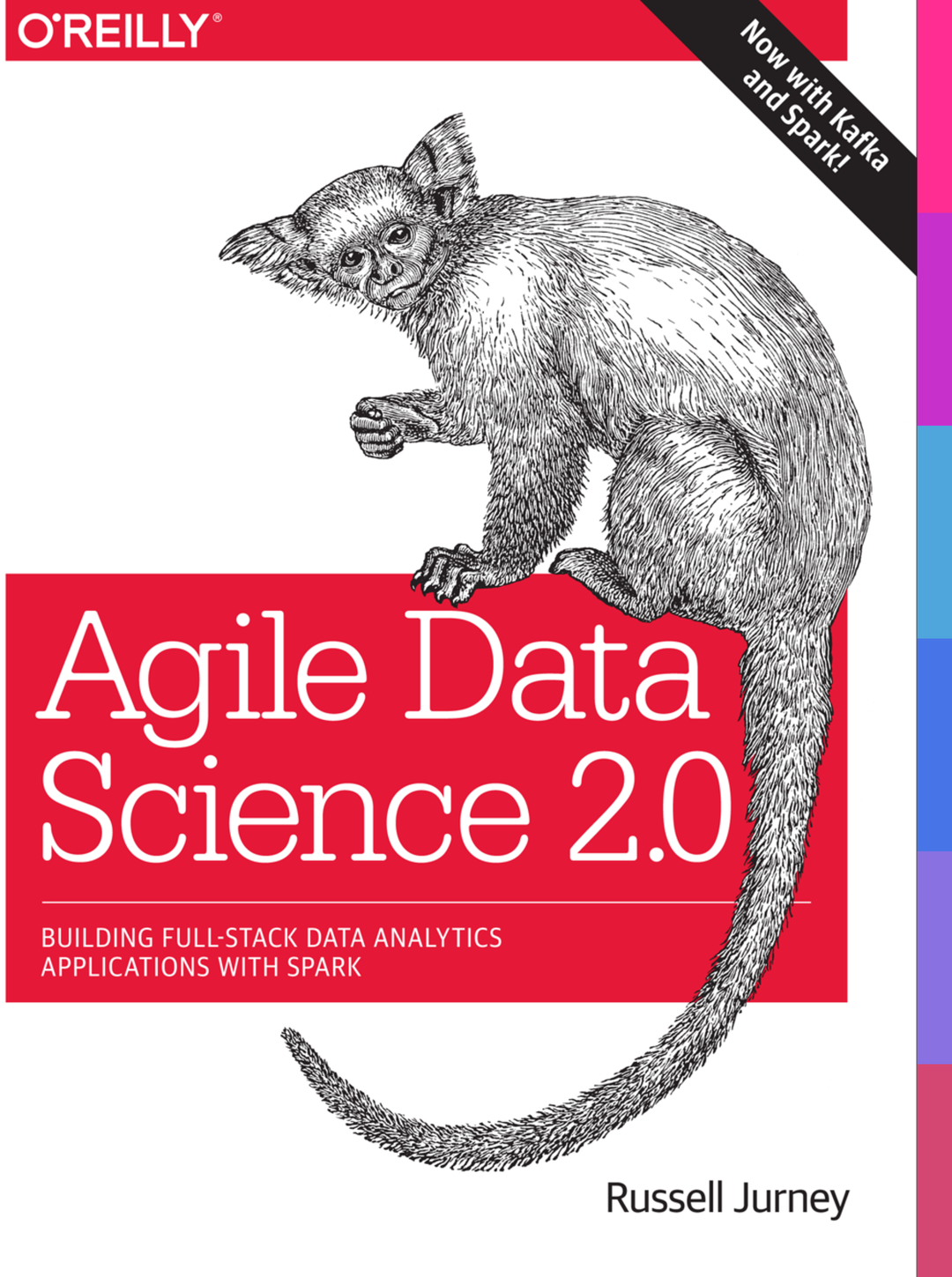
We offer training courses for data scientists and engineers and data science teams,

Video Training

We offer video training courses that rapidly acclimate you with a technology and technique.

An End to End Analytics Web App with Airflow

The presentation I wish I had when I started using Airflow



Agile Data Science 2.0 Stack

Example of a high productivity stack for “big” data applications

Apache Spark

Batch and Realtime

Apache Kafka

Realtime Queue

MongoDB

Document Store

ElasticSearch

Search

Flask

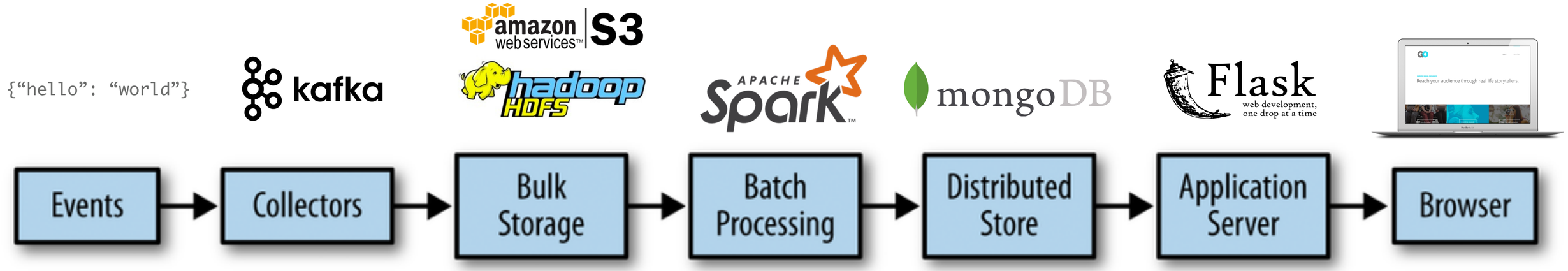
Simple Web App

Airflow

Scheduling

Flow of Data Processing

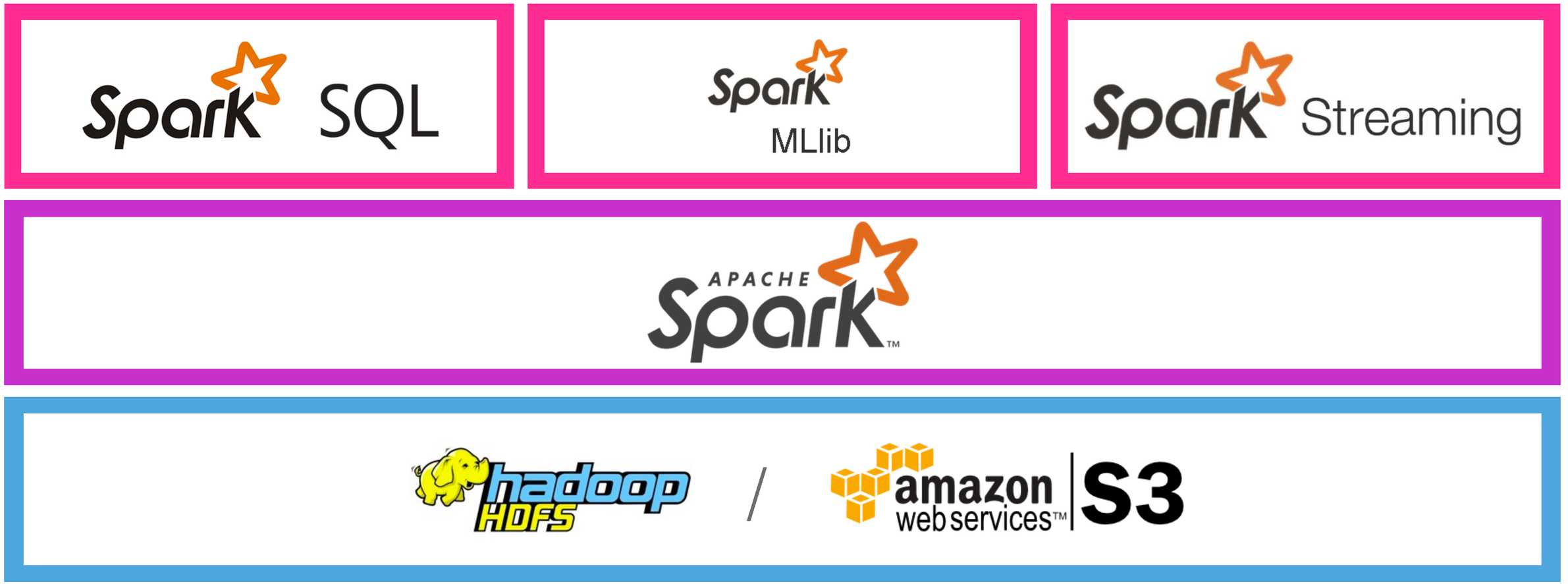
Tools and processes in collecting, refining, publishing and decorating data



Coordination

Apache Spark Ecosystem

HDFS, Amazon S3, Spark, Spark SQL, Spark MLlib, Spark Streaming



Programming Models

SQL or dataflow programming?

```
# Flights that were late arriving...
late_arrivals =
on_time_dataframe.filter(on_time_dataframe.ArrD
elayMinutes > 0)
total_late_arrivals = late_arrivals.count()

# Flights that left late but made up time to
arrive on time...
on_time_heros = on_time_dataframe.filter(
  (on_time_dataframe.DepDelayMinutes > 0)
  &
  (on_time_dataframe.ArrDelayMinutes <= 0)
)
total_on_time_heros = on_time_heros.count()

# Get the percentage of flights that are late,
rounded to 1 decimal place
pct_late = round((total_late_arrivals /
(total_flights * 1.0)) * 100, 1)

print("Total flights:
{:,}".format(total_flights))
print("Late departures:
{:,}".format(total_late_departures))
print("Late arrivals:
{:,}".format(total_late_arrivals))
print("Recoveries:
{:,}".format(total_on_time_heros))
print("Percentage Late: {}%".format(pct_late))

# Why are flights late? Lets look at some
delayed flights and the delay causes
late_flights = spark.sql("""
SELECT
  ArrDelayMinutes,
  WeatherDelay,
  CarrierDelay,
  NASDelay,
  SecurityDelay,
  LateAircraftDelay
FROM
  on_time_performance
WHERE
  WeatherDelay IS NOT NULL
  OR
  CarrierDelay IS NOT NULL
  OR
  NASDelay IS NOT NULL
  OR
  SecurityDelay IS NOT NULL
  OR
  LateAircraftDelay IS NOT NULL
ORDER BY
  FlightDate
""")
late_flights.sample(False, 0.01).show()
```

```
# Calculate the percentage contribution to delay for each source
total_delays = spark.sql("""
SELECT
  ROUND(SUM(WeatherDelay)/SUM(ArrDelayMinutes) * 100, 1) AS
pct_weather_delay,
  ROUND(SUM(CarrierDelay)/SUM(ArrDelayMinutes) * 100, 1) AS
pct_carrier_delay,
  ROUND(SUM(NASDelay)/SUM(ArrDelayMinutes) * 100, 1) AS pct_nas_delay,
  ROUND(SUM(SecurityDelay)/SUM(ArrDelayMinutes) * 100, 1) AS
pct_security_delay,
  ROUND(SUM(LateAircraftDelay)/SUM(ArrDelayMinutes) * 100, 1) AS
pct_late_aircraft_delay
FROM on_time_performance
""")
total_delays.show()

# Generate a histogram of the weather and carrier delays
weather_delay_histogram = on_time_dataframe\
  .select("WeatherDelay")\
  .rdd\
  .flatMap(lambda x: x)\
  .histogram(10)

print("{}\n{}".format(weather_delay_histogram[0],
weather_delay_histogram[1]))

# Eyeball the first to define our buckets
weather_delay_histogram = on_time_dataframe\
  .select("WeatherDelay")\
  .rdd\
  .flatMap(lambda x: x)\
  .histogram([1, 15, 30, 60, 120, 240, 480, 720, 24*60.0])
print(weather_delay_histogram)

# Transform the data into something easily consumed by d3
record = {'key': 1, 'data': []}
for label, count in zip(weather_delay_histogram[0],
weather_delay_histogram[1]):
  record['data'].append(
    {
      'label': label,
      'count': count
    }
  )

# Save to Mongo directly, since this is a Tuple not a dataframe or RDD
from pymongo import MongoClient
client = MongoClient()
client.relato.weather_delay_histogram.insert_one(record)
```

SQL **AND** Dataflow Programming

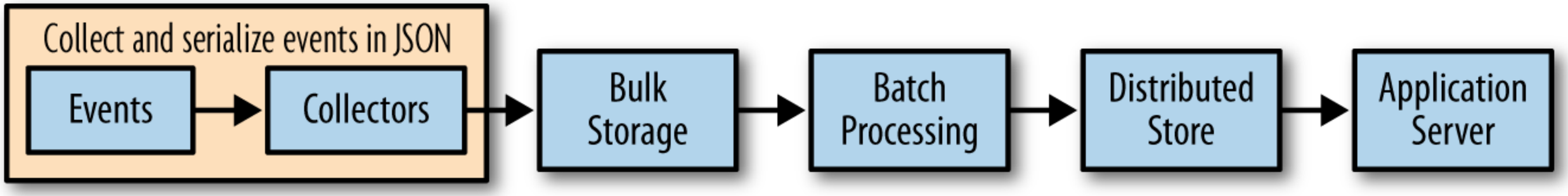
The best of both worlds!

Data

FAA on-time performance data

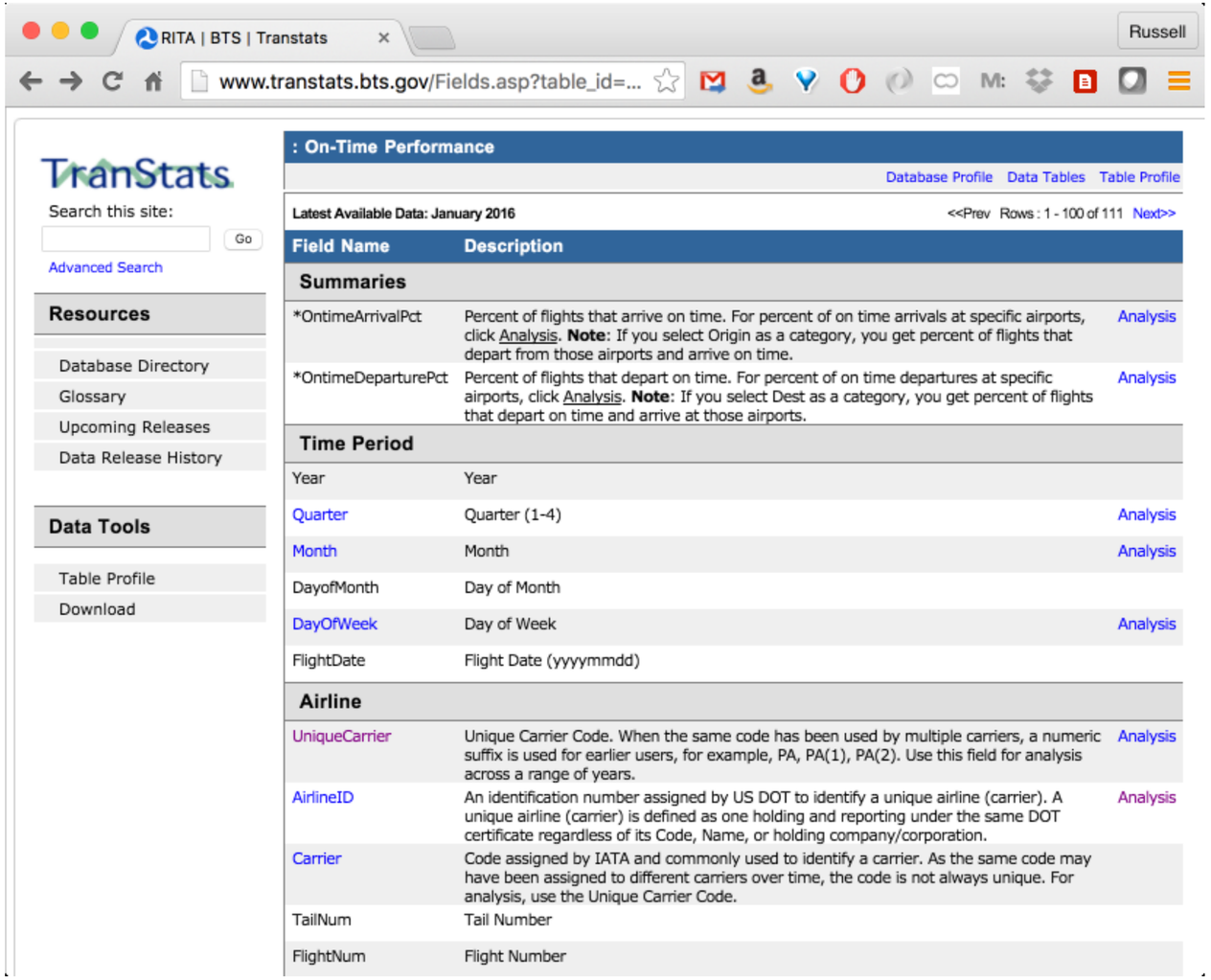
Collect and Serialize Events in JSON

I never regret using JSON



FAA On-Time Performance Records

95% of commercial flights



http://www.transtats.bts.gov/Fields.asp?table_id=236

FAA On-Time Performance Records

95% of commercial flights

"Year","Quarter","Month","DayofMonth","DayOfWeek","**FlightDate**","UniqueCarrier","AirlineID","**Carrier**","TailNum","**FlightNum**",
"OriginAirportID","OriginAirportSeqID","OriginCityMarketID","**Origin**","OriginCityName","OriginState","OriginStateFips",
"OriginStateName","OriginWac","DestAirportID","DestAirportSeqID","DestCityMarketID","**Dest**","DestCityName","DestState",
"DestStateFips","DestStateName","DestWac","**CRSDepTime**","**DepTime**","**DepDelay**","DepDelayMinutes","DepDel15","DepartureDelayGroups",
"DepTimeBlk","TaxiOut","WheelsOff","WheelsOn","TaxiIn","**CRSArrTime**","**ArrTime**","**ArrDelay**","ArrDelayMinutes","ArrDel15",
"ArrivalDelayGroups","ArrTimeBlk","**Cancelled**","CancellationCode","Diverted","CRSElapsedTime","ActualElapsedTime","**AirTime**",
"Flights","**Distance**","DistanceGroup","**CarrierDelay**","**WeatherDelay**","**NASDelay**","**SecurityDelay**","**LateAircraftDelay**",
"FirstDepTime","TotalAddGTime","LongestAddGTime","DivAirportLandings","DivReachedDest","DivActualElapsedTime","DivArrDelay",
"DivDistance","Div1Airport","Div1AirportID","Div1AirportSeqID","Div1WheelsOn","Div1TotalGTime","Div1LongestGTime",
"Div1WheelsOff","Div1TailNum","Div2Airport","Div2AirportID","Div2AirportSeqID","Div2WheelsOn","Div2TotalGTime",
"Div2LongestGTime","Div2WheelsOff","Div2TailNum","Div3Airport","Div3AirportID","Div3AirportSeqID","Div3WheelsOn",
"Div3TotalGTime","Div3LongestGTime","Div3WheelsOff","Div3TailNum","Div4Airport","Div4AirportID","Div4AirportSeqID",
"Div4WheelsOn","Div4TotalGTime","Div4LongestGTime","Div4WheelsOff","Div4TailNum","Div5Airport","Div5AirportID",
"Div5AirportSeqID","Div5WheelsOn","Div5TotalGTime","Div5LongestGTime","Div5WheelsOff","Div5TailNum"

openflights.org Database

Airports, Airlines, Routes

OpenFlights: Airport and airline

Russell

openflights.org/data.html

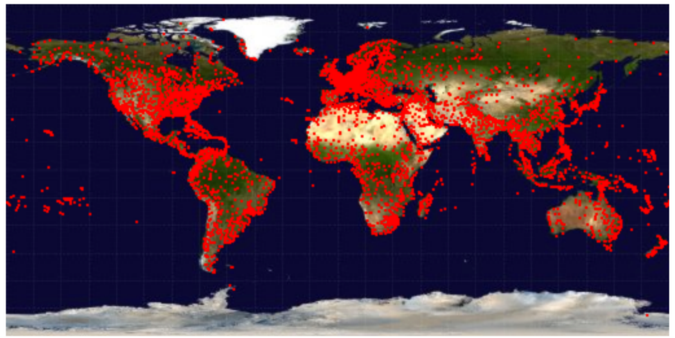
Airport, airline and route data

OpenFlights.org

About | FAQ | Blog

Navigation: Airport | Airline | Route | Schedule | Other | License

Airport database



(click to enlarge)

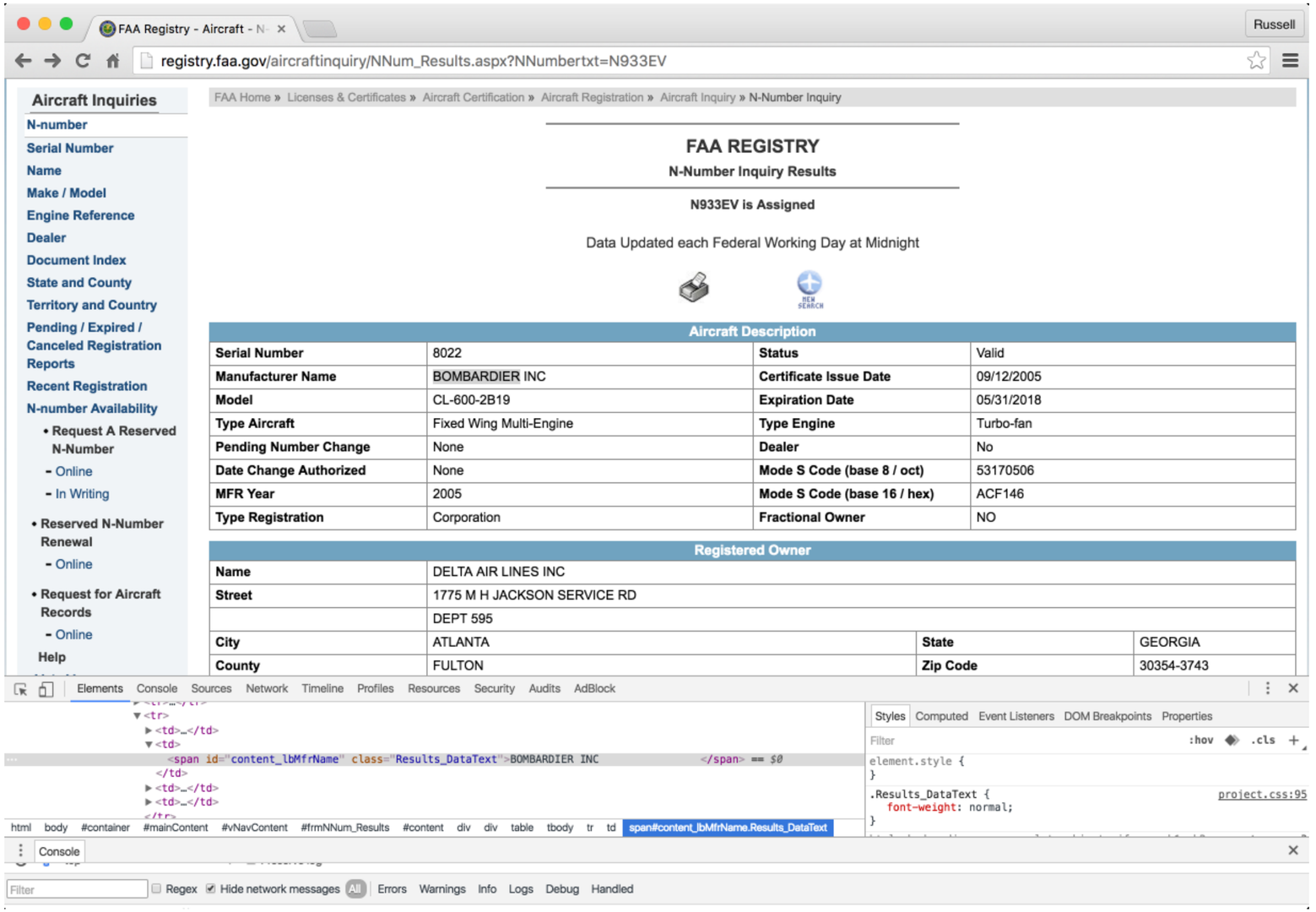
As of January 2017, the OpenFlights Airports Database contains **over 10,000** airports, train stations and ferry terminals spanning the globe, as shown in the map above. Each entry contains the following information:

Airport ID	Unique OpenFlights identifier for this airport.
Name	Name of airport. May or may not contain the **City** name.
City	Main city served by airport. May be spelled differently from **Name**.
Country	Country or territory where airport is located. See countries.dat to cross-reference to ISO 3166-1 codes.
IATA	3-letter IATA code. Null if not assigned/unknown.
ICAO	4-letter ICAO code. Null if not assigned.
Latitude	Decimal degrees, usually to six significant digits. Negative is South, positive is North.
Longitude	Decimal degrees, usually to six significant digits. Negative is West, positive is East.
Altitude	In feet.
Timezone	Hours offset from UTC. Fractional hours are expressed as decimals, eg. India is 5.5.
DST	Daylight savings time. One of E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) or U (Unknown). *See also: Help: Time*
Tz database time zone	Timezone in "tz" (Olson) format, eg. "America/Los_Angeles".
Type	Type of the airport. Value "airport" for air terminals, "station" for train stations, "port" for ferry terminals and "unknown" if not known. *In airports.csv, only type=airport is included.*
Source	Source of this data. "OurAirports" for data sourced from OurAirports, "Legacy" for old data not matched to OurAirports (mostly DAFIF), "User" for unverified user contributions. *In airports.csv, only source=OurAirports is included.*

The data is UTF-8 (Unicode) encoded.

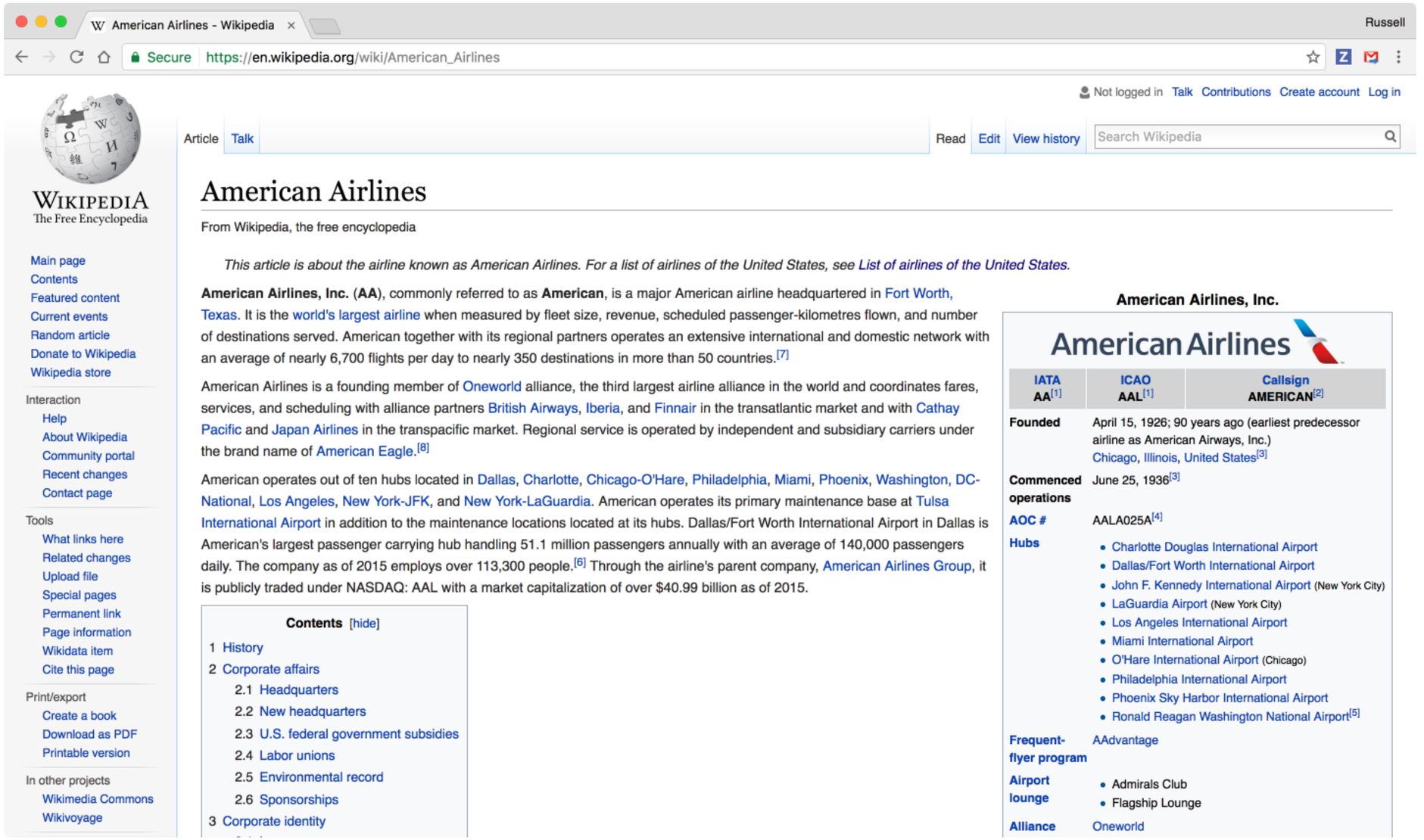
Scraping the FAA Registry

Airplane Data by Tail Number



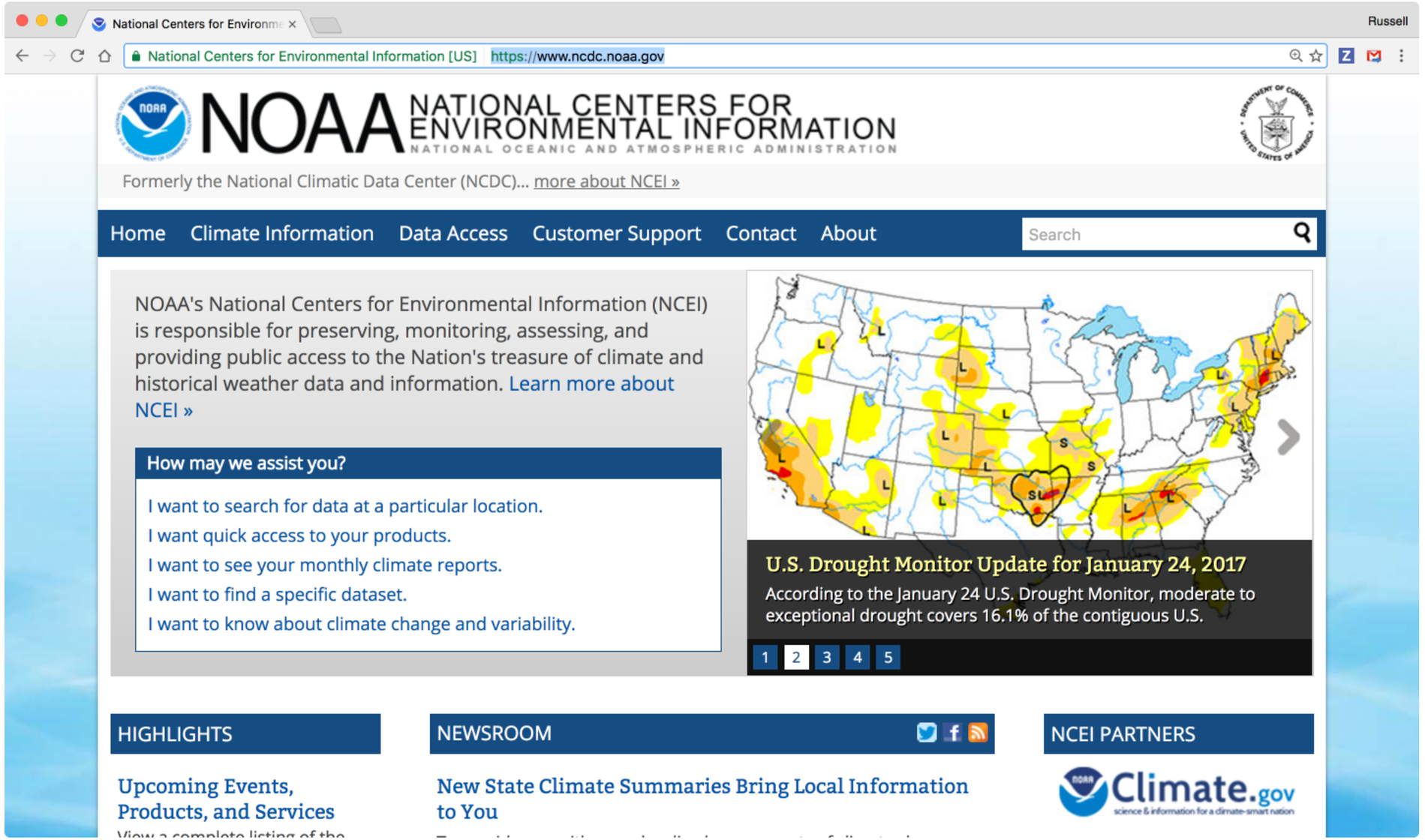
Wikipedia Airlines Entries

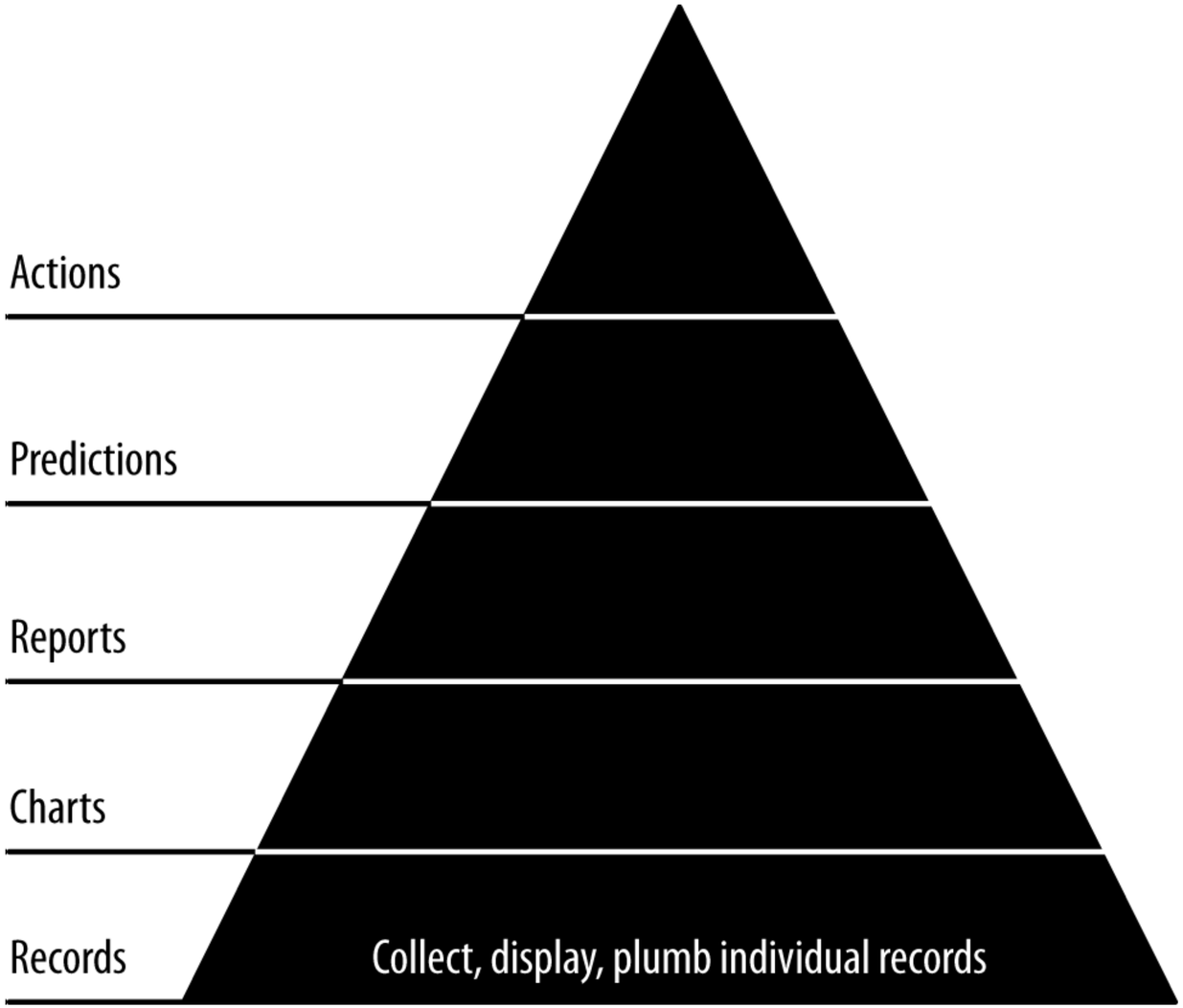
Descriptions of Airlines



National Centers for Environmental Information

Historical Weather Observations



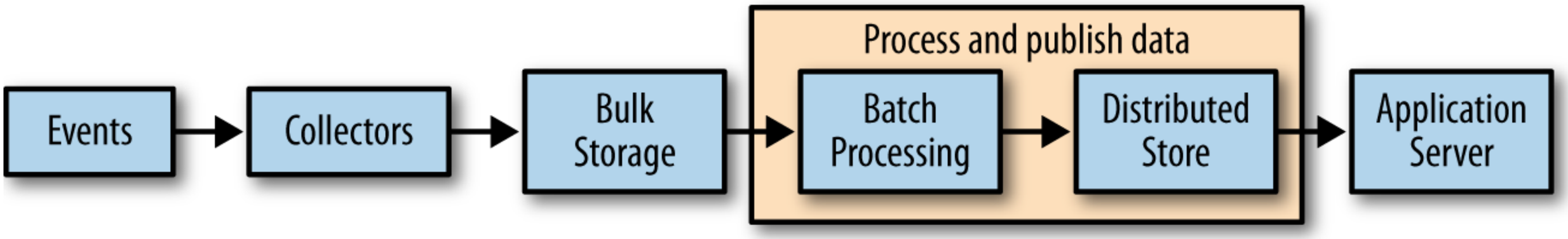


Plumbing

Starting by “plumbing” the system from end to end

Publishing Flight Records

Plumbing our master records through to the web



Airflow DAG Setup

Defining the pattern through which Airflow will work

ch02/airflow_test.py

```
import sys, os, re

from airflow import DAG
from airflow.operators.bash_operator import BashOperator

from datetime import datetime, timedelta
import iso8601

project_home = os.environ["PROJECT_HOME"]

default_args = {
  'owner': 'airflow',
  'depends_on_past': False,
  'start_date': iso8601.parse_date("2016-12-01"),
  'email': ['russell.jurney@gmail.com'],
  'email_on_failure': True,
  'email_on_retry': True,
  'retries': 3,
  'retry_delay': timedelta(minutes=5),
}

# Timedelta 1 is 'run daily'
dag = DAG(
  'agile_data_science_airflow_test',
  default_args=default_args,
  schedule_interval=timedelta(1)
)
```

Anatomy of an Airflow PySpark Task

Defining the pattern through which Airflow will work

ch02/airflow_test.py

```
# Run a simple PySpark Script
pyspark_local_task_one = BashOperator(
  task_id = "pyspark_local_task_one",
  bash_command = """spark-submit \
  --master {{ params.master }}
  {{ params.base_path }}/{{ params.filename }} {{ ds }} {{ params.base_path }}""",
  params = {
    "master": "local[8]",
    "filename": "ch02/pyspark_task_one.py",
    "base_path": "{}/".format(project_home)
  },
  dag=dag
)
```

The PySpark Task Itself

Initializing the PySpark Environment

ch02/pyspark_task_one.py

```
#!/usr/bin/env python

import sys, os, re
import json
import datetime, iso8601

# Pass date and base path to main() from airflow
def main(iso_date, base_path):
  APP_NAME = "pyspark_task_one.py"

  # If there is no SparkSession, create the environment
  try:
    sc and spark
  except NameError as e:
    import findspark
    findspark.init()
    import pyspark
    import pyspark.sql

    sc = pyspark.SparkContext()
    spark = pyspark.sql.SparkSession(sc).builder.appName(APP_NAME).getOrCreate()
```

Date Math and Input Path

Setting up the input path

ch02/pyspark_task_one.py

```
# Get today's date
today_dt = iso8601.parse_date(iso_date)
rounded_today = today_dt.date()

# Load today's data
today_input_path = "{}/ch02/data/example_name_titles_daily.json/{}".format(
  base_path,
  rounded_today.isoformat()
)
```

The actual work of the PySpark Job

Getting things done…

ch02/pyspark_task_one.py

```
# Otherwise load the data and proceed...
people_titles = spark.read.json(today_input_path)
people_titles.show()

# Group by as an RDD
titles_by_name = people_titles.rdd.groupBy(lambda x: x["name"])

# Accept the group key/grouped data and concatenate the various titles...
# into a master title
def concatenate_titles(people_titles):
  name = people_titles[0]
  title_records = people_titles[1]
  master_title = ""
  for title_record in sorted(title_records):
    title = title_record["title"]
    master_title += "{}, ".format(title)
  master_title = master_title[:-2]
  record = {"name": name, "master_title": master_title}
  return record

people_with_contactenated_titles = titles_by_name.map(concatenate_titles)
people_output_json = people_with_contactenated_titles.map(json.dumps)
```

Finishing up the PySpark Task

Finishing up getting things done…

ch02/pyspark_task_one.py

```
  # Get today's output path
    today_output_path = "{}/ch02/data/example_master_titles_daily.json/{}".format(
      base_path,
      rounded_today.isoformat()
    )

    # Write/replace today's output path
    os.system("rm -rf {}".format(today_output_path))
    people_output_json.saveAsTextFile(today_output_path)

if __name__ == "__main__":
    main(sys.argv[1], sys.argv[2])
```

Testing this Task from the Command Line

Making sure things work outside Spark

```
python ch02/pyspark_task_one.py 2016-12-01 .
```

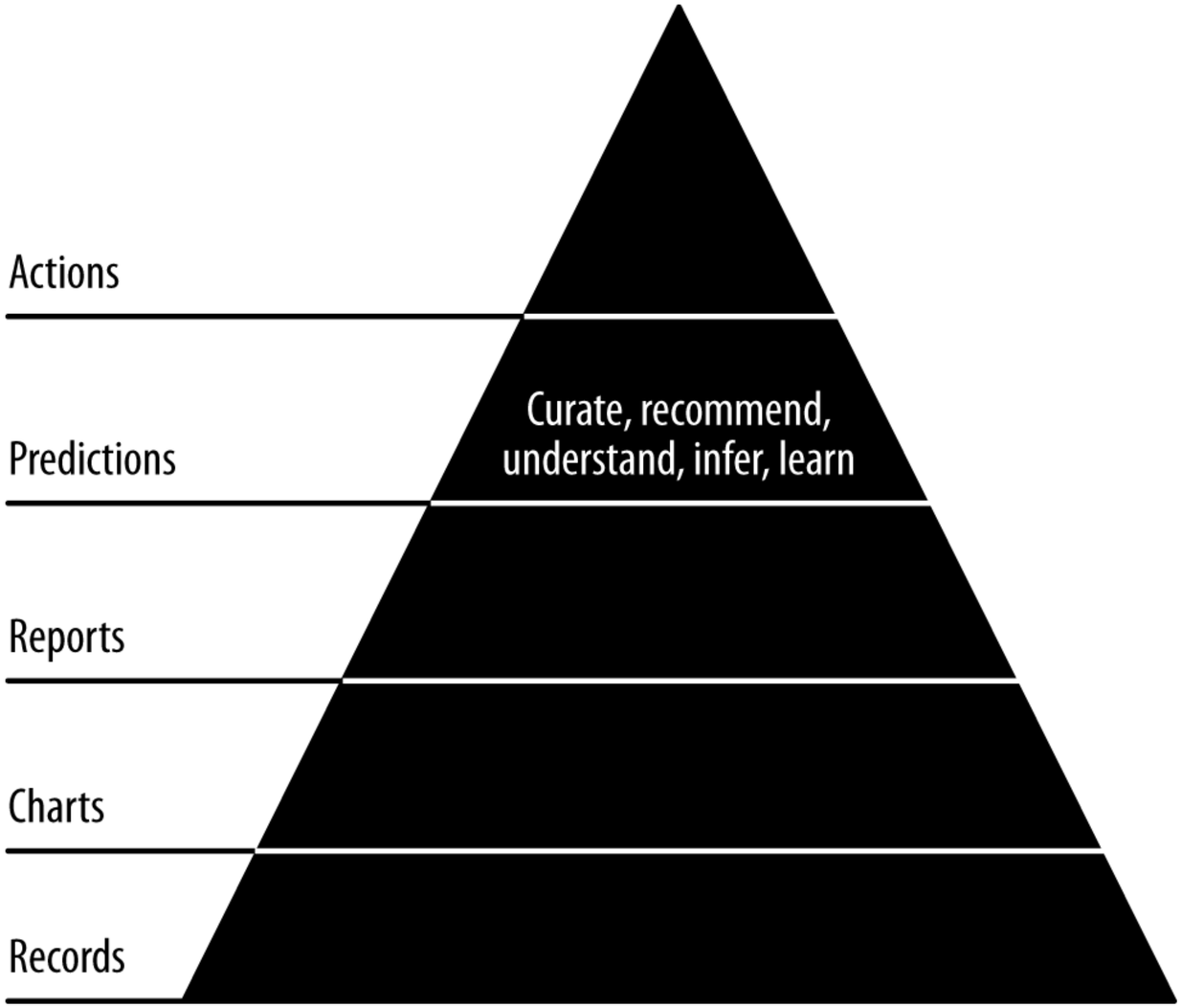
```
Ivy Default Cache set to: /Users/rjurney/.ivy2/cache
The jars for the packages stored in: /Users/rjurney/.ivy2/jars
:: loading settings :: url = jar:file:/Users/rjurney/Software/Agile_Data_Code_2/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-streaming-kafka-0-8_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
	confs: [default]
	found org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.0 in central
	found org.apache.kafka#kafka_2.11;0.8.2.1 in central
	found org.scala-lang.modules#scala-xml_2.11;1.0.2 in central
	found com.yammer.metrics#metrics-core;2.2.0 in list
	found org.slf4j#slf4j-api;1.7.16 in central
	found org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 in central
	found com.101tec#zkclient;0.3 in list
	found log4j#log4j;1.2.17 in list
	found org.apache.kafka#kafka-clients;0.8.2.1 in central
	found net.jpountz.lz4#lz4;1.3.0 in list
	found org.xerial.snappy#snappy-java;1.1.2.6 in central
	found org.apache.spark#spark-tags_2.11;2.1.0 in central
	found org.scalatest#scalatest_2.11;2.2.6 in central
	found org.scala-lang#scala-reflect;2.11.8 in central
	found org.spark-project.spark#unused;1.0.0 in list
:: resolution report :: resolve 2248ms :: artifacts dl 8ms
	:: modules in use:
	com.101tec#zkclient;0.3 from list in [default]
	com.yammer.metrics#metrics-core;2.2.0 from list in [default]
	log4j#log4j;1.2.17 from list in [default]
	net.jpountz.lz4#lz4;1.3.0 from list in [default]
	org.apache.kafka#kafka-clients;0.8.2.1 from central in [default]
	org.apache.kafka#kafka_2.11;0.8.2.1 from central in [default]
	org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.0 from central in [default]
	org.apache.spark#spark-tags_2.11;2.1.0 from central in [default]
	org.scala-lang#scala-reflect;2.11.8 from central in [default]
	org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 from central in [default]
	org.scala-lang.modules#scala-xml_2.11;1.0.2 from central in [default]
	org.scalatest#scalatest_2.11;2.2.6 from central in [default]
	org.slf4j#slf4j-api;1.7.16 from central in [default]
	org.spark-project.spark#unused;1.0.0 from list in [default]
	org.xerial.snappy#snappy-java;1.1.2.6 from central in [default]
	---------------------------------------------------------------------
	|                  |            modules            ||   artifacts   |
	|       conf       | number| search|dwnlded|evicted|| number|dwnlded|
	---------------------------------------------------------------------
	|      default     |  15   |   2   |   2   |   0   ||  15   |   0   |
	---------------------------------------------------------------------

:: problems summary ::
:::: ERRORS
	unknown resolver fs


:: USE VERBOSE OR DEBUG MESSAGE LEVEL FOR MORE DETAILS
:: retrieving :: org.apache.spark#spark-submit-parent
	confs: [default]
	0 artifacts copied, 15 already retrieved (0kB/6ms)
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/03/14 12:52:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/03/14 12:52:22 WARN SparkConf:
SPARK_CLASSPATH was detected (set to '/Users/rjurney/Software/Agile_Data_Code_2/lib/snappy-java-1.1.2.6.jar').
This is deprecated in Spark 1.0+.

Please instead use:
 - ./spark-submit with --driver-class-path to augment the driver classpath
 - spark.executor.extraClassPath to augment the executor classpath

17/03/14 12:52:22 WARN SparkConf: Setting 'spark.executor.extraClassPath' to '/Users/rjurney/Software/Agile_Data_Code_2/lib/snappy-java-1.1.2.6.jar' as a work-around.
17/03/14 12:52:22 WARN SparkConf: Setting 'spark.driver.extraClassPath' to '/Users/rjurney/Software/Agile_Data_Code_2/lib/snappy-java-1.1.2.6.jar' as a work-around.
```

Testing this Task from the Command Line

Making sure things work outside Spark

```
cat ch02/data/example_master_titles_daily.json/2016-12-01/part-00000
```

```
{"master_title": "Author, Data Scientist, Dog Lover", "name": "Russell Jurney"}
{"master_title": "Attorney", "name": "Susan Shu"}
{"master_title": "CEO", "name": "Bob Jones"}
```



Predictions

Predicting the future for fun and profit

```
# !/usr/bin/env python

import sys, os, re

# Pass date and base path to main() from airflow
def main(base_path):

  # Default to "."
  try: base_path
  except NameError: base_path = "."
  if not base_path:
    base_path = "."

  APP_NAME = "train_spark_mllib_model.py"

  # If there is no SparkSession, create the environment
  try:
    sc and spark
  except NameError as e:
    import findspark
    findspark.init()
    import pyspark
    import pyspark.sql

    sc = pyspark.SparkContext()
    spark = pyspark.sql.SparkSession(sc).builder.appName(APP_NAME).getOrCreate()

  #
  # {
  #   "ArrDelay":5.0,"CRSArrTime":"2015-12-31T03:20:00.000-08:00","CRSDepTime":"2015-12-31T03:05:00.000-08:00",
  #   "Carrier":"WN","DayOfMonth":31,"DayOfWeek":4,"DayOfYear":365,"DepDelay":14.0,"Dest":"SAN","Distance":368.0,
  #   "FlightDate":"2015-12-30T16:00:00.000-08:00","FlightNum":"6109","Origin":"TUS"
  # }
  #
  from pyspark.sql.types import StringType, IntegerType, FloatType, DoubleType, DateType, TimestampType
  from pyspark.sql.types import StructType, StructField
  from pyspark.sql.functions import udf

  schema = StructType([
    StructField("ArrDelay", DoubleType(), True),     # "ArrDelay":5.0
    StructField("CRSArrTime", TimestampType(), True),    # "CRSArrTime":"2015-12-31T03:20:00.000-08:00"
    StructField("CRSDepTime", TimestampType(), True),    # "CRSDepTime":"2015-12-31T03:05:00.000-08:00"
    StructField("Carrier", StringType(), True),     # "Carrier":"WN"
    StructField("DayOfMonth", IntegerType(), True), # "DayOfMonth":31
    StructField("DayOfWeek", IntegerType(), True),  # "DayOfWeek":4
    StructField("DayOfYear", IntegerType(), True),  # "DayOfYear":365
    StructField("DepDelay", DoubleType(), True),     # "DepDelay":14.0
    StructField("Dest", StringType(), True),        # "Dest":"SAN"
    StructField("Distance", DoubleType(), True),     # "Distance":368.0
    StructField("FlightDate", DateType(), True),    # "FlightDate":"2015-12-30T16:00:00.000-08:00"
    StructField("FlightNum", StringType(), True),   # "FlightNum":"6109"
    StructField("Origin", StringType(), True),      # "Origin":"TUS"
  ])

  input_path = "{}/data/simple_flight_delay_features.jsonl.bz2".format(
    base_path
  )
  features = spark.read.json(input_path, schema=schema)
  features.first()

  #
  # Check for nulls in features before using Spark ML
  #
  null_counts = [(column, features.where(features[column].isNull()).count()) for column in features.columns]
  cols_with_nulls = filter(lambda x: x[1] > 0, null_counts)
  print(list(cols_with_nulls))

  #
  # Add a Route variable to replace FlightNum
  #
  from pyspark.sql.functions import lit, concat
  features_with_route = features.withColumn(
    'Route',
    concat(
      features.Origin,
      lit('-'),
      features.Dest
    )
  )
  features_with_route.show(6)

  #
  # Use pysmark.ml.feature.Bucketizer to bucketize ArrDelay into on-time, slightly late, very late (0, 1, 2)
  #
  from pyspark.ml.feature import Bucketizer

  # Setup the Bucketizer
  splits = [-float("inf"), -15.0, 0, 30.0, float("inf")]
  arrival_bucketizer = Bucketizer(
    splits=splits,
    inputCol="ArrDelay",
    outputCol="ArrDelayBucket"
  )
```

```
  # Save the bucketizer
  arrival_bucketizer_path = "{}/models/arrival_bucketizer_2.0.bin".format(base_path)
  arrival_bucketizer.write().overwrite().save(arrival_bucketizer_path)

  # Apply the bucketizer
  ml_bucketized_features = arrival_bucketizer.transform(features_with_route)
  ml_bucketized_features.select("ArrDelay", "ArrDelayBucket").show()

  #
  # Extract features tools in with pyspark.ml.feature
  #
  from pyspark.ml.feature import StringIndexer, VectorAssembler

  # Turn category fields into indexes
  for column in ["Carrier", "Origin", "Dest", "Route"]:
    string_indexer = StringIndexer(
      inputCol=column,
      outputCol=column + "_index"
    )

    string_indexer_model = string_indexer.fit(ml_bucketized_features)
    ml_bucketized_features = string_indexer_model.transform(ml_bucketized_features)

    # Drop the original column
    ml_bucketized_features = ml_bucketized_features.drop(column)

    # Save the pipeline model
    string_indexer_output_path = "{}/models/string_indexer_model_{}.bin".format(
      base_path,
      column
    )
    string_indexer_model.write().overwrite().save(string_indexer_output_path)

  # Combine continuous, numeric fields with indexes of nominal ones
  # ...into one feature vector
  numeric_columns = [
    "DepDelay", "Distance",
    "DayOfMonth", "DayOfWeek",
    "DayOfYear"]
  index_columns = ["Carrier_index", "Origin_index",
                   "Dest_index", "Route_index"]
  vector_assembler = VectorAssembler(
    inputCols=numeric_columns + index_columns,
    outputCol="Features_vec"
  )
  final_vectorized_features = vector_assembler.transform(ml_bucketized_features)

  # Save the numeric vector assembler
  vector_assembler_path = "{}/models/numeric_vector_assembler.bin".format(base_path)
  vector_assembler.write().overwrite().save(vector_assembler_path)

  # Drop the index columns
  for column in index_columns:
    final_vectorized_features = final_vectorized_features.drop(column)

  # Inspect the finalized features
  final_vectorized_features.show()

  # Instantiate and fit random forest classifier on all the data
  from pyspark.ml.classification import RandomForestClassifier
  rfc = RandomForestClassifier(
    featuresCol="Features_vec",
    labelCol="ArrDelayBucket",
    predictionCol="Prediction",
    maxBins=4657,
    maxMemoryInMB=1024
  )
  model = rfc.fit(final_vectorized_features)

  # Save the new model over the old one
  model_output_path = "{}/models/spark_random_forest_classifier.flight_delays.5.0.bin".format(
    base_path
  )
  model.write().overwrite().save(model_output_path)

  # Evaluate model using test data
  predictions = model.transform(final_vectorized_features)

  from pyspark.ml.evaluation import MulticlassClassificationEvaluator
  evaluator = MulticlassClassificationEvaluator(
    predictionCol="Prediction",
    labelCol="ArrDelayBucket",
    metricName="accuracy"
  )
  accuracy = evaluator.evaluate(predictions)
  print("Accuracy = {}".format(accuracy))

  # Check the distribution of predictions
  predictions.groupBy("Prediction").count().show()

  # Check a sample
  predictions.sample(False, 0.001, 18).orderBy("CRSDepTime").show(6)

if __name__ == "__main__":
  main(sys.argv[1])
```

190 Line Model

scikit-learn was 166. Spark MLlib is very powerful!

http://bit.ly/train_model_spark

See ch08/train_spark_mllib_model.py

Training in Batch

Most machine learning still happens in batch on historical data…

Initializing the Environment

Setting up the environment…

ch08/train_spark_mllib_model.py

```
# !/usr/bin/env python

import sys, os, re

# Pass date and base path to main() from airflow
def main(base_path):

  # Default to "."
  try: base_path
  except NameError: base_path = "."
  if not base_path:
    base_path = "."

  APP_NAME = "train_spark_mllib_model.py"

  # If there is no SparkSession, create the environment
  try:
    sc and spark
  except NameError as e:
    import findspark
    findspark.init()
    import pyspark
    import pyspark.sql

    sc = pyspark.SparkContext()
    spark = pyspark.sql.SparkSession(sc).builder.appName(APP_NAME).getOrCreate()
```

Running Main

Just what it looks like…

ch08/train_spark_mllib_model.py

```
if __name__ == "__main__":
    main(sys.argv[1])
```

Setting up Airflow DAG for Model Training

Similar to test setup….

ch08/airflow/setup.py

```
import sys, os, re

from airflow import DAG
from airflow.operators.bash_operator import BashOperator

from datetime import datetime, timedelta
import iso8601

PROJECT_HOME = os.environ["PROJECT_HOME"]

default_args = {
  'owner': 'airflow',
  'depends_on_past': False,
  'start_date': iso8601.parse_date("2016-12-01"),
  'email': ['russell.jurney@gmail.com'],
  'email_on_failure': True,
  'email_on_retry': True,
  'retries': 3,
  'retry_delay': timedelta(minutes=5),
}

training_dag = DAG(
  'agile_data_science_batch_prediction_model_training',
  default_args=default_args
)
```

Setting up Reusable Templates

Repeating the same command patter over and over…

ch08/airflow/setup.py

```
# We use the same two commands for all our PySpark tasks
pyspark_bash_command = """
spark-submit --master {{ params.master }} \
  {{ params.base_path }}/{{ params.filename }} \
  {{ params.base_path }}
"""
pyspark_date_bash_command = """
spark-submit --master {{ params.master }} \
  {{ params.base_path }}/{{ params.filename }} \
  {{ ds }} {{ params.base_path }}
"""
```

Feature Extraction BashOperator

Job that gathers all data together for training the model…

ch08/airflow/setup.py

```
# Gather the training data for our classifier
extract_features_operator = BashOperator(
  task_id = "pyspark_extract_features",
  bash_command = pyspark_bash_command,
  params = {
    "master": "local[8]",
    "filename": "ch08/extract_features.py",
    "base_path": "{}/".format(PROJECT_HOME)
  },
  dag=training_dag
)
```

Model Training BashOperator

Job that actually trains the data….

ch08/airflow/setup.py

```
# Train and persist the classifier model
train_classifier_model_operator = BashOperator(
  task_id = "pyspark_train_classifier_model",
  bash_command = pyspark_bash_command,
  params = {
    "master": "local[8]",
    "filename": "ch08/train_spark_mllib_model.py",
    "base_path": "{}/".format(PROJECT_HOME)
  },
  dag=training_dag

# The model training depends on the feature extraction
train_classifier_model_operator.set_upstream(extract_features_operator)
```

Deploying in Batch

Next steps for deploying the model in batch

Separate Daily Prediction DAG

DAG that will act daily to make predictions

ch08/airflow/setup.py

```
daily_prediction_dag = DAG(
  'agile_data_science_batch_predictions_daily',
  default_args=default_args,
  schedule_interval=timedelta(1)
)
```

Fetch Prediction Requests from MongoDB

Prediction requests accumulate in MongoDB for each day

ch08/airflow/setup.py

```
# Fetch prediction requests from MongoDB
fetch_prediction_requests_operator = BashOperator(
  task_id = "pyspark_fetch_prediction_requests",
  bash_command = pyspark_date_bash_command,
  params = {
    "master": "local[8]",
    "filename": "ch08/fetch_prediction_requests.py",
    "base_path": "{}/".format(PROJECT_HOME)
  },
  dag=daily_prediction_dag
)
```

Fetch Prediction Requests from MongoDB

Prediction requests accumulate in MongoDB for each day

ch08/fetch_prediction_requests.py

```
# Get today and tomorrow's dates as iso strings to scope query
today_dt = iso8601.parse_date(iso_date)
rounded_today = today_dt.date()
iso_today = rounded_today.isoformat()
rounded_tomorrow_dt = rounded_today + datetime.timedelta(days=1)
iso_tomorrow = rounded_tomorrow_dt.isoformat()

# Create mongo query string for today's data
mongo_query_string = """{{
  "Timestamp": {{
    "$gte": "{iso_today}",
    "$lte": "{iso_tomorrow}"
  }}
}}""".format(
  iso_today=iso_today,
  iso_tomorrow=iso_tomorrow
)
mongo_query_string = mongo_query_string.replace('\n', '')

# Create the config object with the query string
mongo_query_config = dict()
mongo_query_config["mongo.input.query"] = mongo_query_string
```

```
# Load the day's requests using pymongo_spark
prediction_requests = sc.mongoRDD(
  'mongodb://localhost:27017/agile_data_science.prediction_tasks',
  config=mongo_query_config
)

# Build the day's output path: a date based primary key directory structure
today_output_path = "{}/data/prediction_tasks_daily.json/{}".format(
  base_path,
  iso_today
)

# Generate json records
prediction_requests_json = prediction_requests.map(json_util.dumps)

# Write/replace today's output path
os.system("rm -rf {}".format(today_output_path))
prediction_requests_json.saveAsTextFile(today_output_path)
```

Make Today's Predictions

Make the predictions for today's batch

ch08/airflow/setup.py

```
# Make the actual predictions for today
make_predictions_operator = BashOperator(
  task_id = "pyspark_make_predictions",
  bash_command = pyspark_date_bash_command,
  params = {
    "master": "local[8]",
    "filename": "ch08/make_predictions.py",
    "base_path": "{}/".format(PROJECT_HOME)
  },
  dag=daily_prediction_dag
)
```

Load Today's Predictions into MongoDB

Publish the predictions to our database…

ch08/airflow/setup.py

```
# Load today's predictions to Mongo
load_prediction_results_operator = BashOperator(
  task_id = "pyspark_load_prediction_results",
  bash_command = pyspark_date_bash_command,
  params = {
    "master": "local[8]",
    "filename": "ch08/load_prediction_results.py",
    "base_path": "{}/".format(PROJECT_HOME)
  },
  dag=daily_prediction_dag
)
```

Setup Dependencies for Today's Predictions

Set downstream predictions between the three jobs

ch08/airflow/setup.py

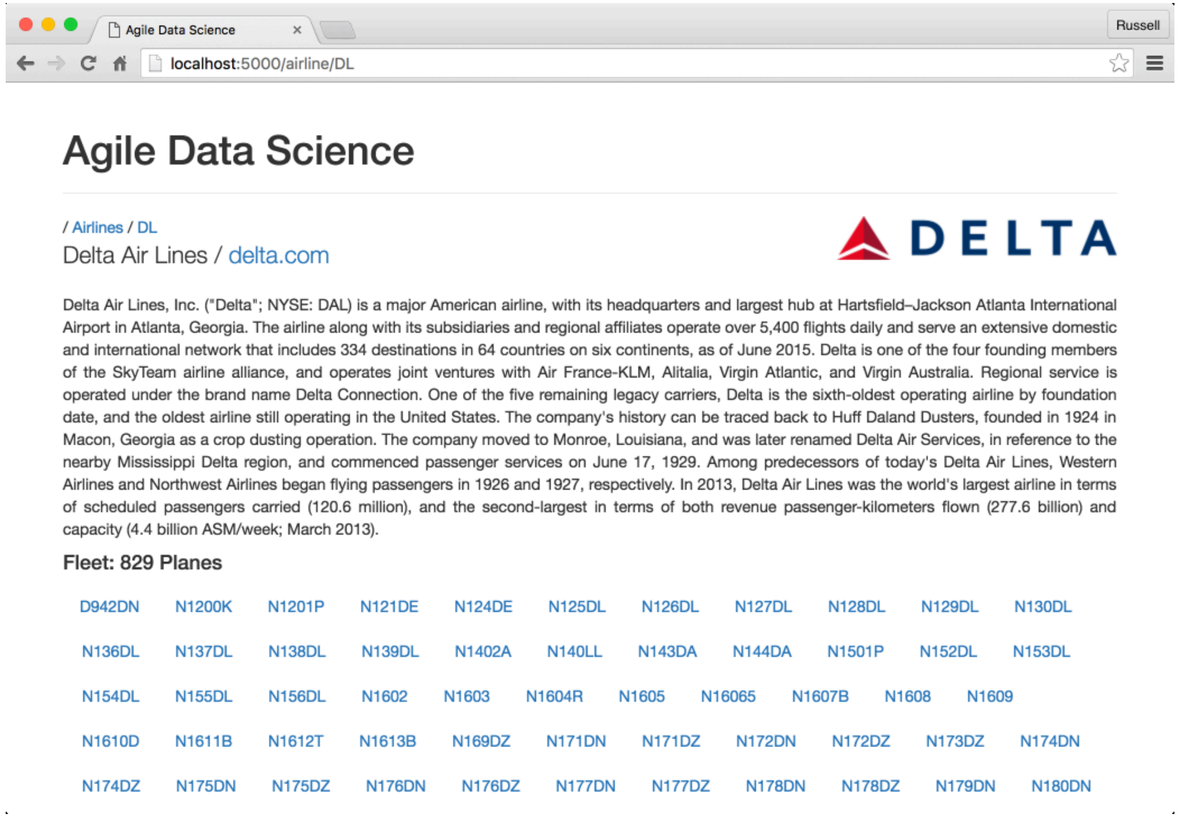
```
# Set downstream dependencies for daily_prediction_dag
fetch_prediction_requests_operator.set_downstream(make_predictions_operator)
make_predictions_operator.set_downstream(load_prediction_results_operator)
```

Testing Airflow Script

Testing our airflow setup with a script

ch08/test_airflow.sh

```
#!/bin/bash

# Compute today's date:
export ISO_DATE=`date "+%Y-%m-%d"`

# List DAGs
airflow list_dags

# List tasks in each DAG
airflow list_tasks agile_data_science_batch_prediction_model_training
airflow list_tasks agile_data_science_batch_predictions_daily

# Test each task in each DAG
airflow test agile_data_science_batch_prediction_model_training pyspark_extract_features $ISO_DATE
airflow test agile_data_science_batch_prediction_model_training pyspark_train_classifier_model $ISO_DATE

airflow test agile_data_science_batch_predictions_daily pyspark_fetch_prediction_requests $ISO_DATE
airflow test agile_data_science_batch_predictions_daily pyspark_make_predictions $ISO_DATE
airflow test agile_data_science_batch_predictions_daily pyspark_load_prediction_results $ISO_DATE

# Test the training and persistence of the models
airflow backfill -s $ISO_DATE -e $ISO_DATE agile_data_science_batch_prediction_model_training

# Test the daily operation of the model
airflow backfill -s $ISO_DATE -e $ISO_DATE agile_data_science_batch_predictions_daily
```



Putting it on the Web

It don't matter if don't nobody see it!

Flask Controller to Display Prediction Submission Form

Routing prediction requests from the user and results to the user

ch08/web/predict_flask.py

```
@app.route("/flights/delays/predict_batch")
def flight_delays_batch_page():
  """Serves flight delay predictions"""

  form_config = [
    {'field': 'DepDelay', 'label': 'Departure Delay'},
    {'field': 'Carrier'},
    {'field': 'FlightDate', 'label': 'Date'},
    {'field': 'Origin'},
    {'field': 'Dest', 'label': 'Destination'},
    {'field': 'FlightNum', 'label': 'Flight Number'},
  ]

  return render_template("flight_delays_predict_batch.html", form_config=form_config)
```

Flask Jinja2 Template to Prediction Submission Form

Routing prediction requests from the user and results to the user

ch08/web/templates/flight_delays_predict_batch.html

```
{% extends "layout.html" %}
{% block body %}
  <!-- Navigation guide -->
  / <a href="/flights/delays/predict_batch">Flight Delay Prediction via Spark in Batch</a>

  <p class="lead" style="margin: 10px; margin-left: 0px;">
    <!-- Airline Name and website-->
    Predicting Flight Delays via Spark in Batch
  </p>

  <!-- Generate form from search_config and request args -->
  <form id="flight_delay_classification" action="/flights/delays/predict/classify" method="post">
    {% for item in form_config %}
      {% if 'label' in item %}
        <label for="{{item['field']}}">{{item['label']}}</label>
      {% else %}
        <label for="{{item['field']}}">{{item['field']}}</label>
      {% endif %}
        <input name="{{item['field']}}" style="width: 36px; margin-right: 10px;" value=""></input>
    {% endfor %}
    <button type="submit" class="btn btn-xs btn-default" style="height: 25px">Submit</button>
  </form>

  <div style="margin-top: 10px;">
      <p>Prediction Request Successful: <span id="result" style="display: inline-block;"></span></p>
  </div>
```

Flask Javascript to Submit Predictions

Routing prediction requests from the user and results to the user

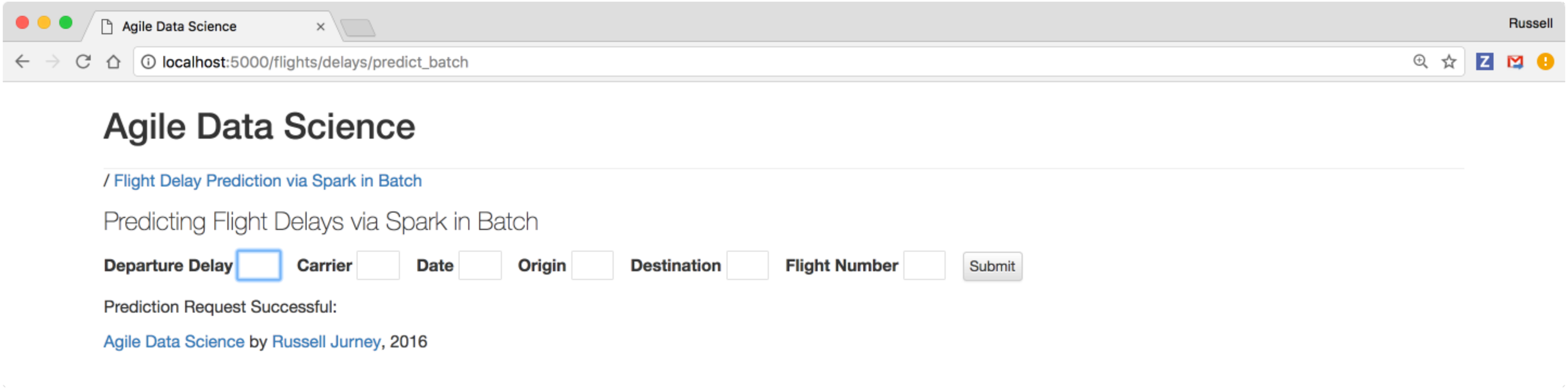
ch08/web/templates/flight_delays_predict_batch.html

```
<script>
   // Attach a submit handler to the form
   $( "#flight_delay_classification" ).submit(function( event ) {

     // Stop form from submitting normally
     event.preventDefault();

     // Get some values from elements on the page:
     var $form = $( this ),
       term = $form.find( "input[name='s']" ).val(),
       url = $form.attr( "action" );

     // Send the data using post
     var posting = $.post( url, $( "#flight_delay_classification" ).serialize() );

     // Put the results in a div
     posting.done(function( data ) {
       $( "#result" ).empty().append( data );
     });
   });
 </script>
{% endblock %}
```

Batch Deployment Application - Submit Page

What our end result looks like!

http://localhost:5000/flights/delays/predict_batch



Flask Controller to Submit Prediction Requests

Routing prediction requests from the user and results to the user

ch08/web/predict_flask.py

```
# Make our API a post, so a search engine wouldn't hit it
@app.route("/flights/delays/predict/classify", methods=['POST'])
def classify_flight_delays():
  """POST API for classifying flight delays"""
  api_field_type_map = \
    {
      "DepDelay": float,
      "Carrier": str,
      "FlightDate": str,
      "Dest": str,
      "FlightNum": str,
      "Origin": str
    }

  api_form_values = {}
  for api_field_name, api_field_type in api_field_type_map.items():
    api_form_values[api_field_name] = request.form.get(api_field_name, type=api_field_type)

  # Set the direct values, which excludes Date
  prediction_features = {}
  for key, value in api_form_values.items():
    prediction_features[key] = value

  # Set the derived values
  prediction_features['Distance'] = predict_utils.get_flight_distance(
    client, api_form_values['Origin'],
    api_form_values['Dest']
  )

  # Turn the date into DayOfYear, DayOfMonth, DayOfWeek
  date_features_dict = predict_utils.get_regression_date_args(
    api_form_values['FlightDate']
  )
  for api_field_name, api_field_value in date_features_dict.items():
    prediction_features[api_field_name] = api_field_value

  # Add a timestamp
  prediction_features['Timestamp'] = predict_utils.get_current_timestamp()

  client.agile_data_science.prediction_tasks.insert_one(
    prediction_features
  )
  return json_util.dumps(prediction_features)
```

Flask Controller to Display Prediction Requests

Routing prediction requests from the user and results to the user

ch08/web/predict_flask.py

```
@app.route("/flights/delays/predict_batch/results/<iso_date>")
def flight_delays_batch_results_page(iso_date):
  """Serves page for batch prediction results"""

  # Get today and tomorrow's dates as iso strings to scope query
  today_dt = iso8601.parse_date(iso_date)
  rounded_today = today_dt.date()
  iso_today = rounded_today.isoformat()
  rounded_tomorrow_dt = rounded_today + datetime.timedelta(days=1)
  iso_tomorrow = rounded_tomorrow_dt.isoformat()

  # Fetch today's prediction results from Mongo
  predictions = client.agile_data_science.prediction_results.find(
    {
      'Timestamp': {
        "$gte": iso_today,
        "$lte": iso_tomorrow,
      }
    }
  )

  return render_template(
    "flight_delays_predict_batch_results.html",
    predictions=predictions,
    iso_date=iso_date
  )
```

Flask Template to Display Predictions

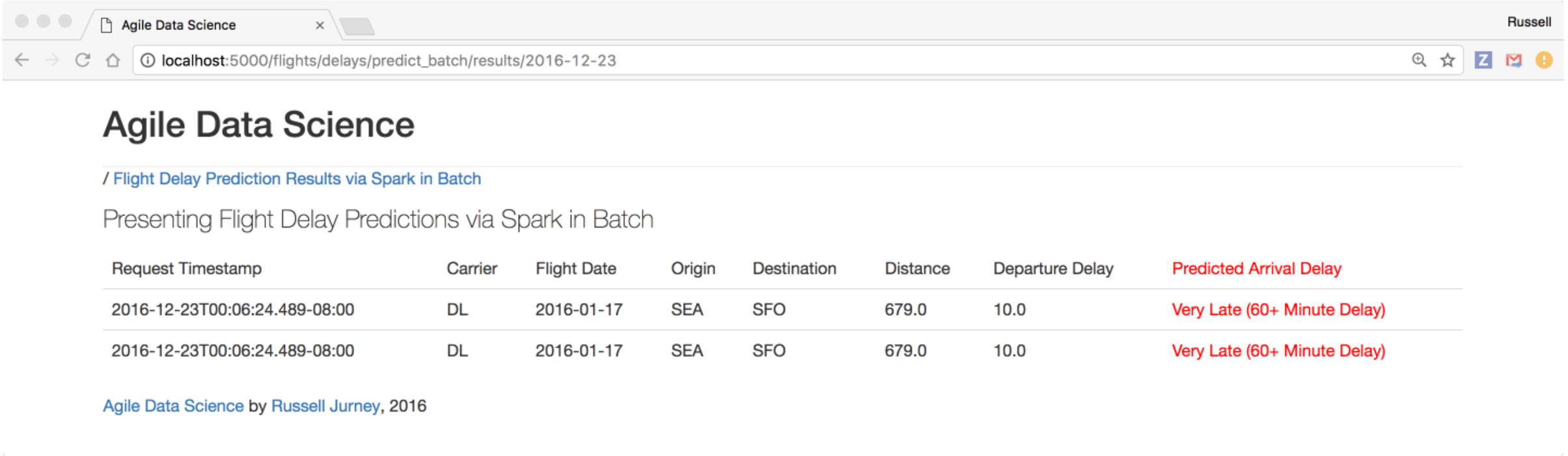
Routing prediction requests from the user and results to the user

ch08/web/templates/flight_delays_predict_batch_results.html

```
<!-- Generate table from prediction results -->
<table class="table">
    <thead>
      <tr>
          <td>Request Timestamp</td>
          <td>Carrier</td>
          <td>Flight Date</td>
          <td>Origin</td>
          <td>Destination</td>
          <td>Distance</td>
          <td>Departure Delay</td>
          <td><span style="color: red;">Predicted Arrival Delay</span></td>
      </tr>
    </thead>
    <tbody>
      {% for item in predictions %}
          <tr>
              <td>{{ item['Timestamp'] }}</td>
              <td>{{ item['Carrier'] }}</td>
              <td>{{ item['FlightDate'] }}</td>
              <td>{{ item['Origin'] }}</td>
              <td>{{ item['Dest'] }}</td>
              <td>{{ item['Distance'] }}</td>
              <td>{{ item['DepDelay'] }}</td>
              <td>
                  <span style="color: red;">
                      {% if item['Prediction'] == 0.0 %}
                          On Time (0-15 Minute Delay)
                      {% elif item['Prediction'] == 1.0 %}
                          Slightly Late (15-60 Minute Delay)
                      {% elif item['Prediction'] == 2.0 %}
                          Very Late (60+ Minute Delay)
                      {% endif %}
                  </span>
              </td>
          </tr>
      {% endfor %}
    </tbody>
</table>
```

Batch Deployment Application - Results Page

What our end result looks like!

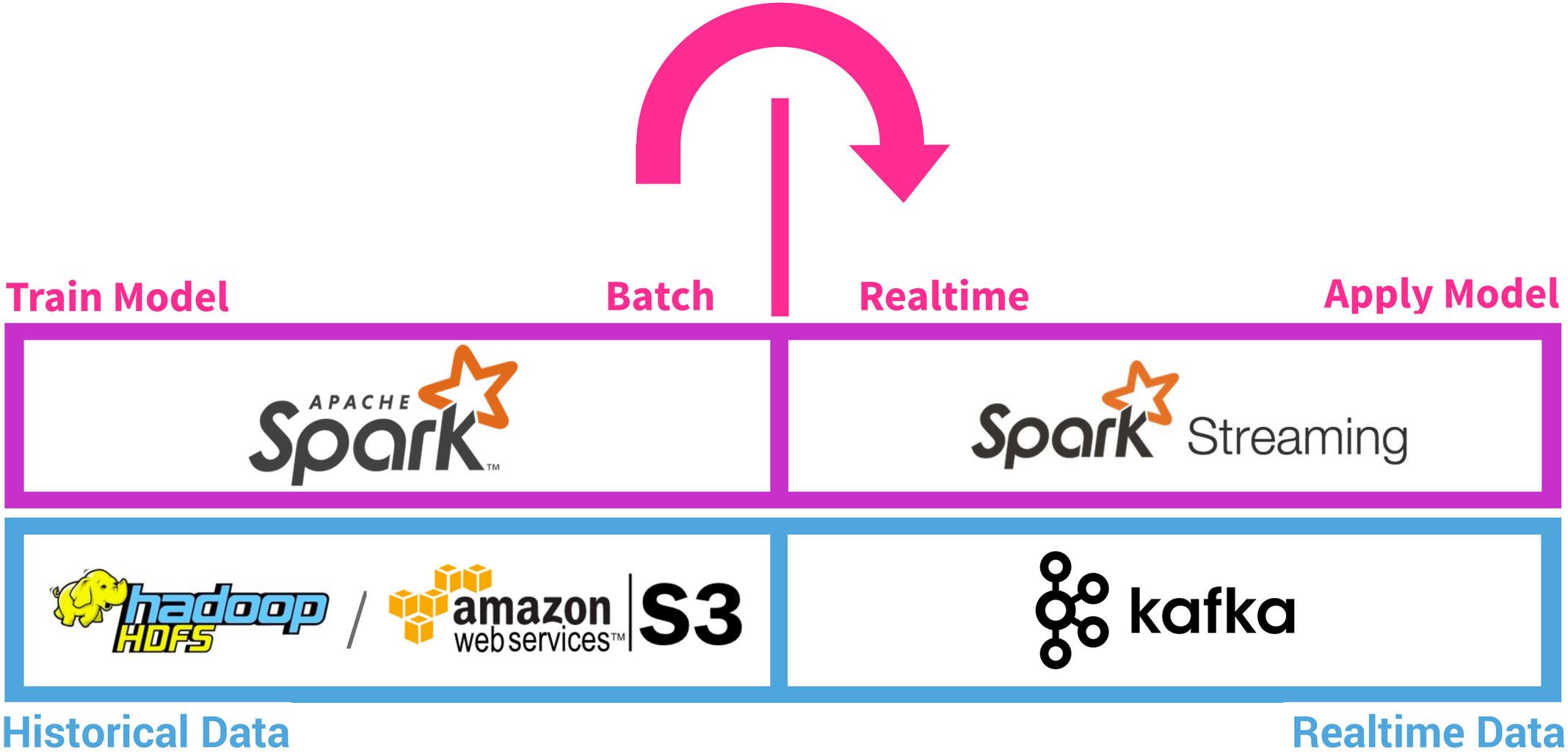


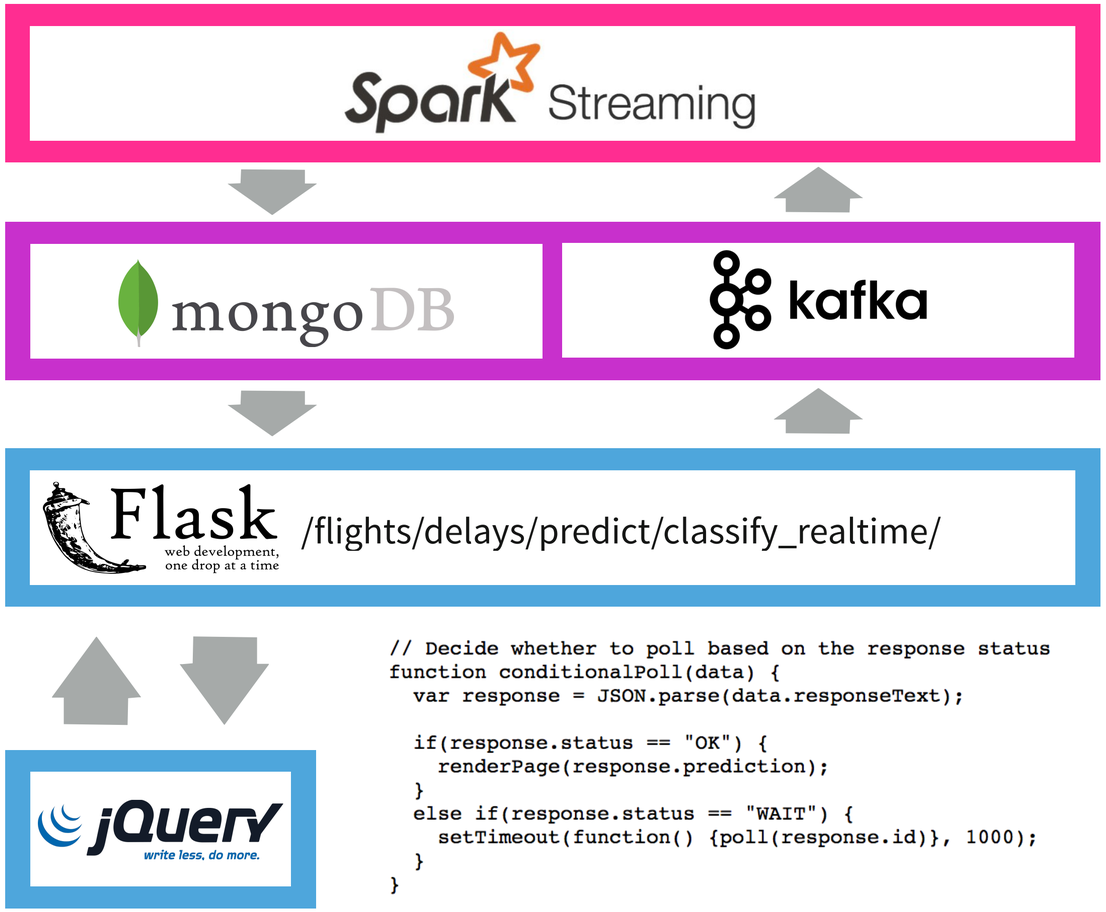
Deploying in Realtime

Next steps for deploying the model in realtime

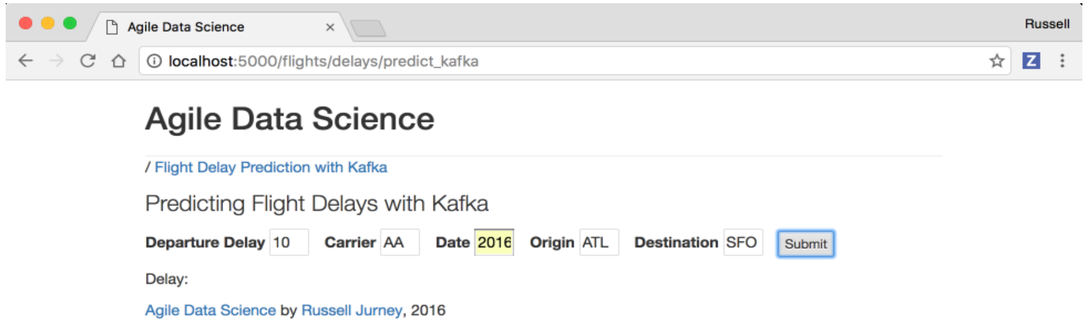
Back End Design

Deep Storage and Spark vs Kafka and Spark Streaming






```
// Decide whether to poll based on the response status
function conditionalPoll(data) {
  var response = JSON.parse(data.responseText);

  if(response.status == "OK") {
    renderPage(response.prediction);
  }
  else if(response.status == "WAIT") {
    setTimeout(function() {poll(response.id)}, 1000);
  }
}
```

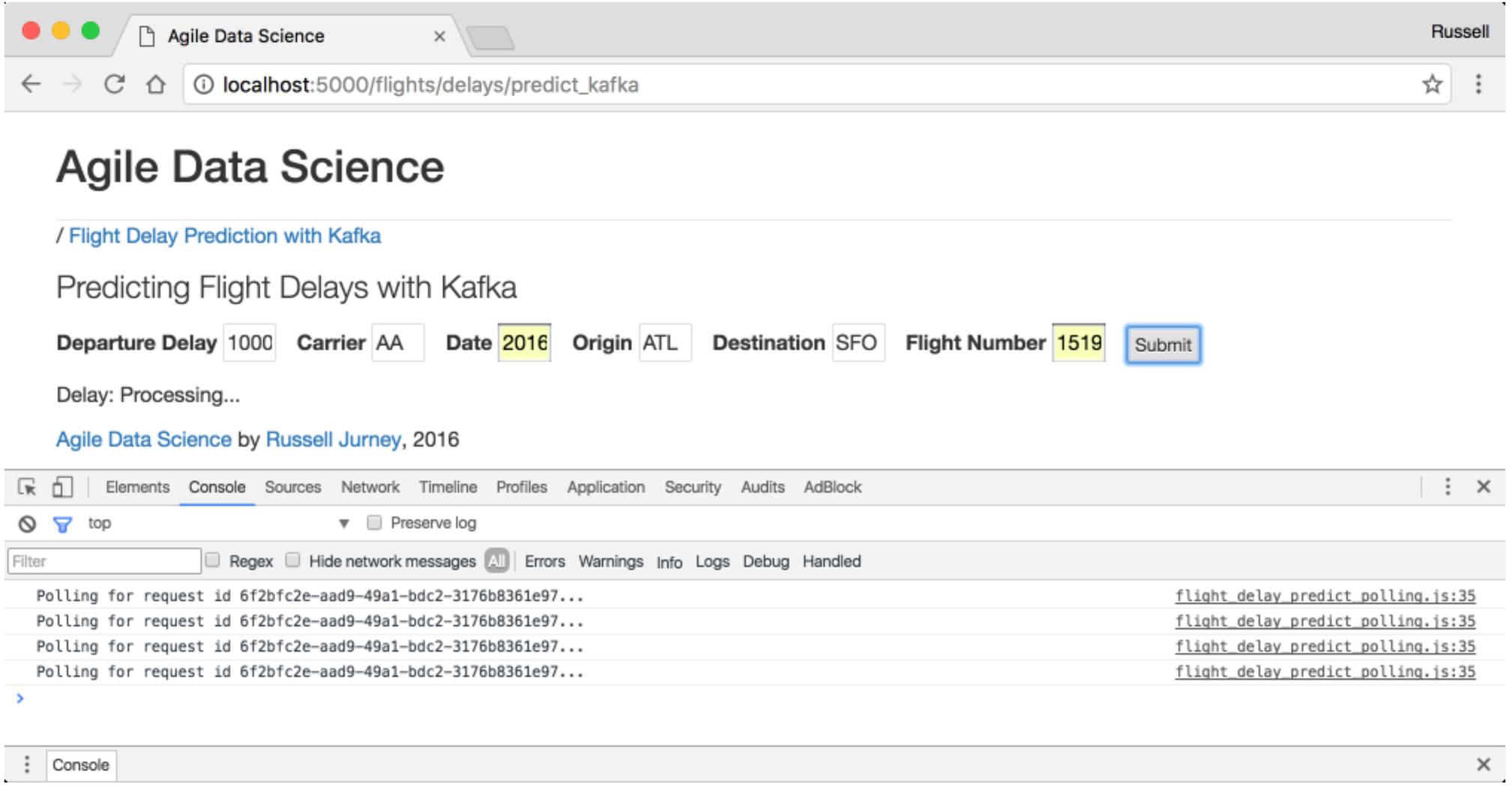


Front End Design

jQuery in the web client submits a form to create the prediction request, and then polls another url every few seconds until the prediction is ready. The request generates a Kafka event, which a Spark Streaming worker processes by applying the model we trained in batch. Having done so, it inserts a record for the prediction in MongoDB, where the Flask app sends it to the web client the next time it polls the server

Realtime User Interface

Where the user submits prediction requests



See ch08/make_predictions_streaming.py

Using PySpark Streaming to deploy our model

```
#!/usr/bin/env python

import sys, os, re
import json
import datetime, iso8601

from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, Row
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

# Save to Mongo
from bson import json_util
import pymongo_spark
pymongo_spark.activate()

def main(base_path):

  APP_NAME = "make_predictions_streaming.py"

  # Process data every 10 seconds
  PERIOD = 10
  BROKERS = 'localhost:9092'
  PREDICTION_TOPIC = 'flight_delay_classification_request'

  try:
    sc and ssc
  except NameError as e:
    import findspark

    # Add the streaming package and initialize
    findspark.add_packages(["org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.0"])
    findspark.init()

    import pyspark
    import pyspark.sql
    import pyspark.streaming

    conf = SparkConf().set("spark.default.parallelism", 1)
    sc = SparkContext(appName="Agile Data Science: PySpark Streaming 'Hello, World!'", conf=conf)
    ssc = StreamingContext(sc, PERIOD)
    spark = pyspark.sql.SparkSession(sc).builder.appName(APP_NAME).getOrCreate()

  #
  # Load all models to be used in making predictions
  #

  # Load the arrival delay bucketizer
  from pyspark.ml.feature import Bucketizer
  arrival_bucketizer_path = "{}/models/arrival_bucketizer_2.0.bin".format(base_path)
  arrival_bucketizer = Bucketizer.load(arrival_bucketizer_path)

  # Load all the string field vectorizer pipelines into a dict
  from pyspark.ml.feature import StringIndexerModel

  string_indexer_models = {}
  for column in ["Carrier", "Origin", "Dest", "Route"]:
    string_indexer_model_path = "{}/models/string_indexer_model_{}.bin".format(
      base_path,
      column
    )
    string_indexer_model = StringIndexerModel.load(string_indexer_model_path)
    string_indexer_models[column] = string_indexer_model

  # Load the numeric vector assembler
  from pyspark.ml.feature import VectorAssembler
  vector_assembler_path = "{}/models/numeric_vector_assembler.bin".format(base_path)
  vector_assembler = VectorAssembler.load(vector_assembler_path)

  # Load the classifier model
  from pyspark.ml.classification import RandomForestClassifier, RandomForestClassificationModel
  random_forest_model_path = "{}/models/spark_random_forest_classifier.flight_delays.5.0.bin".format(
    base_path
  )
  rfc = RandomForestClassificationModel.load(
    random_forest_model_path
  )

  #
  # Process Prediction Requests in Streaming
  #
  stream = KafkaUtils.createDirectStream(
    ssc,
    [PREDICTION_TOPIC],
    {
      "metadata.broker.list": BROKERS,
      "group.id": "0",
    }
  )

  object_stream = stream.map(lambda x: json.loads(x[1]))
  object_stream.pprint()

  row_stream = object_stream.map(
    lambda x: Row(
      FlightDate=iso8601.parse_date(x['FlightDate']),
      Origin=x['Origin'],
      Distance=x['Distance'],
      DayOfMonth=x['DayOfMonth'],
      DayOfYear=x['DayOfYear'],
      UUID=x['UUID'],
      DepDelay=x['DepDelay'],
      DayOfWeek=x['DayOfWeek'],
      FlightNum=x['FlightNum'],
      Dest=x['Dest'],
      Timestamp=iso8601.parse_date(x['Timestamp']),
      Carrier=x['Carrier']
    )
  )
  row_stream.pprint()

  #
  # Create a dataframe from the RDD-based object stream
  #

  def classify_prediction_requests(rdd):

    from pyspark.sql.types import StringType, IntegerType, DoubleType, DateType, TimestampType
    from pyspark.sql.types import StructType, StructField

    prediction_request_schema = StructType([
      StructField("Carrier", StringType(), True),
      StructField("DayOfMonth", IntegerType(), True),
      StructField("DayOfWeek", IntegerType(), True),
      StructField("DayOfYear", IntegerType(), True),
      StructField("DepDelay", DoubleType(), True),
      StructField("Dest", StringType(), True),
      StructField("Distance", DoubleType(), True),
      StructField("FlightDate", DateType(), True),
      StructField("FlightNum", StringType(), True),
      StructField("Origin", StringType(), True),
      StructField("Timestamp", TimestampType(), True),
      StructField("UUID", StringType(), True),
    ])

    prediction_requests_df = spark.createDataFrame(rdd, schema=prediction_request_schema)
    prediction_requests_df.show()

    #
    # Add a Route variable to replace FlightNum
    #

    from pyspark.sql.functions import lit, concat
    prediction_requests_with_route = prediction_requests_df.withColumn(
      'Route',
      concat(
        prediction_requests_df.Origin,
        lit('-'),
        prediction_requests_df.Dest
      )
    )
    prediction_requests_with_route.show(6)

    # Vectorize string fields with the corresponding pipeline for that column
    # Turn category fields into categoric feature vectors, then drop intermediate fields
    for column in ["Carrier", "Origin", "Dest", "Route"]:
      string_indexer_model = string_indexer_models[column]
      prediction_requests_with_route = string_indexer_model.transform(prediction_requests_with_route)

    # Vectorize numeric columns: DepDelay, Distance and index columns
    final_vectorized_features = vector_assembler.transform(prediction_requests_with_route)

    # Inspect the vectors
    final_vectorized_features.show()

    # Drop the individual index columns
    index_columns = ["Carrier_index", "Origin_index", "Dest_index", "Route_index"]
    for column in index_columns:
      final_vectorized_features = final_vectorized_features.drop(column)

    # Inspect the finalized features
    final_vectorized_features.show()

    # Make the prediction
    predictions = rfc.transform(final_vectorized_features)

    # Drop the features vector and prediction metadata to give the original fields
    predictions = predictions.drop("Features_vec")
    final_predictions = predictions.drop("indices").drop("values").drop("rawPrediction").drop("probability")

    # Inspect the output
    final_predictions.show()

    # Store to Mongo
    if final_predictions.count() > 0:
      final_predictions.rdd.map(lambda x: x.asDict()).saveToMongoDB(
        "mongodb://localhost:27017/agile_data_science.flight_delay_classification_response"
      )

  # Do the classification and store to Mongo
  row_stream.foreachRDD(classify_prediction_requests)

  ssc.start()
  ssc.awaitTermination()

if __name__ == "__main__":
  main(sys.argv[1])
```

Next Steps

Next steps for learning more about Agile Data Science 2.0

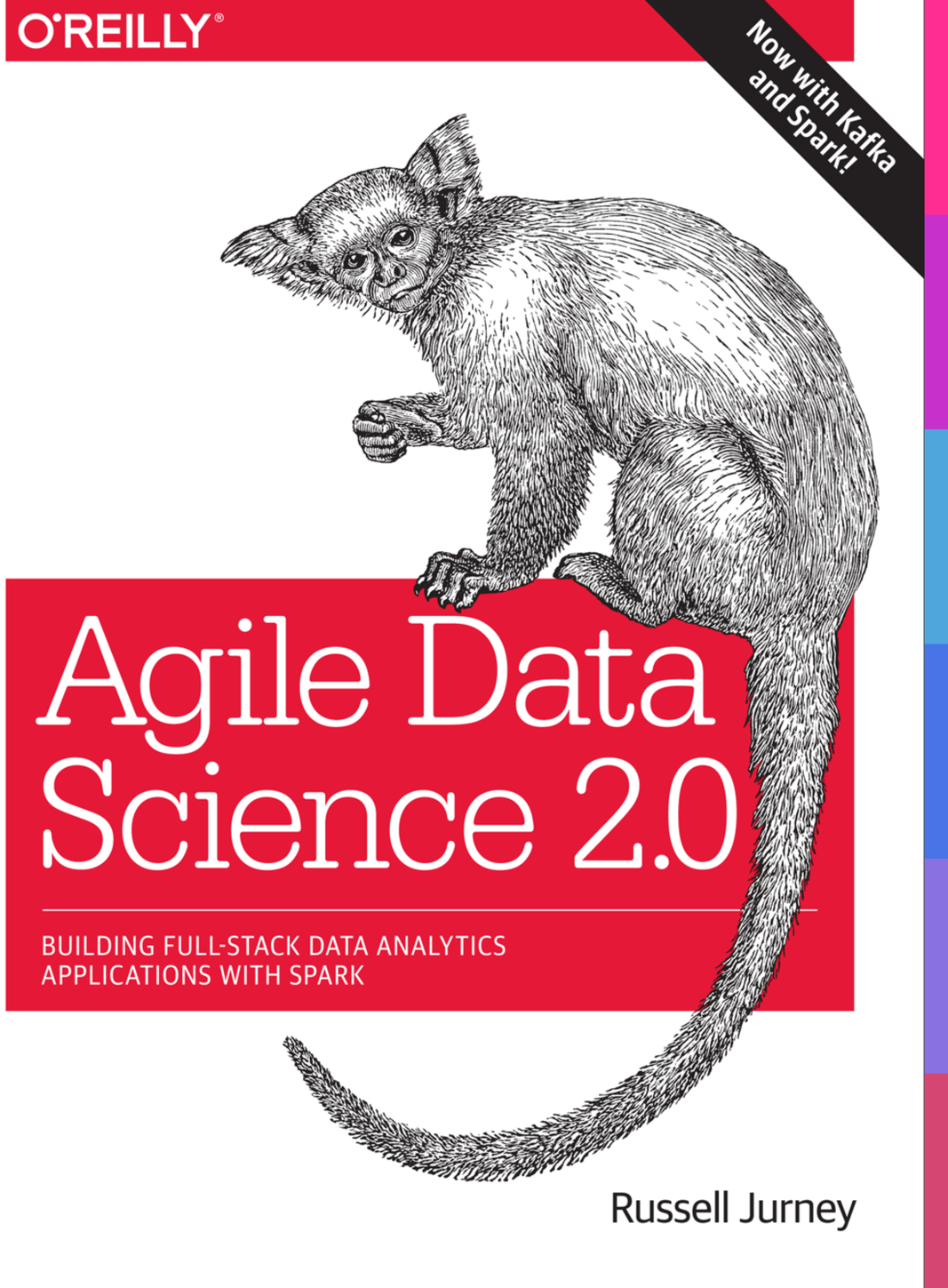
Agile Data Science 2.0

Building Full-Stack Data Analytics Applications with Spark

http://bit.ly/agile_data_science

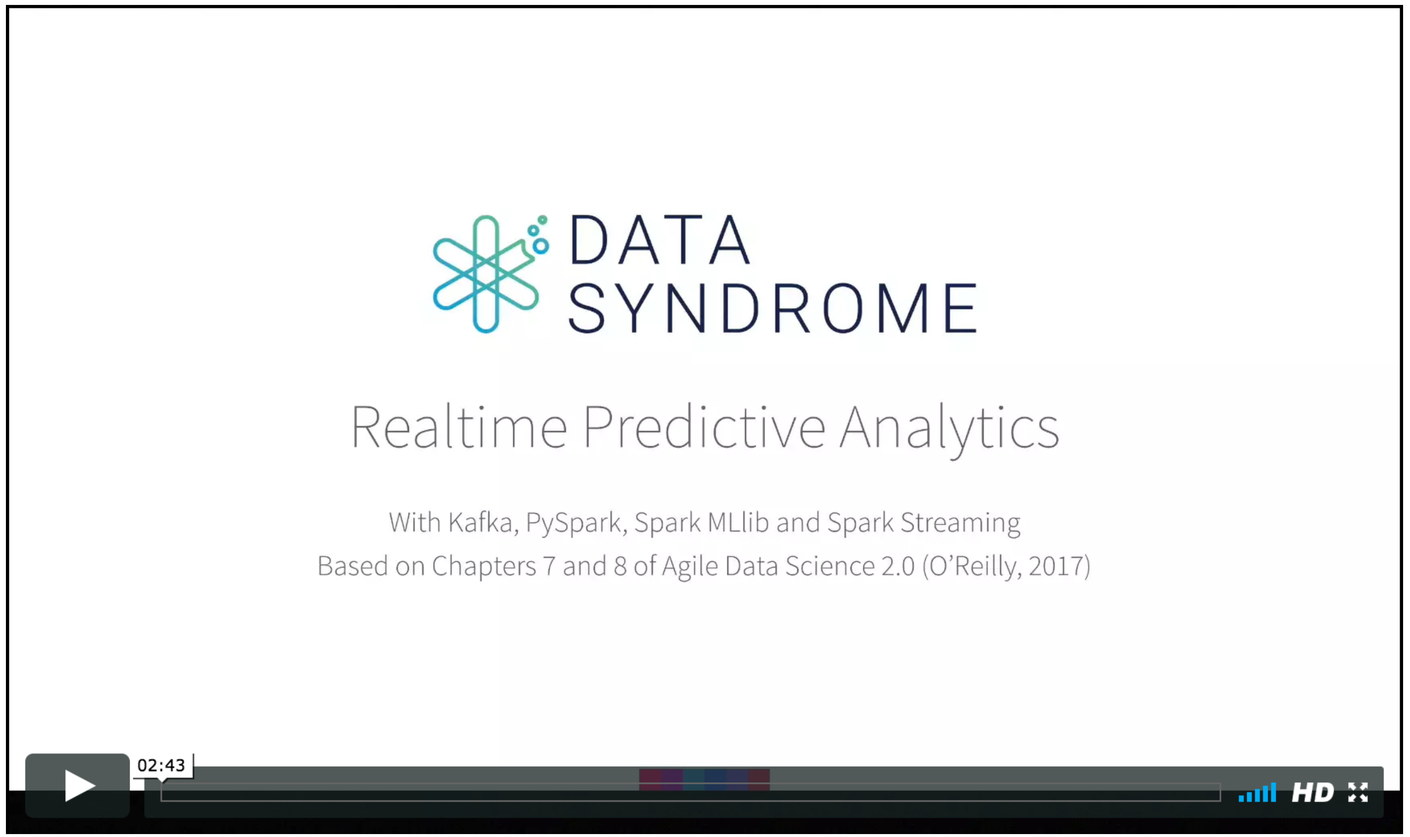
Available Now on O'Reilly Safari: http://bit.ly/agile_data_safari

Code available at: http://github.com/rjurney/Agile_Data_Code_2



Realtime Predictive Analytics

Rapidly learn to build entire predictive systems driven by Kafka, PySpark, Speak Streaming, Spark MLlib and with a web front-end using Python/Flask and JQuery.



Available for purchase at http://datasyndrome.com/video

Data Syndrome

Product Consulting

We build analytics products and systems consisting of big data viz, predictions, recommendations, reports and search.

Corporate Training

We offer training courses for data scientists and engineers and data science teams.

Video Training

We offer video training courses that rapidly acclimate you with a technology and technique.

Russell Jurney
Data Syndrome, LLC
Principal Consultant
Email : rjurney@datasyndrome.com
Web : datasyndrome.com

DATA SYNDROME