

Programming

Databases and Distributed Computing

Daniel Dörr

Faculty of Technology, Bielefeld University

```
332
333
334     if extrapolate is None:
335         extrapolate = self.extrapolate
336     x = np.asarray(x)
337     x_shape, x_ndim = x.shape, x.ndim
338     x = np.ascontiguousarray(x.ravel(), dtype=np
339
340     # With periodic extrapolation we map x to the
341     # [self.t[k], self.t[n]].
342     if extrapolate == 'periodic':
343         n = self.t.size - self.k - 1
344         x = self.t[self.k] + (x - self.t[self.k]) *
345         extrapolate = False
346
347     out = np.empty((len(x), prod(self.c.shape[1:])),
348                   dtype=self.c.dtype)
349     self._ensure_c_contiguous()
350     self._evaluate(x, nu, extrapolate, out)
351     out = out.reshape(x_shape + self.c.shape[1:])
352
353     if self.axis != 0:
354         # transpose to move the calculated values to the
355         # axis
356         l = list(range(out.ndim))
357         l = l[:x_ndim:x_ndim+self.axis] + l[:x_ndim] +
358             l[x_ndim:]
359         out = out.transpose(l)
360
361     return out
362
363 def _evaluate(self, xp, nu, extrapolate, out):
364     _bspl.evaluate_spline(self.t, self.c, reshape(self.c,
365             self.k, xp, nu, extrapolate, out)
366
367 def _ensure_c_contiguous(self):
368     """
369     c and t may be modified by the user. The Cython code
370     ensures that they are c contiguous.
371     """
372     self.c = np.ascontiguousarray(self.c)
373     self.t = np.ascontiguousarray(self.t)
```

Recap

Machine Learning

Unsupervised Learning

- Dimensionality reduction
- Clustering

Supervised Learning

- Classification
- Regression

The Estimator API

Estimators of the Scikit-Learn package share a common API.

Use of estimators:

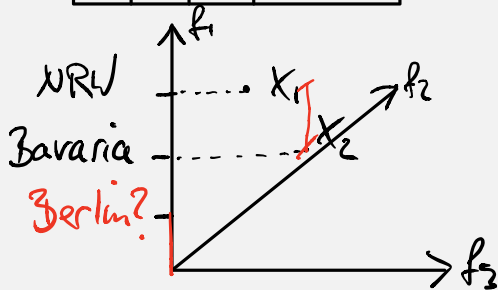
- ❖ Choose model (Estimator)
- ❖ Choose model hyperparameters
- ❖ Instantiate model with hyperparameters
- ❖ Call `fit()` to train the model on a given data set
- ❖ Apply model to new data:
 - ❖ Supervised learning: call `predict()`
 - ❖ Unsupervised learning: call `transform()` or `predict()` (depending on the estimator)

Feature representation

features ← f_1, f_2, f_3

Samples ↑ x_1, x_2, \dots

x_1	NRW		
x_2	Bavaria		
\vdots			...



Categorical features
(e.g. federal states)

NRW - Bavaria = Berlin? No!

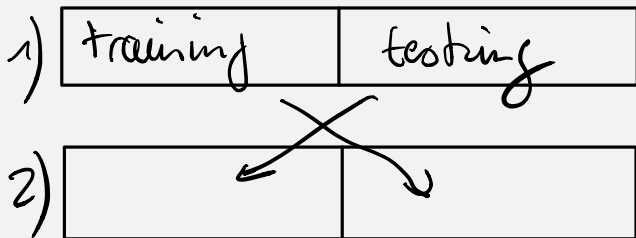
Solution:

f_1 f_2 f_3

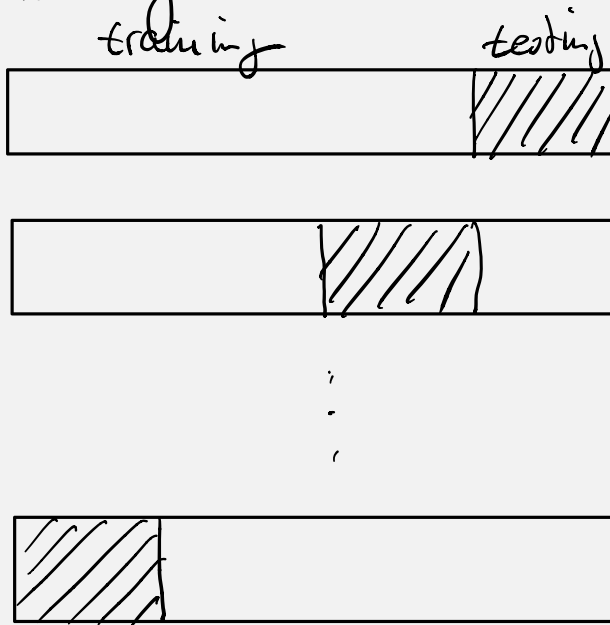
x_1	1	0	0	0	...		
x_2	0	1	0	0	..		
					...		
	NRW	Bavaria	Berlin	...			

Cross-validation

Data set:



More general



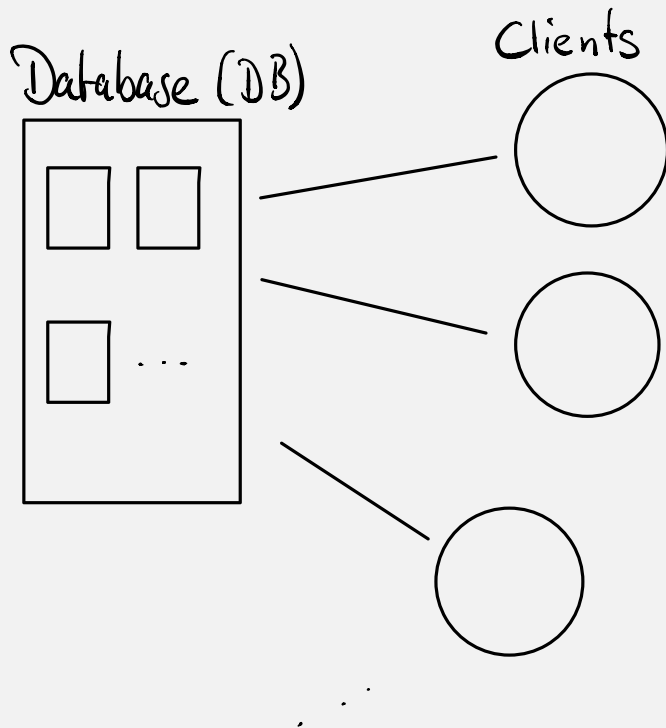
Summarize
evaluation:
- mean
- min
- whisker
- ...

Databases

**Distributed
Tabular Data
Processing**

**Distributed
Machine
Learning**

Databases Overview



Advantages

- Share data among (many) clients
- Save transactions
 - read
 - create / delete
 - update
- High performance

Databases Overview (contd.)

SQL databases

- ... are relational databases
- developed in the 1970s
- general

noSQL databases

- ... are more purpose-specific:
- Key-value
 - Graph
 - Document-oriented
 - Object-oriented

Relational database

- Database: collection of tables
- Described by a schema
 - tables
 - column names & properties
 - primary & secondary keys
 - relationship between tables
- prominent databases:
 - open source: MySQL, PostgreSQL
 - commercial: Oracle, IBM DB2

SQL (Structured Query Language)

Simple language to query and manipulate relational databases

MongoDB

- ❖ Document-oriented database
- ❖ Each DB entry corresponds to a JSON document
- ❖ Popular in web-based applications
- ❖ “Community” edition is open-source



MongoDB

Download MongoDB (<https://www.mongodb.com/try/download/community>), then install it and start the database daemon. Then import the `books.json` dataset, by opening a terminal window and executing the following (adapted) lines of code:

```
/path/to/mongodb/bin/mongoimport -d programming_course  
/path/to/course_material_08/books.json
```

Also, install `pymongo` in Anaconda and restart the Jupyter Notebook server. Then connect to the server by instantiating a `MongoClient` object and connecting to the database `programming_course`.

```
In [1]: import pymongo  
client = pymongo.MongoClient('localhost', 27017)  
  
db = client.programming_course  
db.list_collection_names()
```

```
Out[1]: ['books']
```

```
In [2]: books = db.books
books.find_one()
```

```
Out[2]: {'_id': 4,
'title': 'Flex 3 in Action',
'isbn': '1933988746',
'pageCount': 576,
'publishedDate': datetime.datetime(2009, 2, 2, 8, 0),
'thumbnailUrl': 'https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/ahmed.jpg',
'longDescription': "New web applications require engaging user-friendly interfaces and the cooler, the better. With Flex 3, web developers at any skill level can create high-quality, effective, and interactive Rich Internet Applications (RIAs) quickly and easily. Flex removes the complexity barrier from RIA development by offering sophisticated tools and a straightforward programming language so you can focus on what you want to do instead of how to do it. And now that the major components of Flex are free and open-source, the cost barrier is gone, as well! Flex 3 in Action is an easy-to-follow, hands-on Flex tutorial. Chock-full of examples, this book goes beyond feature coverage and helps you put Flex to work in real day-to-day tasks. You'll quickly master the Flex API and learn to apply the techniques that make your Flex applications stand out from the crowd. Interesting themes, styles, and skins It's in there. Working with databases You got it. Interactive forms and validation You bet. Charting techniques to help you visualize data Bam! The expert authors of Flex 3 in Action have one goal to help you get down to business with Flex 3. Fast. Many Flex books are overwhelming to new users focusing on the complexities of the language and the super-specialized subjects in the Flex ecosystem; Flex 3 in Action filters out the noise and dives into the core topics you need every day. Using numerous easy-to-understand examples, Flex 3 in Action gives you a strong foundation that you can build on as the complexity of your projects increases.",
'status': 'PUBLISH',
'authors': ['Tariq Ahmed with Jon Hirschi', 'Faisal Abid'],
'categories': ['Internet']}
```

Searching for one or more specific books can be accomplished with the same method, but this time passing on search criteria:

```
In [3]: book = books.find_one({'authors': 'Jimmy Bogard'})
print(f'Book: {" , ".join(book["authors"])}: {book["title"]}, {book["publishedDate"]}')
```

```
Book: Jeffrey Palermo, Ben Scheirman, , Jimmy Bogard: ASP.NET MVC in Action, 2
009-09-01 07:00:00
```

```
In [5]: print('Titles of Bogards\'s books in alphabetic order:')
for book in books.find({'authors': 'Jimmy Bogard'}):
    print(book['title'])
```

```
Titles of Bogards's books in alphabetic order:
ASP.NET MVC 2 in Action
ASP.NET MVC 4 in Action
ASP.NET MVC in Action
```

```
In [4]: all_books_by_bogard = list(books.find({'authors': 'Jimmy Bogard'}))
print(f'The collection contains {len(all_books_by_bogard)} books by Jimmy Bogard')
```

The collection contains 3 books by Jimmy Bogard

If you are only interested in counting, then use:

```
In [6]: books.count_documents({'authors': 'Jimmy Bogard'})
```

Out[6]: 3

Range Queries

The keywords `$lt` (lesser than) and `$gt` (greater than) enable range queries:

```
In [51]: import datetime
          d = datetime.datetime(2012, 11, 12, 12)
          books.count_documents({'publishedDate': {"$gt": d}})
```

```
Out[51]: 53
```


Indexing

```
In [8]: books.create_index([('authors', pymongo.ASCENDING)], unique=False)
sorted(list(books.index_information()))
```

```
Out[8]: ['_id_', 'authors_1']
```

```
In [9]: books.find_one({'authors': {'$gt': 'G'}})
```

```
Out[9]: {'_id': 643,
         'title': 'SonarQube in Action',
         'isbn': '1617290955',
         'pageCount': 0,
         'publishedDate': datetime.datetime(2013, 10, 30, 7, 0),
         'thumbnailUrl': 'https://s3.amazonaws.com/AKIAJJC5RLADLUMVRPFDQ.book-thumb-images/papapetrou.jpg',
         'status': 'PUBLISH',
         'authors': ['G. Ann Campbell', 'Patroklos P. Papapetrou'],
         'categories': []}
```

Quiz

❖ *True or false?*

- ❖ Documents can't reference other documents
- ❖ Documents can contain nested structures
- ❖ Indexes allow fast access when searching by the indexed field

❖ What is the result of the following queries?

- ❖ `books.count_documents({'authors': 'David A. Black', 'categories': 'Programming'})`
- ❖ `books.find({'categories': 'Python'})`
- ❖ `books.find_one({'categories': 'Python'})`

Quiz

❖ True or false?

- ❖ Documents can't reference other documents false
- ❖ Documents can contain nested structures true
- ❖ Indexes allow fast access when searching by the indexed field true

❖ What is the result of the following queries?

- ❖ `books.count_documents({'authors': 'David A. Black', 'categories': 'Programming'})`
Number of books of David A. Black in category Programming
- ❖ `books.find({'categories': 'Python'})`
All books in the category Python
- ❖ `books.find_one({'categories': 'Python'})`
One book from the category Python

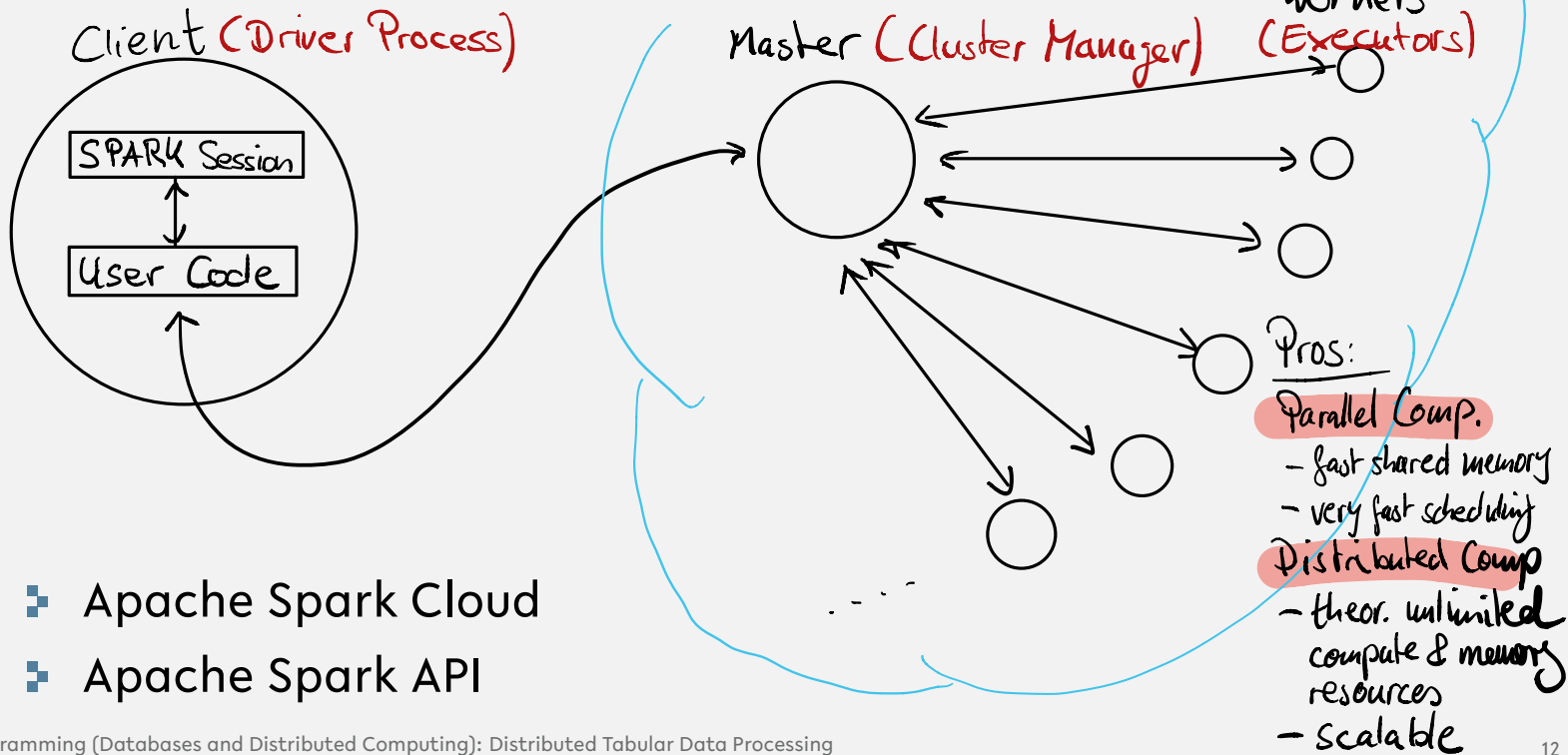
Databases

**Distributed
Tabular Data
Processing**

**Distributed
Machine
Learning**

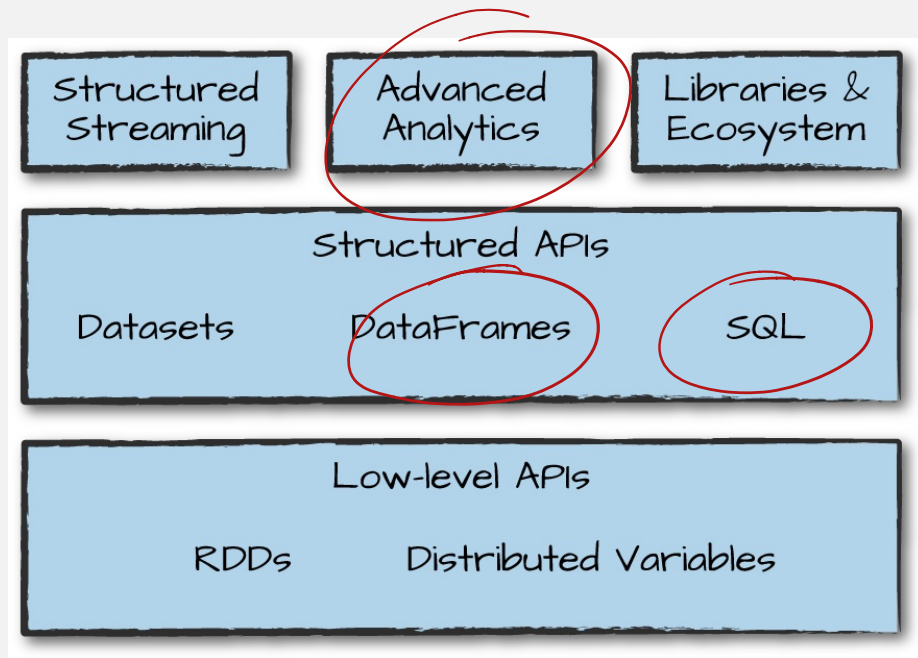
Distributed Computing

- ❖ Distributed computing \neq parallel computing



- ❖ Apache Spark Cloud
- ❖ Apache Spark API

Apache Spark API

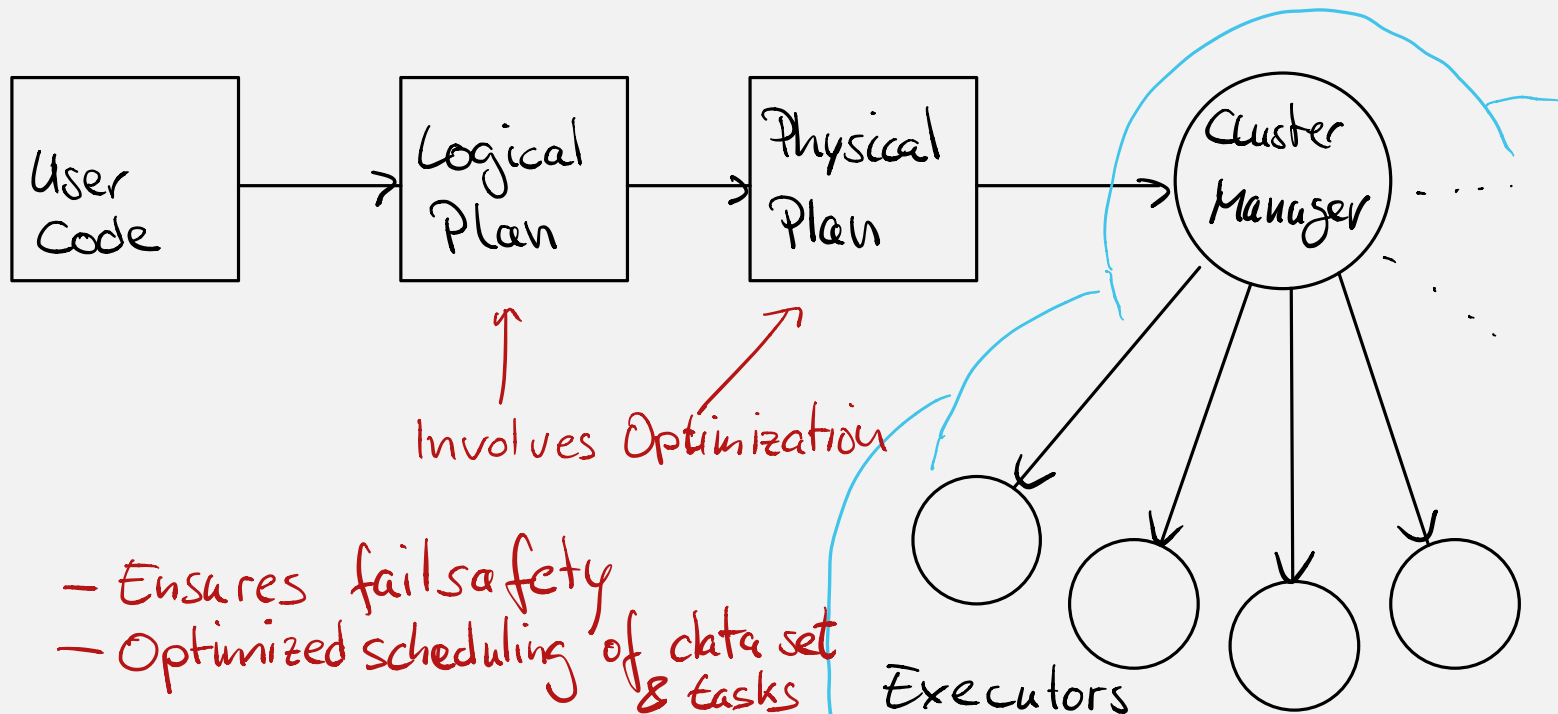


source: Bill Chambers, Matei Zaharia, Spark: The Definitive Guide. O'Reilly Media (2018)

API available in

- ❖ Scala
- ❖ Python
- ❖ R
- ❖ SQL

Spark Computation Planning Process



Spark DataFrame

- ❖ NOT the same as Pandas' DataFrame
- ❖ Completely immutable
- ❖ Abstraction of low-level distributed computing data structure

PySpark

Before you can get started:

1. Ensure that you have JRE 1.8 installed
(<https://www.oracle.com/java/technologies/javase-jre8-downloads.html>
(<https://www.oracle.com/java/technologies/javase-jre8-downloads.html>))
2. Set up environment variables (<https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>
(<https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>)) in
`/path/to/anaconda/etc/conda/activate.d/env_vars{.sh|.bat}`.
 - A. Set `JAVA_HOME` to `/path/to/java/JRE`
 - B. Set `PYSPARK_PYTHON` to `python3.7` (or `python3.8` if you are running the latest version)
3. Install the `pyspark` package (via Anaconda)
4. Start Jupyter Notebook (via Anaconda)

Distributed Tabular Data Processing

The first step in using PySpark is the creation of a session. Specifying `local[*]` as master indicates that all computations will be performed locally, i.e., not submitted to an external cluster node.

```
In [52]: from pyspark.sql import SparkSession  
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Loading a data set into a DataFrame object

Data source: <https://github.com/databricks/Spark-The-Definitive-Guide>
(<https://github.com/databricks/Spark-The-Definitive-Guide>)

```
In [11]: df = spark.read.csv('course_material_08/2015-summary.csv', header=True)
print(type(df))
df.printSchema()
```

```
<class 'pyspark.sql.dataframe.DataFrame'>
root
 |-- DEST_COUNTRY_NAME: string (nullable = true)
 |-- ORIGIN_COUNTRY_NAME: string (nullable = true)
 |-- count: string (nullable = true)
```

By default, all columns are read as strings, unless you ask spark to infer the column types from the data by setting `inferSchema=True` :

```
In [13]: df.show(10)
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344
Egypt	United States	15
United States	India	62
United States	Singapore	1
United States	Grenada	62
Costa Rica	United States	588
Senegal	United States	40
Moldova	United States	1

only showing top 10 rows

Columns are no Series, and no fancy indexes, but *expressions*!

```
In [14]: df['count']
```

```
Out[14]: Column<b'count'>
```

```
In [15]: df.select('count').show(10)
```

```
+-----+
|count|
+-----+
|    15|
|     1|
|   344|
|    15|
|    62|
|     1|
|    62|
|   588|
|    40|
|     1|
+-----+
```

only showing top 10 rows

Expressions

```
In [16]: df['count'] == 1
```

```
Out[16]: Column<b'(count = 1)'\>
```

With expressions, we can e.g. select rows with constraints that are specified by the expression:

```
In [17]: df.where(df['count'] == 1).show(10)
```

```
+-----+-----+-----+
|  DEST_COUNTRY_NAME |ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|      United States |          Croatia   |    1|
|      United States |          Singapore |    1|
|           Moldova |    United States  |    1|
|           Malta    |    United States  |    1|
|      United States |          Gibraltar |    1|
|Saint Vincent and...|    United States  |    1|
|           Suriname |    United States  |    1|
|      United States |          Cyprus    |    1|
|      Burkina Faso  |    United States  |    1|
|           Djibouti |    United States  |    1|
+-----+-----+-----+
```

only showing top 10 rows

Alternative access of columns:

```
In [18]: # does not work (because count() is a method of DataFrame):  
# df.count == 1  
df.where(df.DEST_COUNTRY_NAME == 'United States').show(10)
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344
United States	India	62
United States	Singapore	1
United States	Grenada	62
United States	Sint Maarten	325
United States	Marshall Islands	39
United States	Paraguay	6
United States	Gibraltar	1

only showing top 10 rows

Combining expressions with bitwise operators, i.e., & (and), | (or), ~ (negation):

```
In [19]: expr1 = (df['count'] == 1) & (df['ORIGIN_COUNTRY_NAME'] > 'U')
print(expr1)
df.where(expr1).show(3)

expr2 = (df['count'] == 1) | ~ (df['ORIGIN_COUNTRY_NAME'] > 'U')
print(expr2)
df.where(expr2).show(3)
# alternatively: df.filter(expr).show()
```

```
Column<b'((count = 1) AND (ORIGIN_COUNTRY_NAME > U))'>
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
Moldova	United States	1
Malta	United States	1
Saint Vincent and...	United States	1

only showing top 3 rows

```
Column<b'((count = 1) OR (NOT (ORIGIN_COUNTRY_NAME > U)))'>
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344

only showing top 3 rows

Select

`select` allows to choose one or more columns or expressions thereof from a DataFrame:

```
In [20]: df.select('DEST_COUNTRY_NAME').show(3)
```

```
+-----+
|DEST_COUNTRY_NAME|
+-----+
|    United States|
|    United States|
|    United States|
+-----+
```

only showing top 3 rows

```
In [21]: df.select('DEST_COUNTRY_NAME', 'count').show(3)
```

```
+-----+-----+
|DEST_COUNTRY_NAME|count|
+-----+-----+
|    United States|    15|
|    United States|     1|
|    United States|   344|
+-----+-----+
```

only showing top 3 rows

```
In [22]: df.select(df['DEST_COUNTRY_NAME'] == df['ORIGIN_COUNTRY_NAME']).show(3)
```

```
+-----+
| (DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) |
+-----+
|                                           false|
|                                           false|
|                                           false|
+-----+
```

only showing top 3 rows

Explicit way of creating the same expression:

```
In [23]: from pyspark.sql.functions import expr
df.select(expr('DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME')).show(3)
```

```
+-----+
| (DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) |
+-----+
|                                           false |
|                                           false |
|                                           false |
+-----+
```

only showing top 3 rows

SelectExpr

`selectExpr` is a convenience function to do the same:

```
In [24]: # same as: df.select(expr('DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME')).show(3)
df.selectExpr('DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME').show(3)
```

```
+-----+
| (DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) |
+-----+
|                                           false |
|                                           false |
|                                           false |
+-----+
```

only showing top 3 rows

Creating tables by combining old and new columns:

```
In [25]: df.selectExpr('*', '(DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME)').show(3)
```

```
+-----+-----+-----+-----+
-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|(DEST_COUNTRY_NAME = ORIGIN_COUNTR
RY_NAME)|
+-----+-----+-----+-----+
-----+
|    United States|          Romania|    15|
false|
|    United States|          Croatia|     1|
false|
|    United States|          Ireland|   344|
false|
+-----+-----+-----+-----+
-----+
only showing top 3 rows
```

```
In [26]: df.selectExpr('*', '(DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) as WITHIN_COUNTRY').  
show(3)
```

```
+-----+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|WITHIN_COUNTRY|  
+-----+-----+-----+-----+  
|    United States|           Romania|    15|           false|  
|    United States|           Croatia|     1|           false|  
|    United States|           Ireland|   344|           false|  
+-----+-----+-----+-----+
```

only showing top 3 rows

Aggregators

```
In [27]: df.selectExpr('avg(count) as AVG', # same as 'mean(count)',  
                        'stddev(count) as STDDEV',  
                        'first(count) as FIRST',  
                        'max(count) as MAX',  
                        'count(distinct(DEST_COUNTRY_NAME)) as COUNT_DISTINCT').show(10)
```

AVG	STDDEV	FIRST	MAX	COUNT_DISTINCT
1770.765625	23126.516918551926	41	370002	132

Sorting

```
In [28]: df.sort('count', ascending=False).show(3)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME| count|
+-----+-----+-----+
|      United States|      United States|370002|
|      United States|          Canada|   8483|
|          Canada|      United States|   8399|
+-----+-----+-----+
```

only showing top 3 rows

```
In [29]: df.sort(df['count'].desc()).show(3)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME| count|
+-----+-----+-----+
|      United States|      United States|370002|
|      United States|          Canada|   8483|
|          Canada|      United States|   8399|
+-----+-----+-----+
```

only showing top 3 rows

In [30]: *# return type of where query is again a DataFrame*

```
print(df.sort(df['count'].desc()).where(  
    df['DEST_COUNTRY_NAME'] > 'U'))
```

i.e., multiple queries can be combined

```
df.sort(df['count'].desc()).where(  
    df['DEST_COUNTRY_NAME'] > 'U').where(df['count'] > 8000).show()
```

DataFrame[DEST_COUNTRY_NAME: string, ORIGIN_COUNTRY_NAME: string, count: int]

```
+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME| count|  
+-----+-----+-----+  
|    United States|    United States|370002|  
|    United States|           Canada|  8483|  
+-----+-----+-----+
```

Limit result to top x entries:

```
In [31]: df.sort(df['count'].desc()).limit(3).where(
          df['DEST_COUNTRY_NAME'] > 'U').show()
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME| count|
+-----+-----+-----+
|      United States|      United States|370002|
|      United States|           Canada|  8483|
+-----+-----+-----+
```

SQL

The same commands that can be performed on the DataFrame object can also be performed with SQL queries:

```
In [32]: # make DataFrame available as SQL table under specific name
df.createOrReplaceTempView('flight_data')

# DataFrame query:
# df.select('DEST_COUNTRY_NAME', 'count').show(n=10)

# analog SQL query:
spark.sql('SELECT DEST_COUNTRY_NAME, count FROM flight_data').show(3)
```

```
+-----+-----+
|DEST_COUNTRY_NAME|count|
+-----+-----+
|    United States|    15|
|    United States|     1|
|    United States|   344|
+-----+-----+
only showing top 3 rows
```

In [33]:

```
# DataFrame query:
#df.select('DEST_COUNTRY_NAME', 'count').where(
#    df['ORIGIN_COUNTRY_NAME'] == 'The Bahamas').show()

# analog SQL query:
spark.sql('SELECT DEST_COUNTRY_NAME, count FROM flight_data WHERE ORIGIN_COUNTRY_NAME="The Bahamas"').show()
```

```
+-----+-----+
|DEST_COUNTRY_NAME|count|
+-----+-----+
|    United States|   986|
+-----+-----+
```

```
In [34]: # DataFrame query:abs
#df.groupby('DEST_COUNTRY_NAME').sum('count').select(
#      'DEST_COUNTRY_NAME', 'sum(count)').show(n=10)

# analog SQL query:
spark.sql('SELECT DEST_COUNTRY_NAME, sum(count) FROM flight_data GROUP BY DEST_COUNTRY_NAME').show(n=10)
```

```
+-----+-----+
|DEST_COUNTRY_NAME|sum(count)|
+-----+-----+
|           Anguilla|         41|
|           Russia|        176|
|          Paraguay|         60|
|           Senegal|         40|
|           Sweden|        118|
|          Kiribati|         26|
|           Guyana|         64|
|    Philippines|        134|
|          Djibouti|          1|
|          Malaysia|          2|
+-----+-----+
```

only showing top 10 rows

Quiz

❖ *True or false?*

- ❖ Spark distributes data automatically across the workers
- ❖ Spark optimizes the execution plan
- ❖ DataFrames are completely immutable
- ❖ The columns in DataFrames don't have a fixed data type

❖ Which method can be used to return a column?

- ❖ `df.select("DEST_COUNTRY_NAME")`
- ❖ `df.DEST_COUNTRY_NAME`
- ❖ `df["DEST_COUNTRY_NAME"]`
- ❖ `df("DEST_COUNTRY_NAME")`

Quiz

❖ True or false?

- ❖ Spark distributes data automatically across the workers true
- ❖ Spark optimizes the execution plan true
- ❖ DataFrames are completely immutable true
- ❖ The columns in DataFrames don't have a fixed data type false

❖ Which method can be used to return a column?

- ❖ `df.select("DEST_COUNTRY_NAME")` ✓
- ❖ `df.DEST_COUNTRY_NAME`
- ❖ `df["DEST_COUNTRY_NAME"]`
- ❖ `df("DEST_COUNTRY_NAME")`

Quiz 2

❖ *True or false?*

- ❖ The datatype of each column has to be predefined
- ❖ Queries always have the form
FROM <table> SELECT <columns> WHERE <conditions>
- ❖ Every record needs a unique primary key

❖ What is the outcome of the following queries?

- ❖ `SELECT SUM(count)FROM flight_data GROUP BY
ORIGIN_COUNTRY_NAME`
- ❖ `SELECT * FROM flight_data WHERE DEST_COUNTRY_NAME = "United
States"ORDER BY count`

Quiz 2

❖ True or false?

- ❖ The datatype of each column has to be predefined true
- ❖ Queries always have the form
FROM <table> SELECT <columns> WHERE <conditions> false
- ❖ Every record needs a unique primary key true

❖ What is the outcome of the following queries?

- ❖ `SELECT SUM(count)FROM flight_data GROUP BY ORIGIN_COUNTRY_NAME`
Sum of all flights per origin country
- ❖ `SELECT * FROM flight_data WHERE DEST_COUNTRY_NAME = "United States"ORDER BY count`
All flights with destination United States sorted by their number

Databases

**Distributed
Tabular Data
Processing**

**Distributed
Machine
Learning**

Distributed Machine Learning

Description of dataset

Abalone are marine snails. A image of such a snail can be found [here](https://en.wikipedia.org/wiki/Abalone#/media/File:LivingAbalone.JPG) (<https://en.wikipedia.org/wiki/Abalone#/media/File:LivingAbalone.JPG>). To determine the age of them, marine biologist have to cut the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. The number of rings directly correlates with the age of the snail. Other measurements, which are easier to obtain, are used to predict the age. ([Source](http://mlr.cs.umass.edu/ml/datasets/Abalone) (<http://mlr.cs.umass.edu/ml/datasets/Abalone>), with small changes)

Name / Data Type / Measurement Unit / Description

Sex / nominal / -- / M, F, and I (infant)

Length / continuous / mm / Longest shell measurement

Height / continuous / mm / with meat in shell

Whole weight / continuous / grams / whole abalone

Shell weight / continuous / grams / after being dried

Rings / integer / -- / +1.5 gives the age in years

```
In [35]: df = spark.read.csv('course_material_08/abalone.csv', header=True, inferSchema=True)
df.printSchema()
df.drop("sex").describe().toPandas().set_index("summary").transpose()
```

```
root
|-- sex: string (nullable = true)
|-- length: double (nullable = true)
|-- height: double (nullable = true)
|-- whole-weight: double (nullable = true)
|-- shell-weight: double (nullable = true)
|-- rings: integer (nullable = true)
```

Out[35]:

	summary	count	mean	stddev	min	max
length		4177	0.5239920995930099	0.12009291256479936	0.075	0.815
height		4177	0.1395163993296614	0.04182705660725731	0.0	1.13
whole-weight		4177	0.82874215944458	0.49038901823099795	0.002	2.8255
shell-weight		4177	0.23883085946851795	0.13920266952238622	0.0015	1.005
rings		4177	9.933684462532918	3.2241690320681315	1	29

Create pairwise scatterplots for random subsample

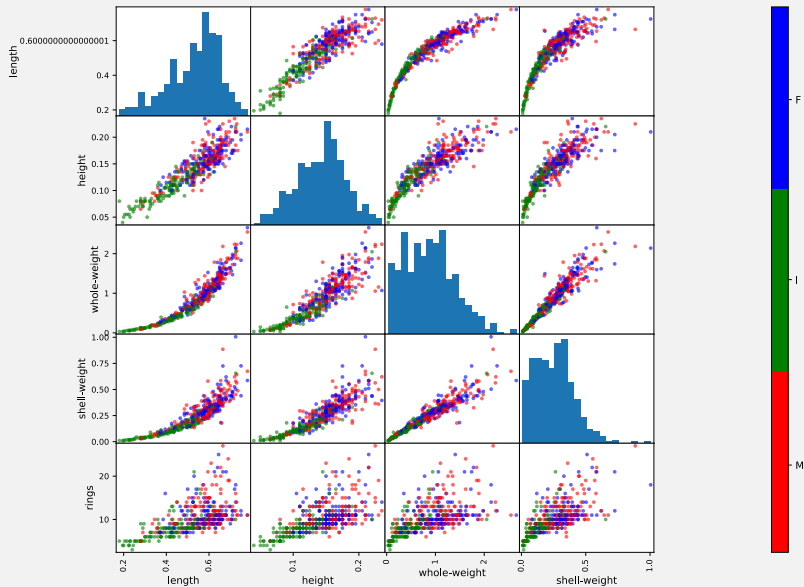
```
In [58]: from matplotlib import colors, colorbar
sampled_df = df.sample(False, 0.1, 42).toPandas()
rgb = {'M': 'r', 'I': 'g', 'F': 'b'}

plt.figure(figsize=(12, 10))
ax = plt.subplot(1, 40, 1)
pd.plotting.scatter_matrix(
    sampled_df.drop(columns = ['sex']),
    c = sampled_df['sex'].apply(lambda x: rgb[x]),
    ax = ax, marker='o', hist_kwds={'bins': 20}, s=14, alpha=.6)

ax = plt.subplot(1, 40, 40)
# color bar
color_key, color_val = zip(*rgb.items())
cmap = colors.ListedColormap(color_val)
cb = colorbar.ColorbarBase(ax, cmap, ticks=(0.16, 0.5, 0.83))
cb.set_ticklabels(color_key)
```

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10: UserWarning: To output multiple subplots, the figure containing the passed axes is being cleared

```
# Remove the CWD from sys.path while we load stuff.
```



Convert sex variable from string to numerical variable

```
In [37]: sexIndexer = StringIndexer(inputCol="sex", outputCol="sex_index")
indexer_model = sexIndexer.fit(df)
print("Indices for 'sex' variable", indexer_model.labels)
indexed_df = indexer_model.transform(df)
indexed_df.limit(5).show()
```

Indices for 'sex' variable ['M', 'I', 'F']

sex	length	height	whole-weight	shell-weight	rings	sex_index
M	0.455	0.095	0.514	0.15	15	0.0
M	0.35	0.09	0.2255	0.07	7	0.0
F	0.53	0.135	0.677	0.21	9	2.0
M	0.44	0.125	0.516	0.155	10	0.0
I	0.33	0.08	0.205	0.055	7	1.0

```
In [38]: len(indexer_model.labels)
```

Out[38]: 3

Use OneHot encoding for sex numerical sex variable

```
In [59]: sexOneHot = OneHotEncoderEstimator(inputCols=["sex_index"],
                                             outputCols=["sex_vec"])
encoded_df = sexOneHot.fit(indexed_df).transform(indexed_df)
encoded_df.limit(3).toPandas()
```

Out[59]:

	sex	length	height	whole-weight	shell-weight	rings	sex_index	sex_vec
0	M	0.455	0.095	0.5140	0.15	15	0.0	(1.0,0.0)
1	M	0.350	0.090	0.2255	0.07	7	0.0	(1.0,0.0)
2	F	0.530	0.135	0.6770	0.21	9	2.0	(0.0,0.0)

Convert features into a single vector per row

```
In [41]: vectorAssembler = VectorAssembler(inputCols = ["length", "height", "whole-weight",  
"shell-weight", "sex_vec"], outputCol = 'features')  
assembled_df = vectorAssembler.transform(encoded_df).select(['features', 'rings'])  
assembled_df.limit(5).toPandas()
```

Out[41]:

	features	rings
0	[0.455, 0.095, 0.514, 0.15, 1.0, 0.0]	15
1	[0.35, 0.09, 0.2255, 0.07, 1.0, 0.0]	7
2	[0.53, 0.135, 0.677, 0.21, 0.0, 0.0]	9
3	[0.44, 0.125, 0.516, 0.155, 1.0, 0.0]	10
4	[0.33, 0.08, 0.205, 0.055, 0.0, 1.0]	7

Merge the three transformers into one pipeline

```
In [42]: pipeline = Pipeline().setStages([sexIndexer, sexOneHot, vectorAssembler])
transformed_df = pipeline.fit(df).transform(df).select("features", "rings")
transformed_df.limit(5).toPandas()
```

Out[42]:

	features	rings
0	[0.455, 0.095, 0.514, 0.15, 1.0, 0.0]	15
1	[0.35, 0.09, 0.2255, 0.07, 1.0, 0.0]	7
2	[0.53, 0.135, 0.677, 0.21, 0.0, 0.0]	9
3	[0.44, 0.125, 0.516, 0.155, 1.0, 0.0]	10
4	[0.33, 0.08, 0.205, 0.055, 0.0, 1.0]	7

Split into training and test set

```
In [43]: train_df, test_df = transformed_df.randomSplit([0.7, 0.3])
```

Train simple Linear Regression model

```
In [44]: from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol = 'features', labelCol='rings',standardization=False, fitIntercept=False)
lr_model = lr.fit(train_df)
print("Coefficients:\n")
features = list(df.columns[:4]) + indexer_model.labels[:2]
for feature, coeff in zip(features, lr_model.coefficients):
    print("{}: {:.2f}".format(feature, coeff))
```

Coefficients:

```
sex: 16.27
length: 14.12
height: -8.20
whole-weight: 26.02
M: 0.24
I: -0.44
```

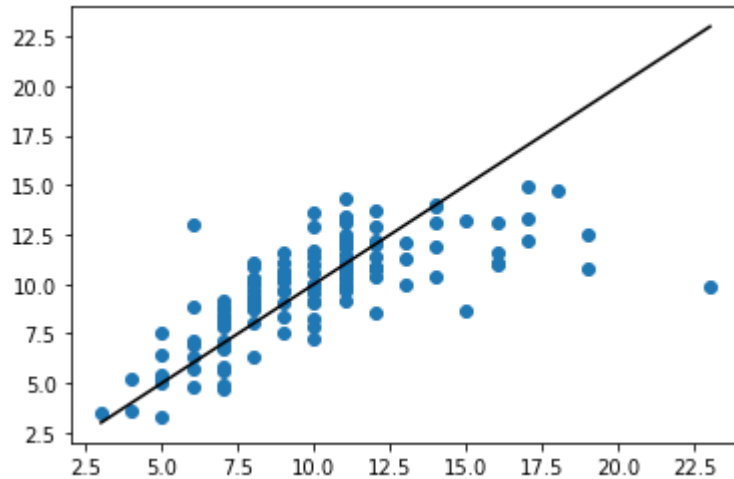
```
In [47]: predictions = lr_model.transform(test_df)
         predictions.limit(5).toPandas()
```

Out[47]:

	features	rings	prediction
0	[0.13, 0.03, 0.013000000000000001, 0.004, 0.0, ...	3	2.091571
1	[0.13, 0.035, 0.0105, 0.0035, 0.0, 1.0]	4	2.169672
2	[0.155, 0.025, 0.024, 0.0075, 1.0, 0.0]	5	3.118008
3	[0.165, 0.02, 0.019, 0.005, 0.0, 1.0]	4	2.496648
4	[0.165, 0.05, 0.021, 0.013999999999999999, 0.0...	3	3.138163

Visualization of ground truth vs predictions for the test set subsample

```
In [48]: test_sample = predictions.sample(False, 0.1, 42).toPandas()
minmax = (test_sample["rings"].min(), test_sample["rings"].max())
plt.scatter(test_sample["rings"], test_sample["prediction"])
_ = plt.plot(minmax, minmax, c="black")
```



Quiz

❖ *True or false?*

- ❖ The `OneHotEncoderEstimator` can transform categorical string variables into binary features
- ❖ You can split a `DataFrame` with the method `randomSplit()`

❖ What is the function of the following estimators?

- ❖ `StringIndexer`
- ❖ `OneHotEncoderEstimator`
- ❖ `VectorAssembler`

Quiz

❖ True or false?

- ❖ The `OneHotEncoderEstimator` can transform categorical string variables into binary features false
- ❖ You can split a `DataFrame` with the method `randomSplit()` true

❖ What is the function of the following estimators?

- ❖ `StringIndexer` Transforms a categorical string variable into a categorical float variable
- ❖ `OneHotEncoderEstimator` Transforms one categorical feature with n possible values into $n - 1$ binary features
- ❖ `VectorAssembler` Transforms n feature columns into one n -th dimensional vector column

Recap

Summary

- ❖ Database
 - ❖ SQL vs. noSQL databases
 - ❖ MongoDB
 - ❖ Setup, and querying a database
- ❖ Distributed computing
 - ❖ Apache Spark
 - ❖ DataFrame and SQL API
 - ❖ Machine learning with Spark's "Estimator API"