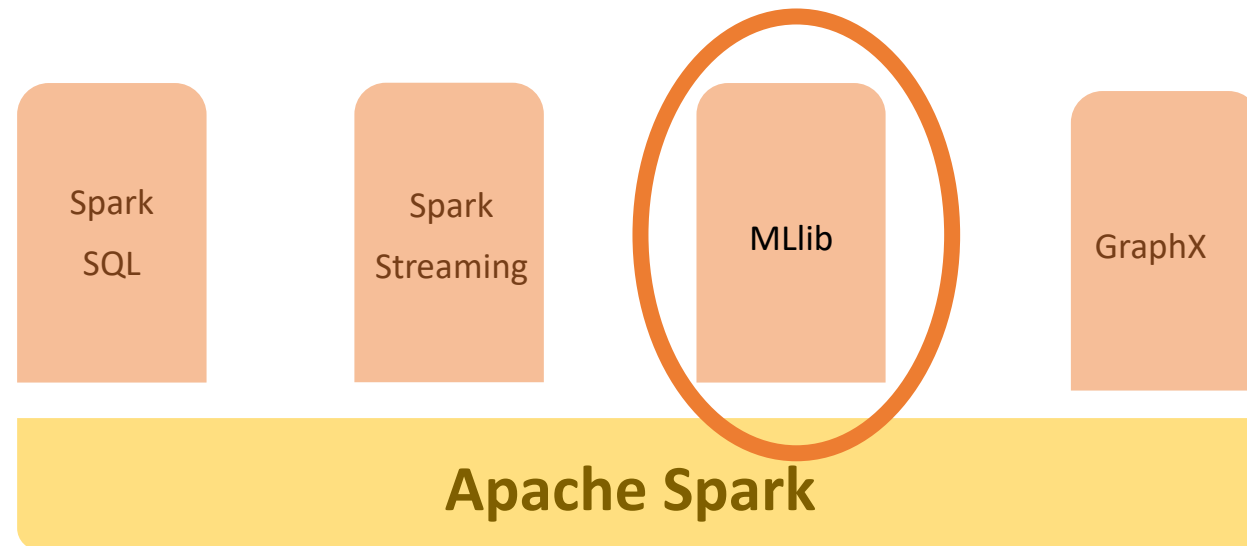


Apache Spark



Mahtab Jamali (Mahtab.Jamali@mau.se)
Ali Soleimani (ali.Soleimani@mau.se)

Key Points

- In this LAB, with work with ApacheSpark, MLlib library.
- For implementation, you are expected to first install Spark on your Windows. However, if you encounter any issues regarding installation, you can use Google Colab instead.
- You are expected to do 2 projects: one involving **a Machine Learning Project without Mllib Pipeline**, and the other involving **a Machine Learning Project with Mllib Pipeline**. Each project consists of 15 points (totaling 30 points). Completion of both projects is mandatory to pass this lab.
- For Project Number 1, Checking Spark jobs (If you install Spark on your Windows) is optional, and if you skip this step, you will get 10 points. If you use Colab instead, you will get 15 points without this section because on colab you cannot check the jobs! . Regarding Project Number 2, there is no optional task and you are expected to do all steps. (Keep in mind that to pass all labs, you generally need 100 points.)
- There are several slides about **Deep Learning with PySpark MLlib** after the implementation slides. This section is intended solely for your knowledge, and if you wish to undertake a Deep Learning project such as CNN models, you can also utilize Spark. No implementation is expected for this part

Spark MLlib



Apache Spark comes with a library named **MLlib** to perform Machine Learning tasks using the Spark framework.

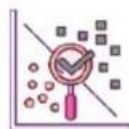


Spark MLlib Tools



ML Algorithms

classification, regression, clustering, and collaborative filtering



Featurization

feature extraction, transformation, dimensionality reduction, and selection



Pipelines

tools for constructing, evaluating, and tuning ML pipelines



Persistence

saving and loading algorithms, models and pipelines



Utilities

linear algebra, statistics, data handling

MLlib: Supported Algorithms

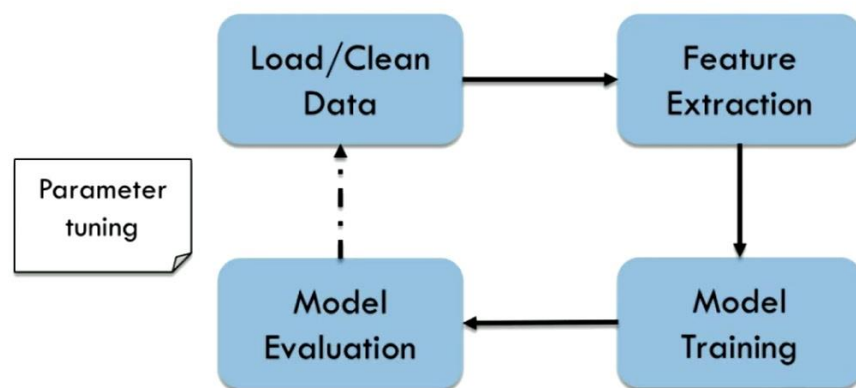
- **Data types**
- **Basic statistics**
 - Summary Statistics
 - Correlations
 - Stratified Sampling
 - Hypothesis Testing
 - Random Data Generation
- **Classification and regression**
 - **Linear Models (SVMs, logistic regression, linear regression)**
 - **Naive Bayes**
 - **Decision Trees**
 - **Ensembles of Trees** (Random Forests and Gradient - Boosted Trees)
- **Collaborative filtering**
 - Alternating Least Squares (ALS)
- **Clustering**
 - k-Means
 - Gaussian Mixture
 - Power Iteration
- **Dimensionality reduction**
 - Singular Value Decomposition (SVD)
 - Principal Component Analysis (PCA)
- **Feature extraction and transformation**
- **Optimization (developer)**
 - Stochastic Gradient Descent
 - Limited-Memory BFGS (L-BFGS)

Spark ML Concepts

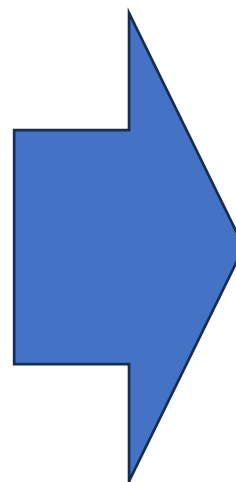
Concept	Explanation
DataFrame	Spark ML uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a DataFrame could have different columns storing text, feature vectors, true labels, and predictions
Transformer	A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms DataFrame with features into a DataFrame with predictions
Estimator	An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model
Pipeline	A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow
Parameter	All Transformers and Estimators now share a common API for specifying parameters

MLib Pipeline Concepts

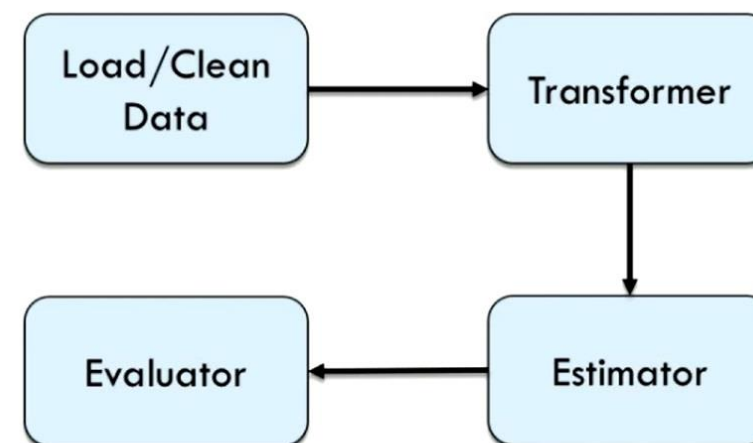
Machine Learning Pipeline



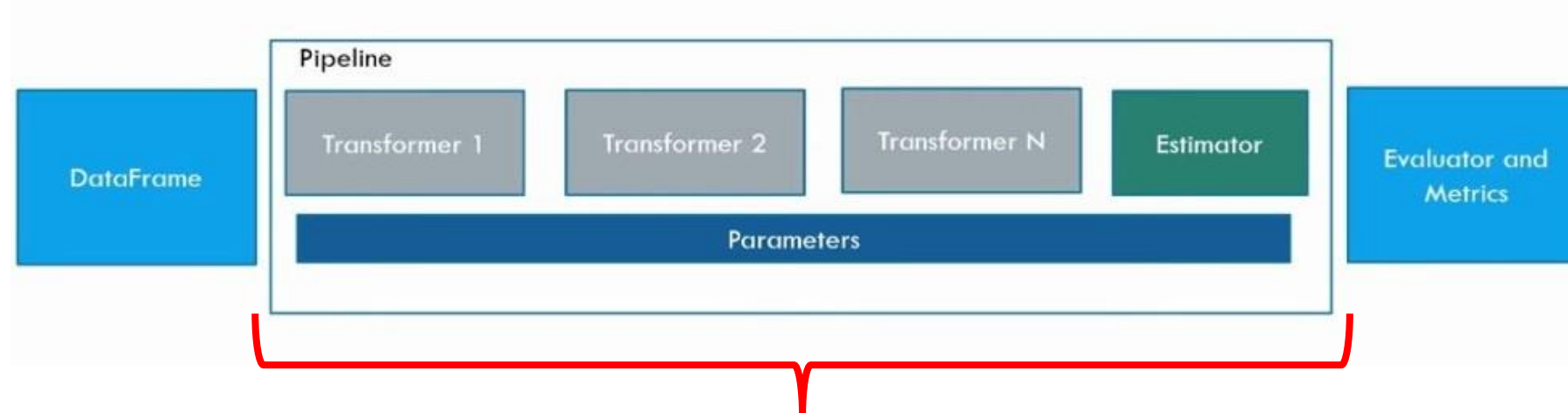
Repeatable Workflow Process



MLib Pipeline Concepts



MLlib Pipeline



Pipeline is combination of Transformers and one Estimator

The input of a transformer is a dataframe and the output of the transformer is a dataframe.
The input of the estimator is a dataframe and the output of the dataframe is a model.



Transformer and estimator

TRANSFORMER

- Feature Transformers
 - Tokenizer
 - StopWordsRemover
 - n -gram
 - Binarizer
 - PCA
 - PolynomialExpansion
 - Discrete Cosine Transform (DCT)
 - StringIndexer
 - IndexToString
 - OneHotEncoder (Deprecated since 2.3.0)
 - OneHotEncoderEstimator
 - VectorIndexer
 - Interaction
 - Normalizer
 - StandardScaler
 - MinMaxScaler
 - MaxAbsScaler
 - Bucketizer
 - ElementwiseProduct
 - SQLTransformer
 - VectorAssembler
 - VectorSizeHint
 - QuantileDiscretizer
 - Imputer

ESTIMATOR

- Classification
 - Logistic regression
 - Binomial logistic regression
 - Multinomial logistic regression
 - Decision tree classifier
 - Random forest classifier
 - Gradient-boosted tree classifier
 - Multilayer perceptron classifier
 - Linear Support Vector Machine
 - One-vs-Rest classifier (a.k.a. One-vs-All)
 - Naive Bayes
- Regression
 - Linear regression
 - Generalized linear regression
 - Available families
 - Decision tree regression
 - Random forest regression
 - Gradient-boosted tree regression
 - Survival regression
 - Isotonic regression
- Linear methods

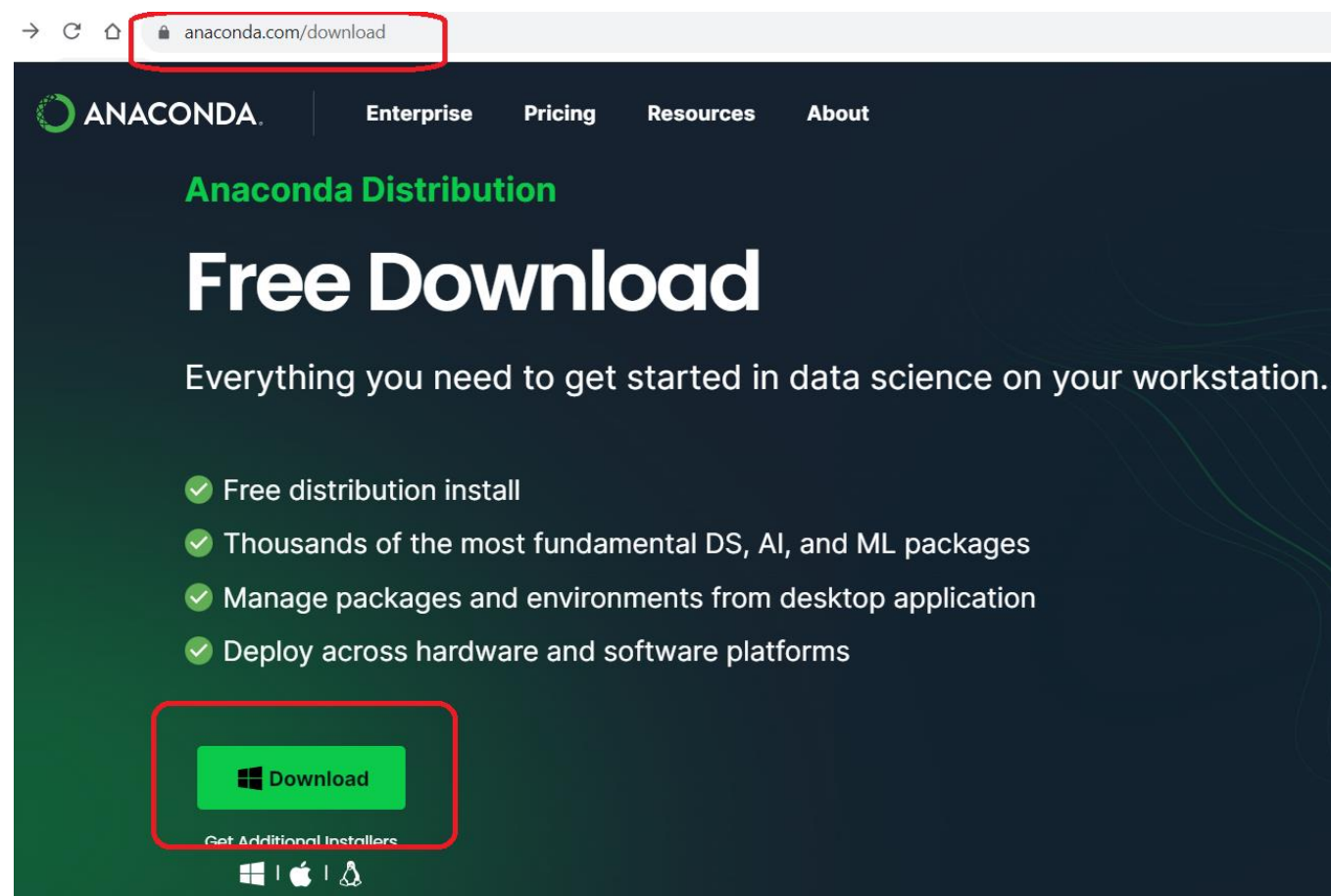
Project 1: Machine Learning Project without Mllib Pipeline

Contents of implementation (without pipeline)

- Setting up Spark Environment on Windows and Google Colab
- Data loading
- Data Preprocessing using SparkML
- Model Training and Testing using SparkML
- Prediction on Test data
- Evaluation of predictions

•Setting up Spark Environment on Windows

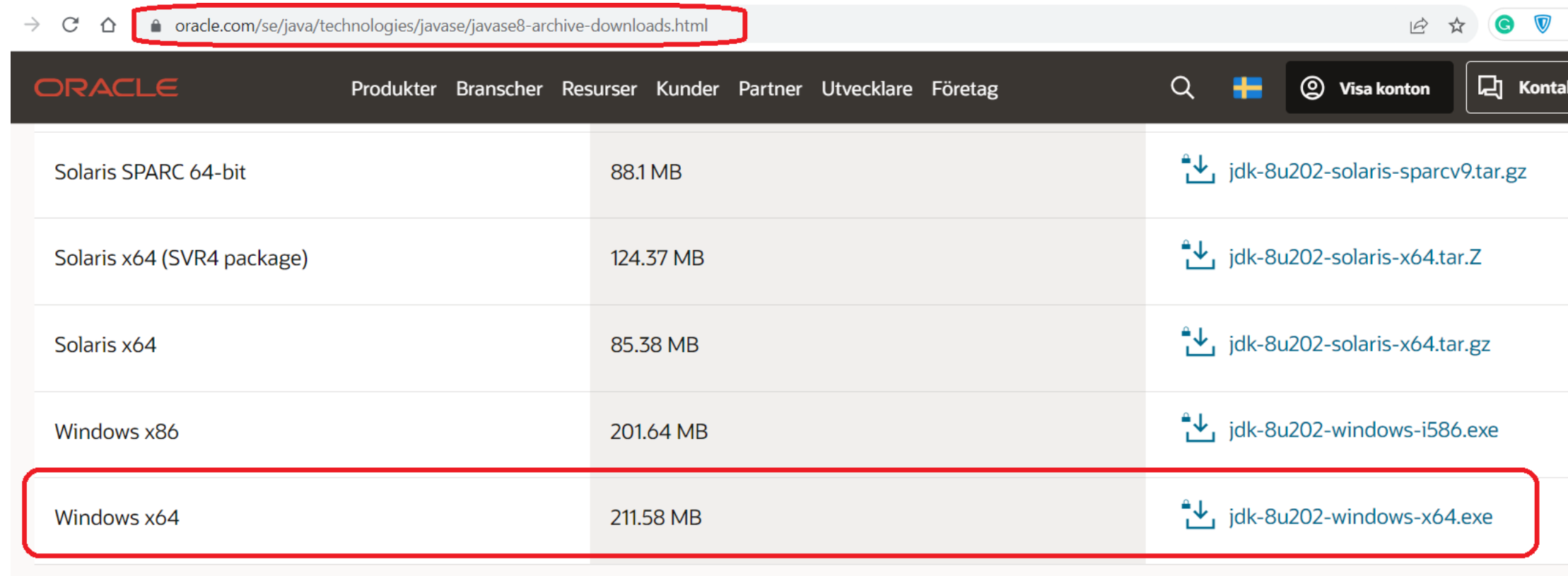
1. Anaconda Installation



•Setting up Spark Environment on Windows

2. Downloading Java Version 8

<https://www.oracle.com/se/java/technologies/javase/javase8-archive-downloads.html>



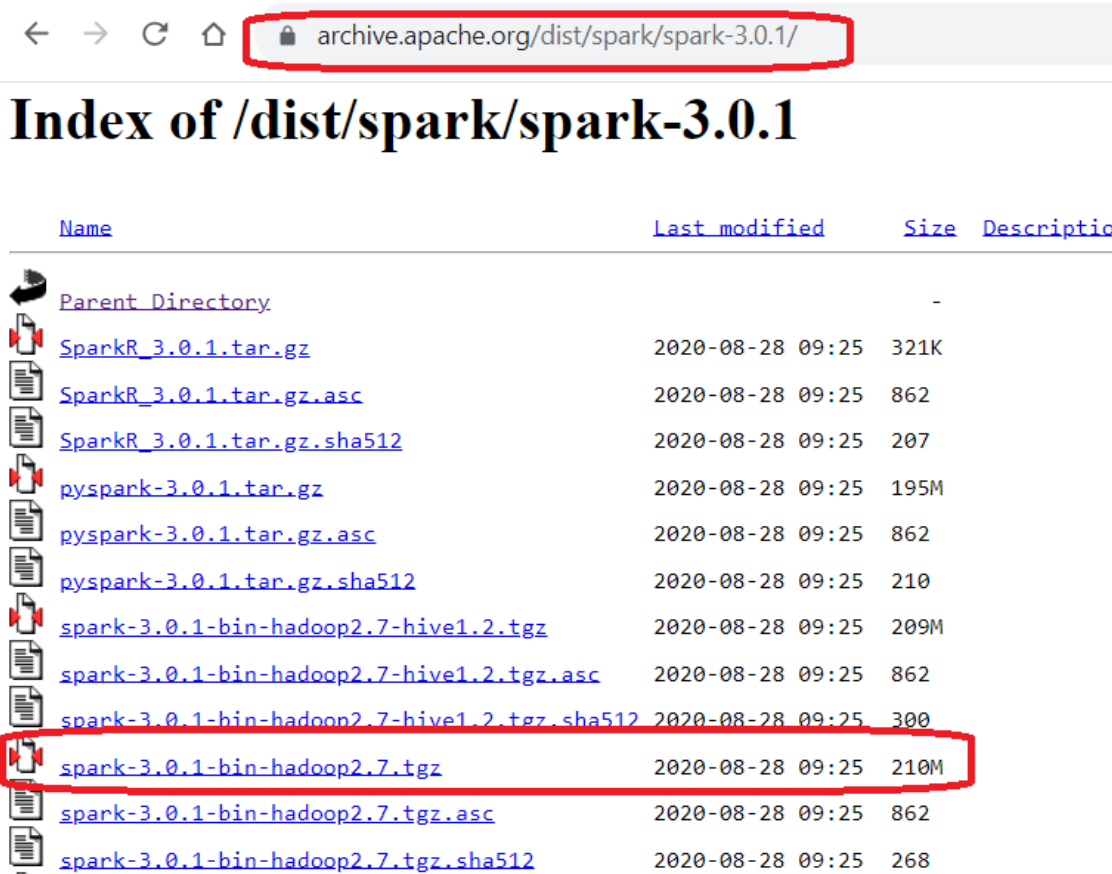
The screenshot shows the Oracle Java 8 archive download page. The browser's address bar is highlighted with a red box, showing the URL: `oracle.com/se/java/technologies/javase/javase8-archive-downloads.html`. The page features a dark navigation bar with the Oracle logo and links for Produkter, Branscher, Resurser, Kunder, Partner, Utvecklare, and Företag. On the right of the navigation bar are search, language (Sweden), login (Visa konton), and contact (Kontakt) options. Below the navigation bar is a table of download links for various operating systems. The table has three columns: the operating system name, the file size, and the download link. The last row, for Windows x64, is highlighted with a red box.

Solaris SPARC 64-bit	88.1 MB	jdk-8u202-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	124.37 MB	jdk-8u202-solaris-x64.tar.Z
Solaris x64	85.38 MB	jdk-8u202-solaris-x64.tar.gz
Windows x86	201.64 MB	jdk-8u202-windows-i586.exe
Windows x64	211.58 MB	jdk-8u202-windows-x64.exe

•Setting up Spark Environment on Windows














3. Spark Installation

- Go to the <https://archive.apache.org/dist/spark/>
- Open **spark-3.0.1/** folder
- Download **spark-3.0.1-bin-hadoop2.7.tgz**



← → ↻ ⌂ 🔒 archive.apache.org/dist/spark/spark-3.0.1/

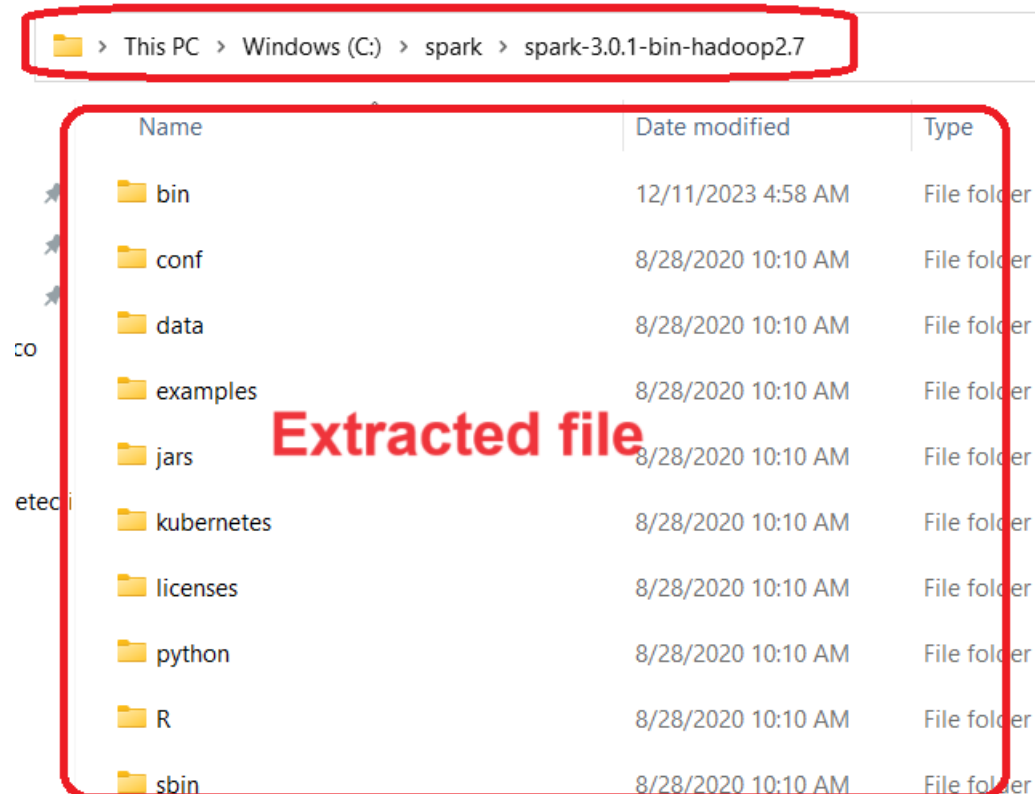
Index of /dist/spark/spark-3.0.1

Name	Last modified	Size	Description
 Parent Directory	-	-	-
 SparkR_3.0.1.tar.gz	2020-08-28 09:25	321K	
 SparkR_3.0.1.tar.gz.asc	2020-08-28 09:25	862	
 SparkR_3.0.1.tar.gz.sha512	2020-08-28 09:25	207	
 pyspark-3.0.1.tar.gz	2020-08-28 09:25	195M	
 pyspark-3.0.1.tar.gz.asc	2020-08-28 09:25	862	
 pyspark-3.0.1.tar.gz.sha512	2020-08-28 09:25	210	
 spark-3.0.1-bin-hadoop2.7-hive1.2.tgz	2020-08-28 09:25	209M	
 spark-3.0.1-bin-hadoop2.7-hive1.2.tgz.asc	2020-08-28 09:25	862	
 spark-3.0.1-bin-hadoop2.7-hive1.2.tgz.sha512	2020-08-28 09:25	300	
 spark-3.0.1-bin-hadoop2.7.tgz	2020-08-28 09:25	210M	
 spark-3.0.1-bin-hadoop2.7.tgz.asc	2020-08-28 09:25	862	
 spark-3.0.1-bin-hadoop2.7.tgz.sha512	2020-08-28 09:25	268	

•Setting up Spark Environment on Windows

4. Spark Installation

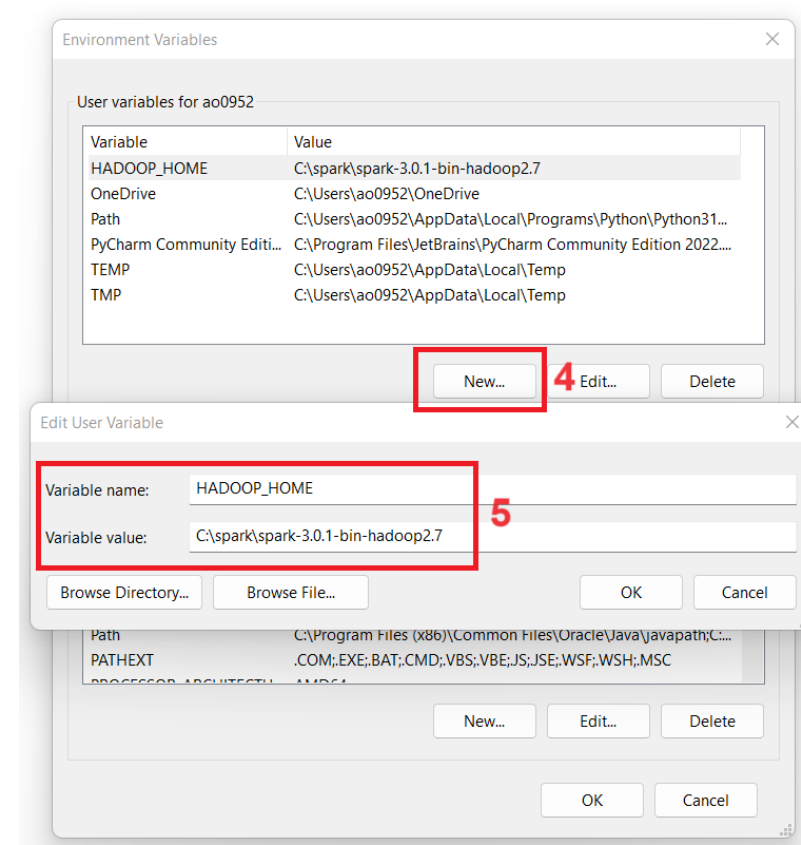
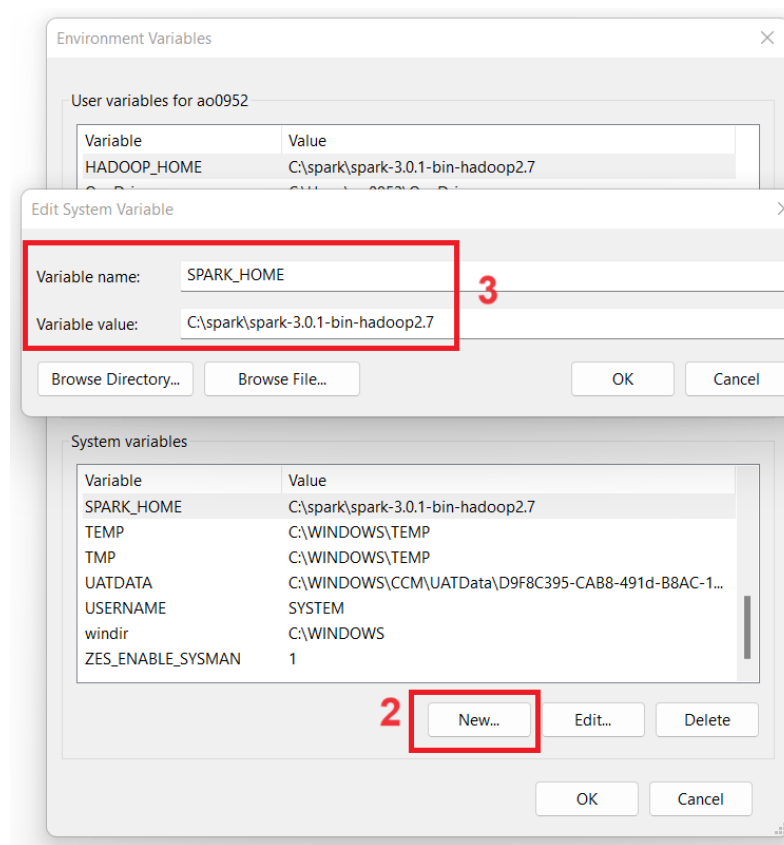
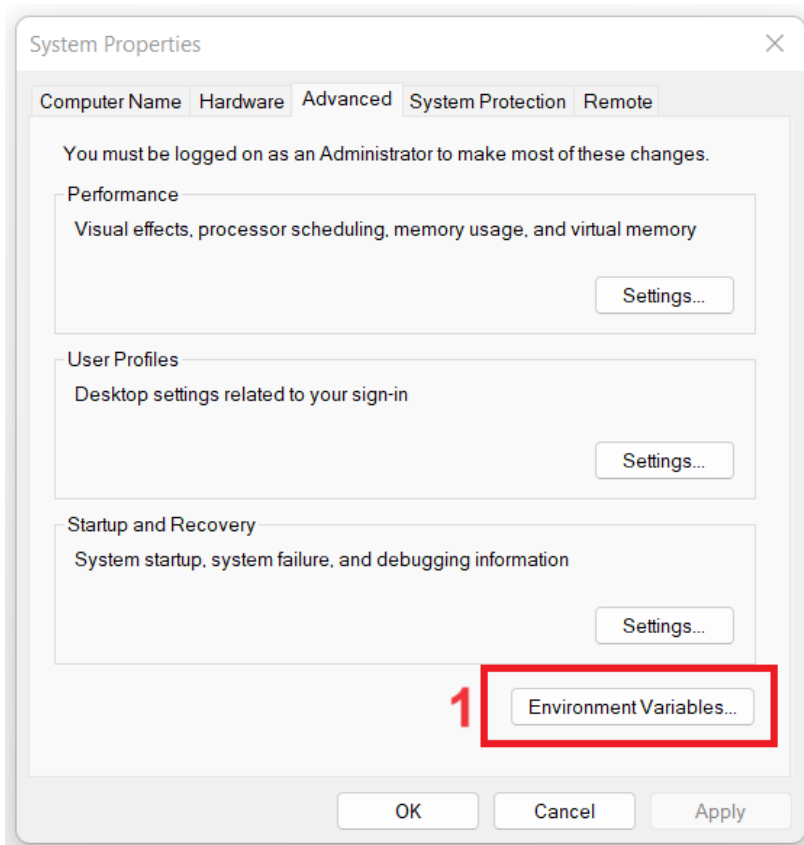
- Extract downloaded spark file
- Create a folder in Drive C titled 'Spark' and paste it there



Name	Date modified	Type
bin	12/11/2023 4:58 AM	File folder
conf	8/28/2020 10:10 AM	File folder
data	8/28/2020 10:10 AM	File folder
examples	8/28/2020 10:10 AM	File folder
jars	8/28/2020 10:10 AM	File folder
kubernetes	8/28/2020 10:10 AM	File folder
licenses	8/28/2020 10:10 AM	File folder
python	8/28/2020 10:10 AM	File folder
R	8/28/2020 10:10 AM	File folder
sbin	8/28/2020 10:10 AM	File folder

•Setting up Spark Environment on Windows

5. Go to the **advanced system settings** of your PC, and add 2 variables.



Environment Variables

User variables for ao0952

Variable	Value
HADOOP_HOME	C:\spark\spark-3.0.1-bin-hadoop2.7
OneDrive	C:\Users\ao0952\OneDrive
Path	C:\Users\ao0952\AppData\Local\Programs\Python\Python31...
PyCharm Community Editi...	C:\Program Files\JetBrains\PyCharm Community Edition 2022....
TEMP	C:\Users\ao0952\AppData\Local\Temp
TMP	C:\Users\ao0952\AppData\Local\Temp

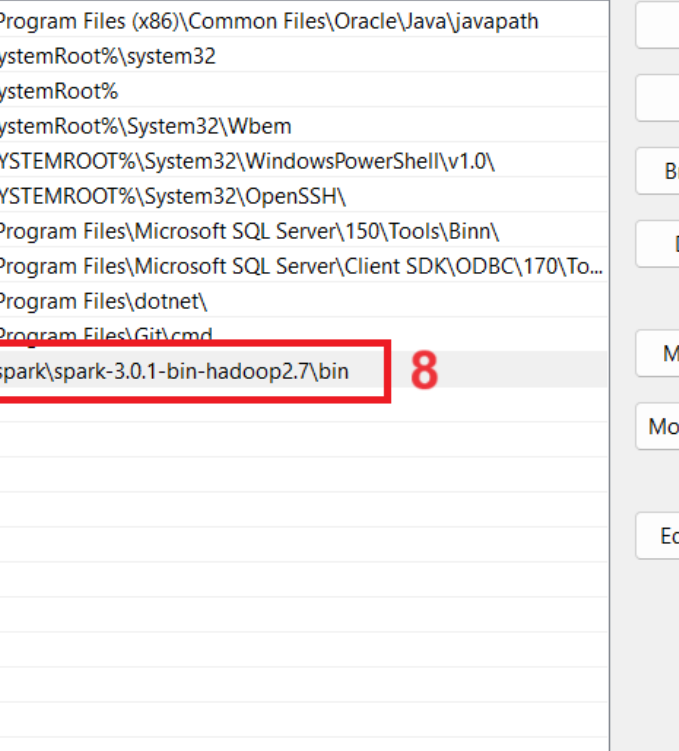
New... Edit... Delete

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
IGCCSVC_DB	AQAAANCMnd8BFdERjHoAwE/CI+sBAAAAHlvWPKxFbEG3rJo...
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

New... Edit... Delete

OK Cancel



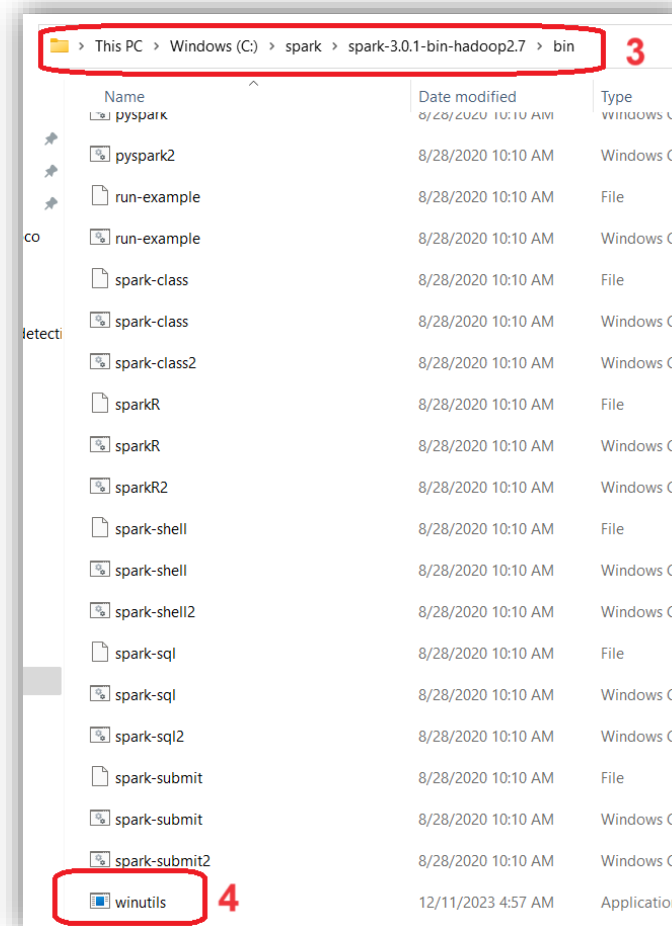
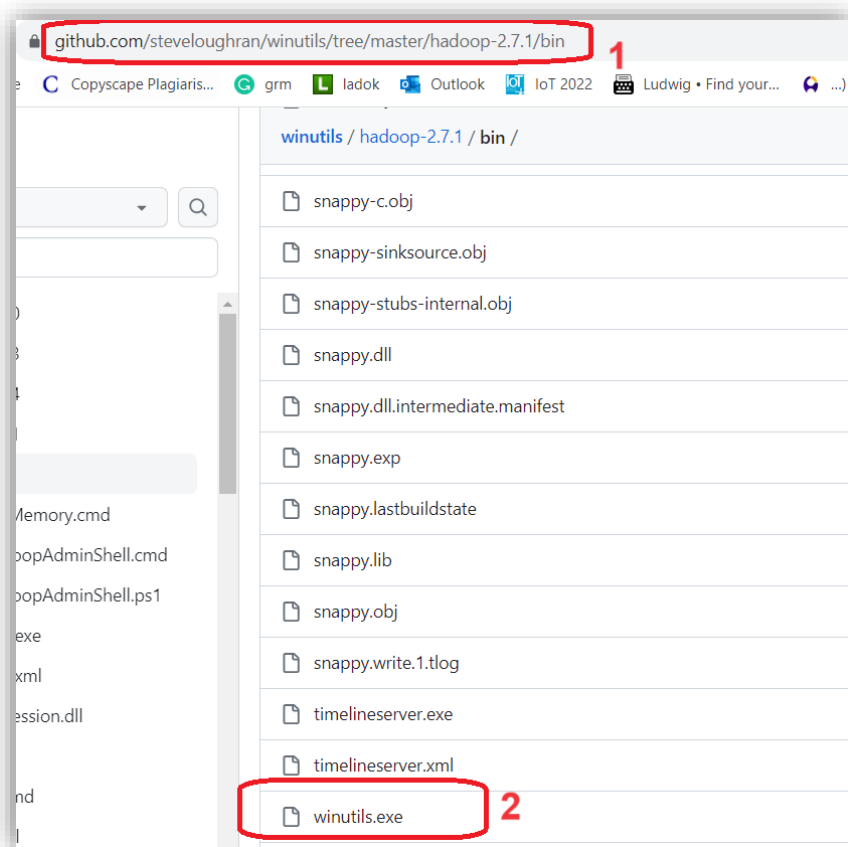
8

•Setting up Spark Environment on Windows

7. Download Winutils.exe from the following link:

<https://github.com/steveloughran/winutils/tree/master/hadoop-2.7.1/bin>

8. Paste it in the bin folder of spark folder



•Setting up Spark Environment on Windows

9. Create a folder in Drive C (C:\tmp\hive)
10. Open cmd of your PC (Run as administrator mode)
11. Follow the instructions below

```
C:\> Administrator: Command Prompt
Microsoft Windows [Version 10.0.22000.2600]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd\ 1
C:\>cd spark 2
C:\spark>cd spark-3.0.1-bin-hadoop2.7\bin 3
C:\spark\spark-3.0.1-bin-hadoop2.7\bin>winutils.exe chmod -R 777 C:\tmp\hive 4
C:\spark\spark-3.0.1-bin-hadoop2.7\bin>winutils.exe ls -F C:\tmp\hive 5
```

•Setting up Spark Environment on Windows

12. Open Anaconda Prompt on your personal PC
13. Follow the instructions to install **findspark**

Anaconda Prompt

```
(base) C:\Users\ao0952>conda install -c conda-forge findspark
```

1

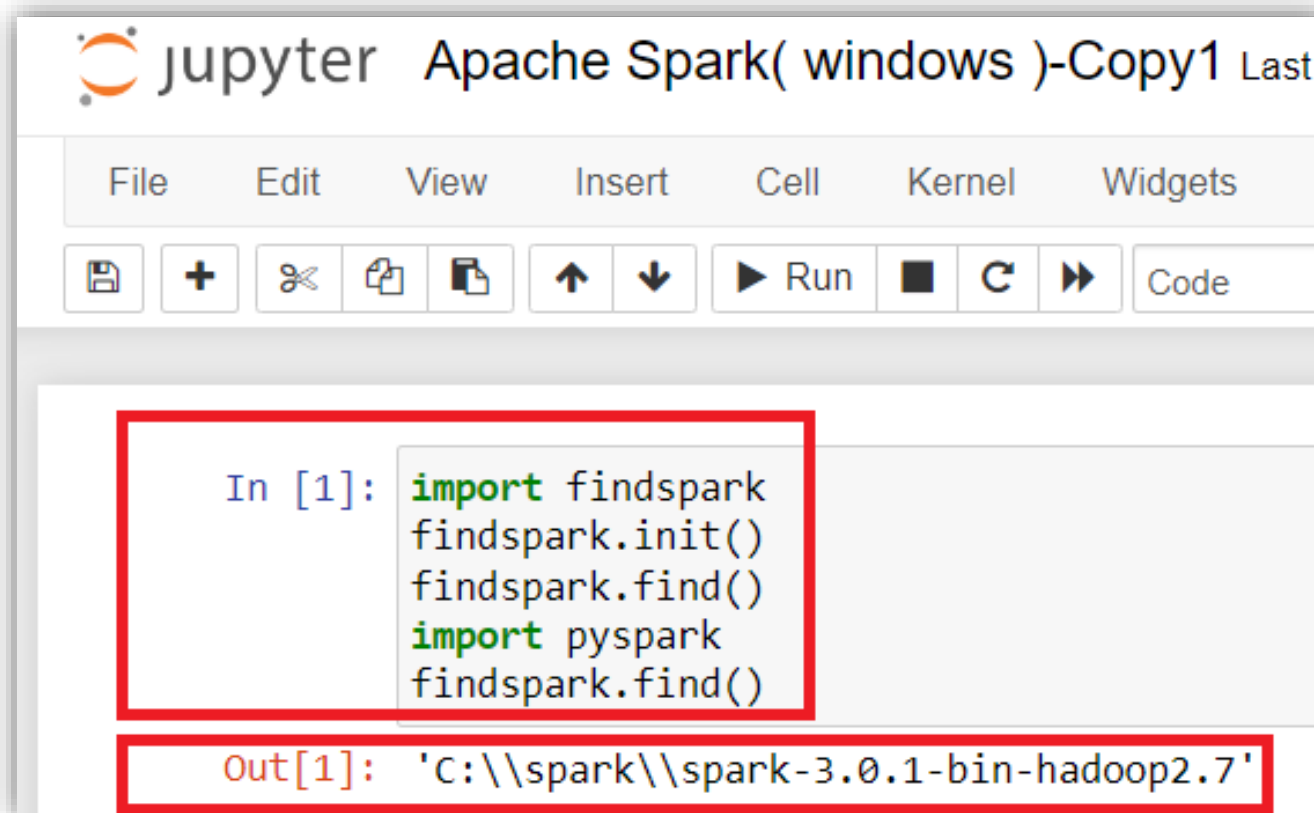
```
Proceed ([y]/n)? y
```

2

•Setting up Spark Environment on Windows

If you have followed all the steps correctly, after entering the following code in Jupyter Notebook, you will get : 'C:\\spark\\spark-3.0.1-bin-hadoop2.7'

You did it

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter Apache Spark(windows)-Copy1'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Widgets'. The toolbar contains icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code cell contains the following Python code:

```
In [1]: import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

 The output cell below shows the result:

```
Out[1]: 'C:\\spark\\spark-3.0.1-bin-hadoop2.7'
```

 Both the code and output cells are highlighted with red rectangular boxes.

•Setting up Spark Environment on Colab

If you encounter issues installing Spark on your Windows system, you can utilize Google Colab instead.

```
# Setting up the PySpark environment

# Download Java Virtual Machine (JVM)
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# Replace 'path_to_spark_archive' with the actual path to your uploaded Spark archive
path_to_spark_archive = '/content/drive/MyDrive/big data course/spark-3.4.1-bin-hadoop3.tgz'

# Unzip the spark file
!tar xf "{path_to_spark_archive}"

## Add environmental variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = '/content/spark-3.2.1-bin-hadoop3.2'

# Install library for finding Spark
!pip install -q findspark

# Import the library
import findspark

# Replace 'path_to_spark' with the actual path to your Spark installation directory
path_to_spark = '/content/spark-3.4.1-bin-hadoop3'

# Initiate findspark with the correct path
findspark.init(path_to_spark)

# Check the location for Spark
findspark.find()
```

```
# Import SparkSession
from pyspark.sql import SparkSession

# Create a Spark Session
spark = SparkSession.builder \
    .master("local") \
    .appName("Titanic data") \
    .getOrCreate()

# Check Spark Session Information
spark
```

SparkSession - in-memory

SparkContext

[Spark UI](#)

Version

v3.4.1

Master

local

AppName

Titanic data

- **After Spark installation on Windows**

After installing Spark on your personal PC, it's time to dive into coding. In the next slides, you will learn how to import the necessary libraries, initialize Spark, and, finally, train a model for your dataset. So, let's go!

Initialize SparkSession

```
In [2]: from pyspark.sql import SparkSession
```

```
In [3]: spark = SparkSession \
        .builder \
        .appName ('Titanic Data') \
        .getOrCreate()
```

```
In [4]: spark
```

```
Out[4]: SparkSession - in-memory
SparkContext
```

[Spark UI](#)

Version

v3.0.1

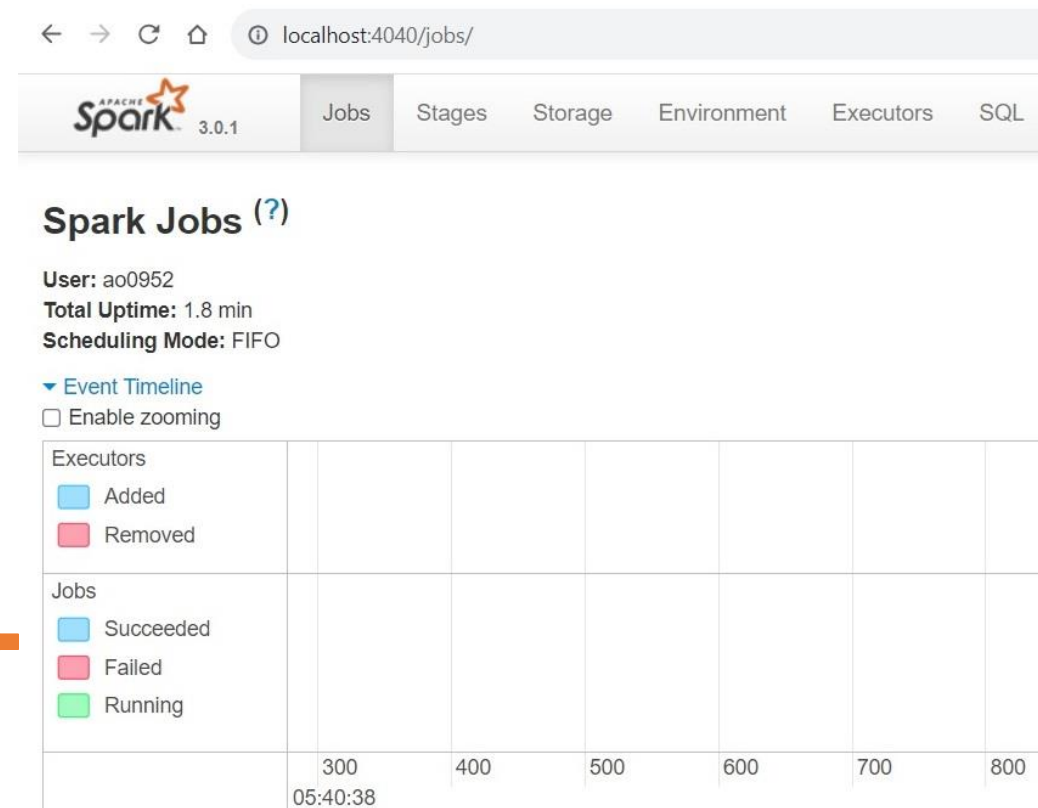
Master

local[*]

AppName

Titanic Data

If you click on Spark UI on Windows, the Spark environment will open in your browser, allowing you to monitor the processes of Spark masters and workers and manage them. Initially, the environment is empty.



Reading Data

- Dataset (Titanic): <https://www.kaggle.com/c/titanic/data>

```
df = (spark.read
      .format("csv")
      .option("header", "true")
      .load("/content/drive/MyDrive/big data course/titanic dataset/train.csv"))
```

```
df.show(5)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. ...	female	26	0	0	STON/O2. 3101282	7.925	null	S
4	1	1	Futrelle, Mrs. Ja...	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. Willia...	male	35	0	0	373450	8.05	null	S

only showing top 5 rows

Selecting some columns (if needed)

- From `pyspark.sql.functions` import `col` and then select columns

```
from pyspark.sql.functions import col
```

```
dataset = df.select(col('Survived').cast('float'),  
                    col('Pclass').cast('float'),  
                    col('Sex'),  
                    col('Age').cast('float'),  
                    col('Fare').cast('float'),  
                    col('Embarked')  
                    )
```

```
dataset.show(4)
```

Survived	Pclass	Sex	Age	Fare	Embarked
0.0	3.0	male	22.0	7.25	S
1.0	1.0	female	38.0	71.2833	C
1.0	3.0	female	26.0	7.925	S
1.0	1.0	female	35.0	53.1	S

only showing top 4 rows

Removing null values (if needed)

```
from pyspark.sql.functions import isnull, when, count, col
```

```
dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset.columns]).show()
```

Survived	Pclass	Sex	Age	Fare	Embarked
0	0	0	177	0	2

```
dataset = dataset.replace('?', None)\  
    .dropna(how='any')
```

```
dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset.columns]).show()
```

Survived	Pclass	Sex	Age	Fare	Embarked
0	0	0	0	0	0

Converting categorical variables to numeric values

- Spark only supports numeric values and is incapable of handling categorical variables. For modeling, all categorical variables must be converted to numeric values. To achieve this, **StringIndexer** is employed.

1

```
dataset.show(3)
```

Should be converted

Survived	Pclass	Sex	Age	Fare	Embarked
0.0	3.0	male	22.0	7.25	S
1.0	1.0	female	38.0	71.2833	C
1.0	3.0	female	26.0	7.925	S

only showing top 3 rows

4

```
dataset.show(2)
```

Survived	Pclass	Sex	Age	Fare	Embarked	Gender	Boarded
0.0	3.0	male	22.0	7.25	S	0.0	0.0
1.0	1.0	female	38.0	71.2833	C	1.0	1.0

only showing top 2 rows

2

```
from pyspark.ml.feature import StringIndexer
```

3

```
dataset = StringIndexer(  
    inputCol='Sex',  
    outputCol='Gender',  
    handleInvalid='keep').fit(dataset).transform(dataset)
```

```
dataset = StringIndexer(  
    inputCol='Embarked',  
    outputCol='Boarded',  
    handleInvalid='keep').fit(dataset).transform(dataset)
```

Finally, we can drop 'sex' and 'embarked' columns

5

```
#Drop unnecessary columns  
dataset = dataset.drop('Sex')  
dataset = dataset.drop('Embarked')
```

Feature engineering

- Spark learns through two columns, **label** and **feature**. Therefore, all columns except the target column must be combined into a single column. This is accomplished via **VectorAssembler**.

```
#Assemble all the feautures with VectorAssembler
from pyspark.ml.feature import VectorAssembler
```

```
require_featured = ['Pclass', 'Age', 'Fare', 'Gender', 'Boarded']
assembler = VectorAssembler(inputCols=require_featured, outputCol='features')
transformed_data = assembler.transform(dataset)
```

```
transformed_data.show(5)
```

Label
(target variables)

Survived	Pclass	Age	Fare	Gender	Boarded	features
0.0	3.0	22.0	7.25	0.0	0.0	[3.0,22.0,7.25,0....]
1.0	1.0	38.0	71.2833	1.0	1.0	[1.0,38.0,71.2833...]
1.0	3.0	26.0	7.925	1.0	0.0	[3.0,26.0,7.92500...]
1.0	1.0	35.0	53.1	1.0	0.0	[1.0,35.0,53.0999...]
0.0	3.0	35.0	8.05	0.0	0.0	[3.0,35.0,8.05000...]

only showing top 5 rows

feature



Modeling

```
#splitting dataset into train and test
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2])
print("Number of train samples: " + str(training_data.count()))
print("Number of train samples: " + str(test_data.count()))
```

Number of train samples: 572

Number of train samples: 140

```
from pyspark.ml.classification import RandomForestClassifier
```

```
rf = RandomForestClassifier(labelCol='Survived',
                           featuresCol='features',
                           maxDepth=5)
```

```
model = rf.fit(training_data)
```

```
predictions = model.transform(test_data)
```

Evaluation

```
#Evaluation
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

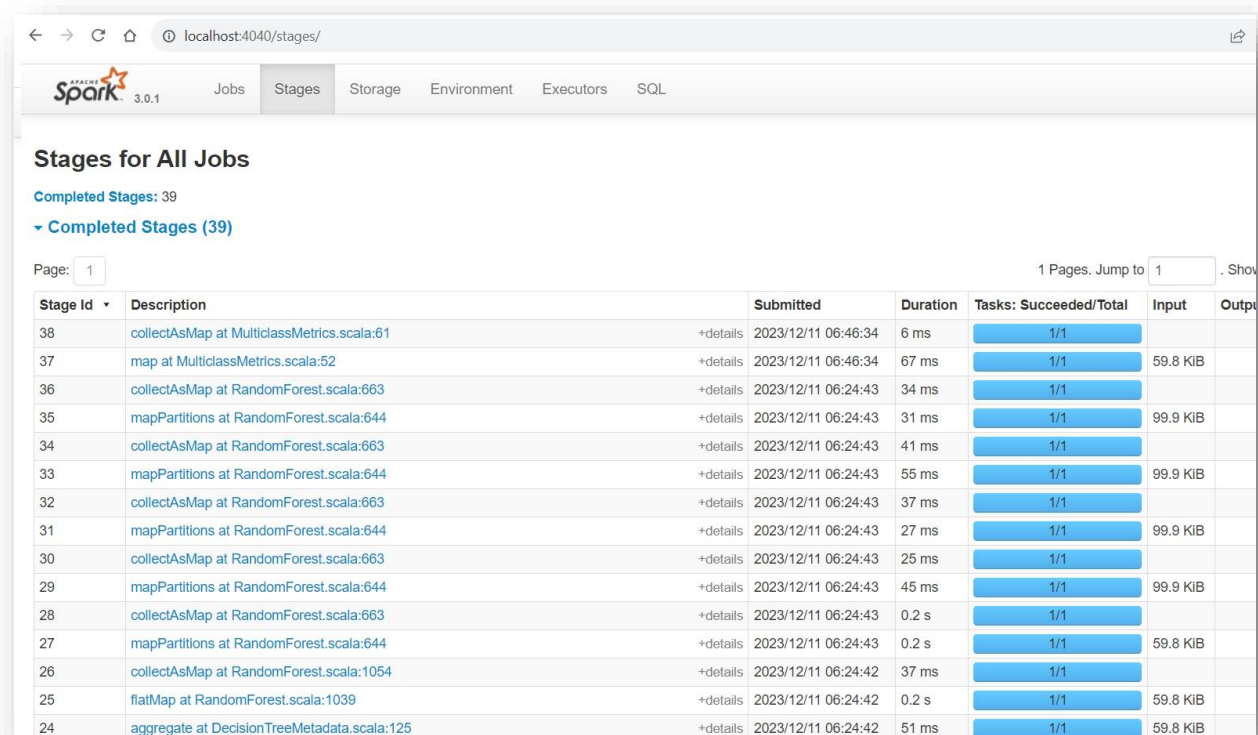
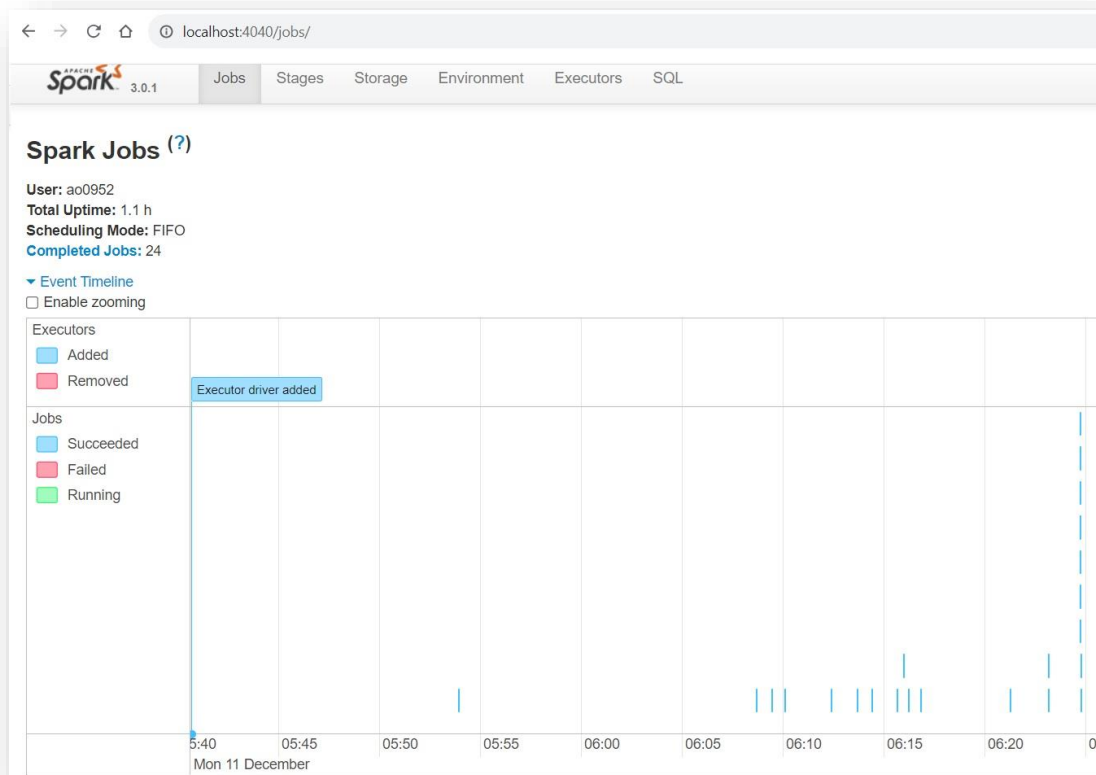
```
evaluator = MulticlassClassificationEvaluator(  
    labelCol='Survived',  
    predictionCol='prediction',  
    metricName = 'accuracy')
```

```
accuracy = evaluator.evaluate(predictions_test)  
print('Training Accuracy = ', accuracy)
```

```
Training Accuracy = 0.8091603053435115
```

Checking Spark jobs

After all you can check **Spark Jobs** on your local machine and manage them.



← → ↻ 🏠 localhost:4040/stages/

Apache Spark 3.0.1 Jobs Stages Storage Environment Executors SQL

Stages for All Jobs

Completed Stages: 39

▼ Completed Stages (39)

Page: 1 1 Pages. Jump to 1 . Show

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output
38	collectAsMap at MulticlassMetrics.scala:61	2023/12/11 06:46:34	6 ms	1/1		
37	map at MulticlassMetrics.scala:52	2023/12/11 06:46:34	67 ms	1/1	59.8 KiB	
36	collectAsMap at RandomForest.scala:663	2023/12/11 06:24:43	34 ms	1/1		
35	mapPartitions at RandomForest.scala:644	2023/12/11 06:24:43	31 ms	1/1	99.9 KiB	
34	collectAsMap at RandomForest.scala:663	2023/12/11 06:24:43	41 ms	1/1		
33	mapPartitions at RandomForest.scala:644	2023/12/11 06:24:43	55 ms	1/1	99.9 KiB	
32	collectAsMap at RandomForest.scala:663	2023/12/11 06:24:43	37 ms	1/1		
31	mapPartitions at RandomForest.scala:644	2023/12/11 06:24:43	27 ms	1/1	99.9 KiB	
30	collectAsMap at RandomForest.scala:663	2023/12/11 06:24:43	25 ms	1/1		
29	mapPartitions at RandomForest.scala:644	2023/12/11 06:24:43	45 ms	1/1	99.9 KiB	
28	collectAsMap at RandomForest.scala:663	2023/12/11 06:24:43	0.2 s	1/1		
27	mapPartitions at RandomForest.scala:644	2023/12/11 06:24:43	0.2 s	1/1	59.8 KiB	
26	collectAsMap at RandomForest.scala:1054	2023/12/11 06:24:42	37 ms	1/1		
25	flatMap at RandomForest.scala:1039	2023/12/11 06:24:42	0.2 s	1/1	59.8 KiB	
24	aggregate at DecisionTreeMetadata.scala:125	2023/12/11 06:24:42	51 ms	1/1	59.8 KiB	

Project 2: Machine Learning Project with Mllib Pipeline

Contents of implementation (with pipeline)

Contents:

1. Setting up the environment
2. Read data
3. Leverage Spark ML pipeline

Setting up the environment

```
# Setting up the PySpark environment

# Download Java Virtual Machine (JVM)
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# Replace 'path_to_spark_archive' with the actual path to your uploaded Spark archive
path_to_spark_archive = '/content/drive/MyDrive/big data course/spark-3.4.1-bin-hadoop3.tgz'

# Unzip the spark file
!tar xf "{path_to_spark_archive}"

## Add environmental variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = '/content/spark-3.2.1-bin-hadoop3.2'

# Install library for finding Spark
!pip install -q findspark

# Import the library
import findspark

# Replace 'path_to_spark' with the actual path to your Spark installation directory
path_to_spark = '/content/spark-3.4.1-bin-hadoop3'

# Initiate findspark with the correct path
findspark.init(path_to_spark)

# Check the location for Spark
findspark.find()
```

Initialize SparkSession

```
# Import SparkSession
from pyspark.sql import SparkSession

# Create a Spark Session
spark = SparkSession.builder \
    .master("local") \
    .appName("Titanic data") \
    .getOrCreate()

# Check Spark Session Information
spark
```

SparkSession - in-memory

SparkContext

[Spark UI](#)

Version

v3.4.1

Master

local

AppName

Titanic data

Reading data

```
df = (spark.read
      .format("csv")
      .option("header", "true")
      .load("/content/drive/MyDrive/big data course/titanic dataset/train.csv"))
```

```
df.show(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. ...	female	26	0	0	STON/O2. 3101282	7.925	null	S

only showing top 3 rows

Importing functions

```
from pyspark.sql import functions as F
from pyspark.sql import types as T

# StringIndexer is similar to labelencoder which gives a label to each category
# OneHotEncoder created onehot encoding vector
from pyspark.ml.feature import StringIndexer, OneHotEncoder

# VectorAssembler is used to create vector from the features. Modeling takes vector as an input
from pyspark.ml.feature import VectorAssembler

# DecisionTreeClassifier is used for classification problems
from pyspark.ml.classification import RandomForestClassifier
```

Using pipeline

```
# Import pipeline from PySpark ML
from pyspark.ml import Pipeline
```

```
(train_df, test_df) = dataset.randomSplit([0.8, 0.2], 11)
print("Number of train samples: " + str(train_df.count()))
print("Number of test samples: " + str(test_df.count()))
```

```
Number of train samples: 562
Number of test samples: 150
```

```
# Label Encoding of categorical variables without any .fit or .transform
Sex_indexer = StringIndexer(inputCol="Sex", outputCol="Gender")
Embarked_indexer = StringIndexer(inputCol="Embarked", outputCol="Boarded")

# Assemble all the features with VectorAssembler
inputCols = ['Pclass', 'Age', 'Fare', 'Gender', 'Boarded']
outputCol = "features"
vector_assembler = VectorAssembler(inputCols = inputCols, outputCol = outputCol)

# Modeling using DecisionTreeClassifier
dt_model = RandomForestClassifier(labelCol="Survived", featuresCol="features")
```

Using pipeline

Transformers


Estimator

```
# Setup the pipeline
pipeline = Pipeline(stages=[Sex_indexer, Embarked_indexer, vector_assembler, dt_model])

# Fit the pipeline model
final_pipeline = pipeline.fit(train_df)

# Predict on test data
test_predictions_from_pipeline = final_pipeline.transform(test_df)

test_predictions_from_pipeline.show(5, truncate=False)
```



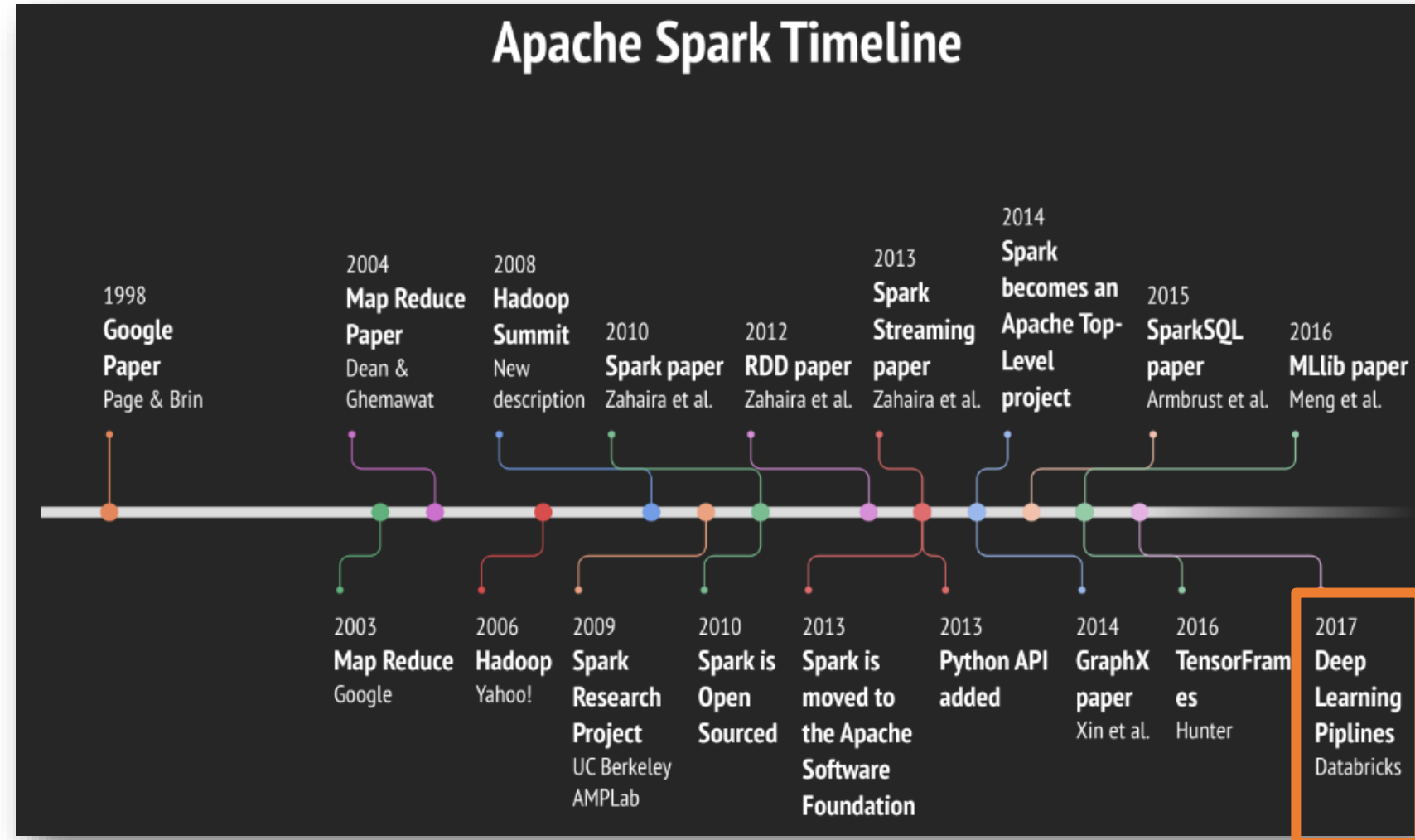
Deep learning with PySpark Mllib

Deep Learning Pipelines for Apache Spark

Deep Learning Pipelines is a new library published by Databricks to provide deep learning models and transfer learning via integration of popular deep learning libraries with MLib Pipelines and Spark SQL.

Deep Learning Pipelines provides a suite of tools around working with and processing images using deep learning. The tools can be categorized as:

- Working with images
- Transfer learning
- Applying deep learning models at scale



Using Spark to load images as a DataFrame

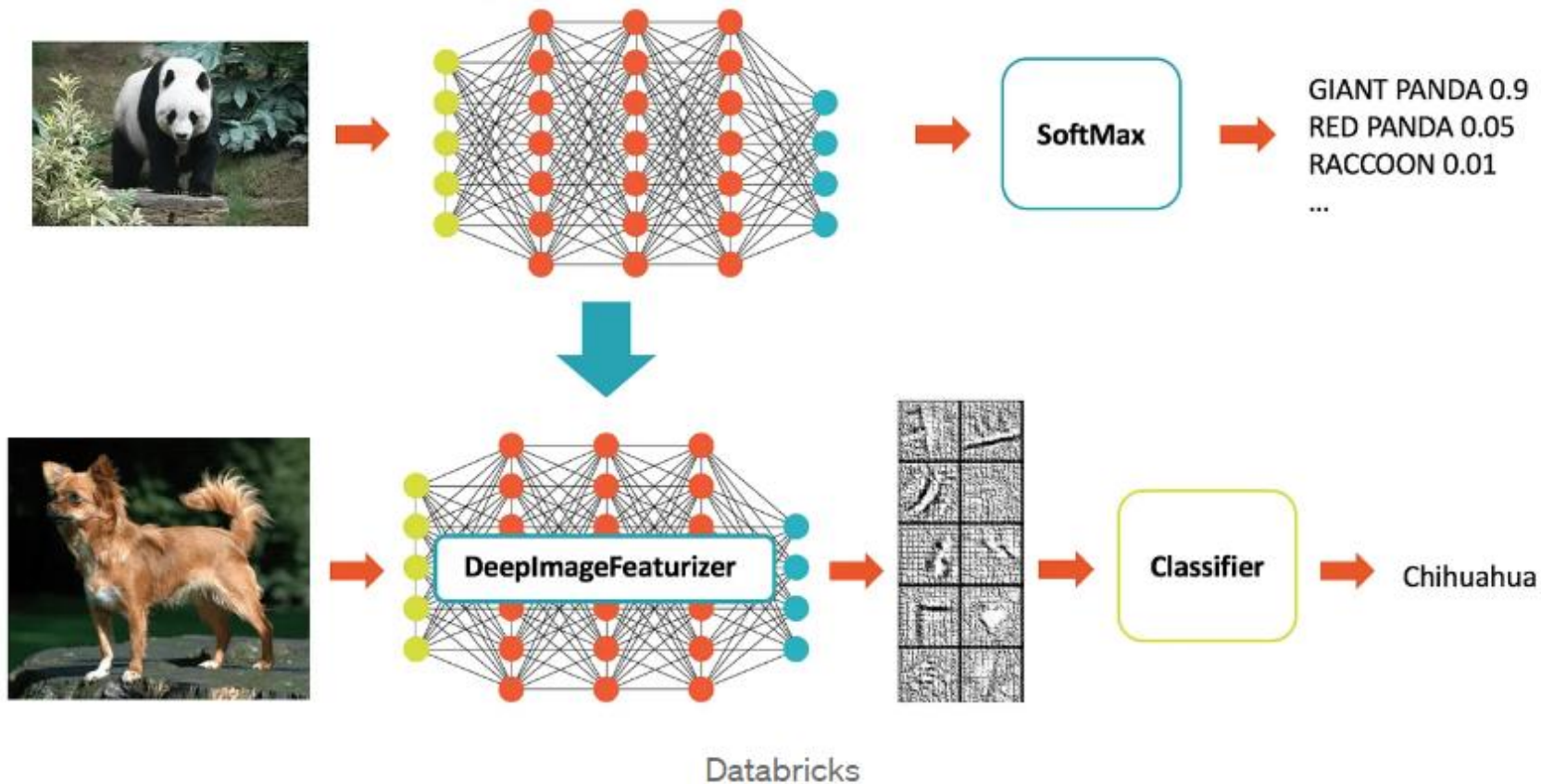
```
pip install sparkdl  
from sparkdl import readImages
```

```
[21] image_df.show()
```

```
+-----+-----+  
|          filePath|          image|  
+-----+-----+  
|file:/content/flo...|{RGB, 263, 320, 3...|  
|file:/content/flo...|{RGB, 313, 500, 3...|  
|file:/content/flo...|{RGB, 209, 320, 3...|  
+-----+-----+
```

Transfer learning

Deep Learning Pipelines enables fast transfer learning with the concept of a *Featurizer*.



Transfer learning

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from sparkdl import DeepImageFeaturizer
```



```
featurizer = DeepImageFeaturizer(inputCol="image", outputCol="features", modelName="InceptionV3")
lr = LogisticRegression(maxIter=10, regParam=0.05, elasticNetParam=0.3, labelCol="label")
p = Pipeline(stages=[featurizer, lr])

p_model = p.fit(train_df)
```

Machine Learning



References

- <https://spark.apache.org/docs/latest/ml-guide.html>
- <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bfcf/5669198905533692/3647723071348946/3983381308530741/latest.html>
- <https://github.com/FavioVazquez/deep-learning-pyspark>
- <https://towardsdatascience.com/deep-learning-with-apache-spark-part-2-2a2938a36d35>



Deep Learning