Scalable Data Processing, Legal & Ethical Foundations of Data Science (Lab3)

Ines Akaichi

20.01.2026

With contributions from: Michael S. Feurstein, Astrid Krickl & Sabrina Kirrane

Course Outline

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Title	**0842 Data Processing 2: Scalable Data Processing, Legal & Ethical Foundations of Data Science**
Instructors	Ines Akaichi, MSc., Assoz.Prof PD Dr. Sabrina Kirrane
Contact details	Sabrina.Kirrane@wu.ac.at, Ines.Akaichi@wu.ac.at
Type	PI
Weekly hours	2
Language of instruction	Englisch
Registration	09/10/25 to 11/27/25 Registration via LPIS
Notes to the course	
Subject(s)	Specialization in Business Administration Course III - Data Science
Bachelor Programs	Course III - Data Science
Dates	Course III - Data Science

Day	Date	Time	Room
Tuesday	12/02/25	01:30 PM - 05:30 PM	TC.0.04
Tuesday	12/09/25	08:30 AM - 12:00 PM	TC.-1.61 (P&S)
Tuesday	12/16/25	08:30 AM - 12:00 PM	TC.-1.61 (P&S)
Friday	12/19/25	12:00 PM - 04:00 PM	TC.0.02
Tuesday	01/13/26	08:30 AM - 12:00 PM	TC.-1.61 (P&S)
Tuesday	01/20/26	08:30 AM - 12:00 PM	TC.-1.61 (P&S)
Tuesday	01/27/26	12:00 PM - 04:00 PM	TC.1.02

Title	**1236 Data Processing 2: Scalable Data Processing, Legal & Ethical Foundations of Data Science**
Instructors	Ines Akaichi, MSc., Assoz.Prof PD Dr. Sabrina Kirrane
Contact details	Sabrina.Kirrane@wu.ac.at, Ines.Akaichi@wu.ac.at
Type	PI
Weekly hours	2
Language of instruction	Englisch
Registration	09/10/25 to 11/27/25 Registration via LPIS
Notes to the course	
Subject(s)	Specialization in Business Administration Course III - Data Science
Bachelor Programs	Course III - Data Science
Dates	Course III - Data Science

Day	Date	Time	Room
Tuesday	12/02/25	01:30 PM - 05:30 PM	TC.0.04
Tuesday	12/09/25	01:30 PM - 05:00 PM	TC.-1.61 (P&S)
Tuesday	12/16/25	01:30 PM - 05:00 PM	TC.-1.61 (P&S)
Friday	12/19/25	12:00 PM - 04:00 PM	TC.0.02
Tuesday	01/13/26	01:30 PM - 05:00 PM	TC.-1.61 (P&S)
Tuesday	01/20/26	01:30 PM - 05:00 PM	TC.-1.61 (P&S)
Tuesday	01/27/26	12:00 PM - 04:00 PM	TC.1.02

What are the attendance requirements?

Attendance requirements

According to the examination regulation full attendance is intended for a PI. Absence in one unit is tolerated if a proper reason is given.

If a student cannot attend a particular class, the student should send an email to the course instructor before the class starts, providing a legitimate justification for their absence.

Attendance is absolutely mandatory at the first course date (an unexcused absence will result in the loss of a place).

How is the course structured?

Week 1 (Sabrina Kirrane)

02.12.2025 – Lecture 1 – Introduction to Scalable Data Processing and Legal & Ethical Foundations of Data Science

Big Data, Scalability, Ethics & Intellectual Property

Week 2 (Ines Akaichi)

09.12.2025 – Lab 1 – Intro to Apache Spark

Data Manipulation with PySpark & hands on exercises

Week 3 (Ines Akaichi)

~~16.12.2025 – Lab 2 – Intro to Machine Learning using Apache Spark~~

~~Machine Learning with PySpark & hands on exercises~~

Deadline: Homework from Lab 1 ***19.12.2025*** ✔

Week 4 (Sabrina Kirrane)

19.12.2025 – Lecture 2 – Legal & Ethical Scalable Data Processing

Algorithmic Bias & Data Protection

Deadline: Project proposal ***16.1.2026 17:00*** ✔

Week 5 (Ines Akaichi)

13.01.2026 – Lab 2 – Intro to Machine Learning using Apache Spark

Machine Learning with PySpark & hands on exercises

Week 6 (Ines Akaichi)

20.01.2026 – Lab 3 & 4 – Apache Kafka & Machine Learning using Apache Spark

Apache Spark, Apache Kafka & Sentiment Analysis

Week 7 (Sabrina Kirrane)

27.01.2026 – Lecture 3 – Big Data Legal and Ethical Outlook

LLMs and AI Future Trends

Deadline: Final project submission ***13.02.2026***

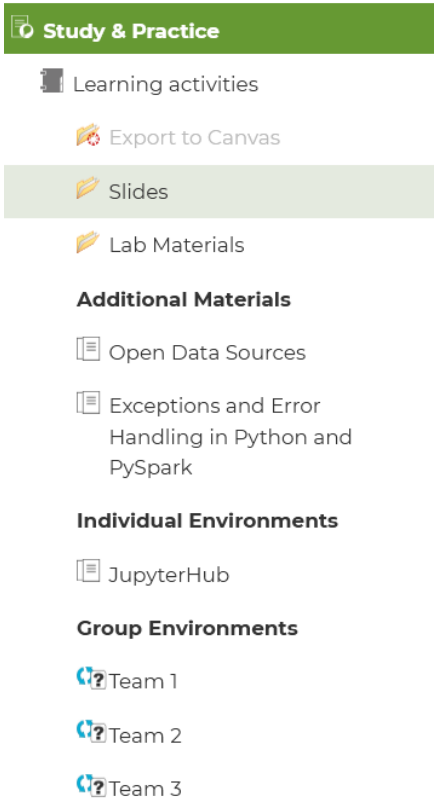
Lab 3 AM Kafka and Stream Processing (Rough Schedule)

8:30 – 9:00	Introduction to lab
9:00– 10:00	Sentiment Analysis and Topic Modeling using Reddit Data
10:00 – 10:15	Break
10:15 – 11:30	Deep dive into Apache Kafka & Stream Processing Dealing with real world data using an API for real rime flood-monitoring

Lab 3 PM Kafka and Stream Processing (Rough Schedule)

1:30 – 2:00	Introduction to lab
2:00 – 3:00	Sentiment Analysis and Topic Modeling using Reddit Data
3:00 – 3:15	Break
3:15 – 4:30	Deep dive into Apache Kafka & Stream Processing Dealing with real world data using an API for real rime flood-monitoring

Group Environments



Make sure that all the team members have access to your group environments

You find your respective group's environment on Learn

Sentiment Analysis

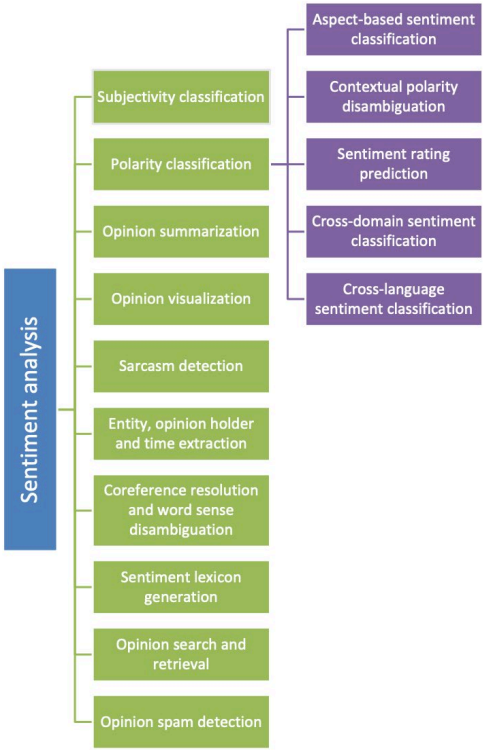
Sentiment Analysis
Introduction

- Also known as „*opinion mining*“

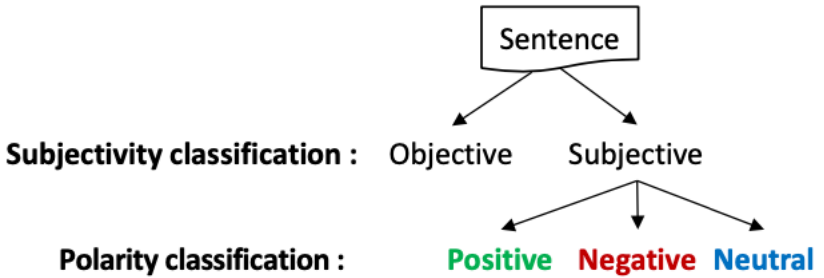
- The aim of sentiment analysis is to define automatic tools able to **extract subjective information from texts in natural language**, such as **opinions** and **sentiments**, so as to create structured and actionable knowledge to be used by either a decision support system or a decision maker.

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#overview

Sentiment Analysis
Tasks



- Objective versus Subjective classification
- If it's objective → OK
- If it's subjective → Polarity classification



Sentiment Analysis
Tasks

- ***Subjectivity classification*** is the task that distinguishes sentences that express objective (or factual) information (objective sentences) from sentences that express subjective views and opinions (subjective sentences)

→ **Objective Sentence:** „*The iPhone is a smartphone*"

→ **Subjective Sentence:** „The iPhone is awesome!"

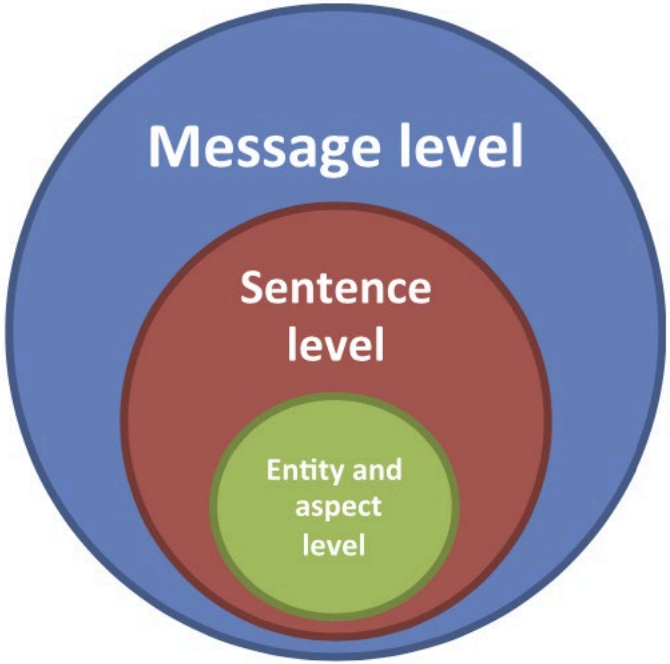
- **Polarity classification** is the task that distinguishes sentences that express positive, negative, or neutral polarities.

Sentiment Analysis
Levels of Analysis

- What do we mean by „text"
- What's our scope?

- Three levels of analysis

1. **Message:** classify polarity of specific message (paragraph of text)
2. **Sentence:** determine polarity of sentence contained in a text message
3. **Entity and aspect:** finer-grained analysis by splitting up sentence



Sentiment Analyis
Challenges

- Contrastive conjunction – two contradictory words
  - “The weather was amazing, but the dogs were annoying”
- Named-entity recognition
  - Does “2012” refer to the year or the movie
- Anaphora resolution – To what does the pronoun refers to
  - “We went for a lunch and went to a concert. It was amazing”
- Sarcasm
  - “I am so happy the dog barked all night”

https://brand24.com/blog/sentiment-analysis/

Sentiment Analyis – How it works

- Automatic
  - Create or get a dataset with labeled data
  - Use machine learning
  - The more data the more accurate
  - Less transparent, but can handle more data than rule-based algorithm

- Rule-based
  - Lexicon: positive and negative strings of words
  - See which words dominate
  - Transparent and easy to implement

- Hybrid
  - Combination of both approaches
  - Most effective: high accuracy of machine learning, stability from rules/lexicon

https://brand24.com/blog/sentiment-analysis/

Sentiment Analyis – Use cases

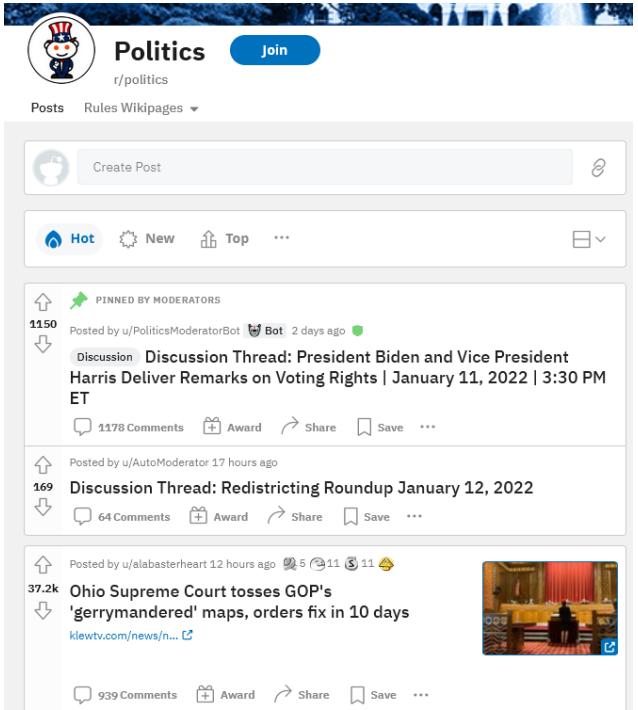
- E-commerce
- Marketing
- Advertising
- Politics
- Market research (like, dislike, expectations)
- Other research
- Crisis prevention (shit storm on United Airlines)

- Uber: Monitor if users like the new version of the app

https://brand24.com/blog/sentiment-analysis/

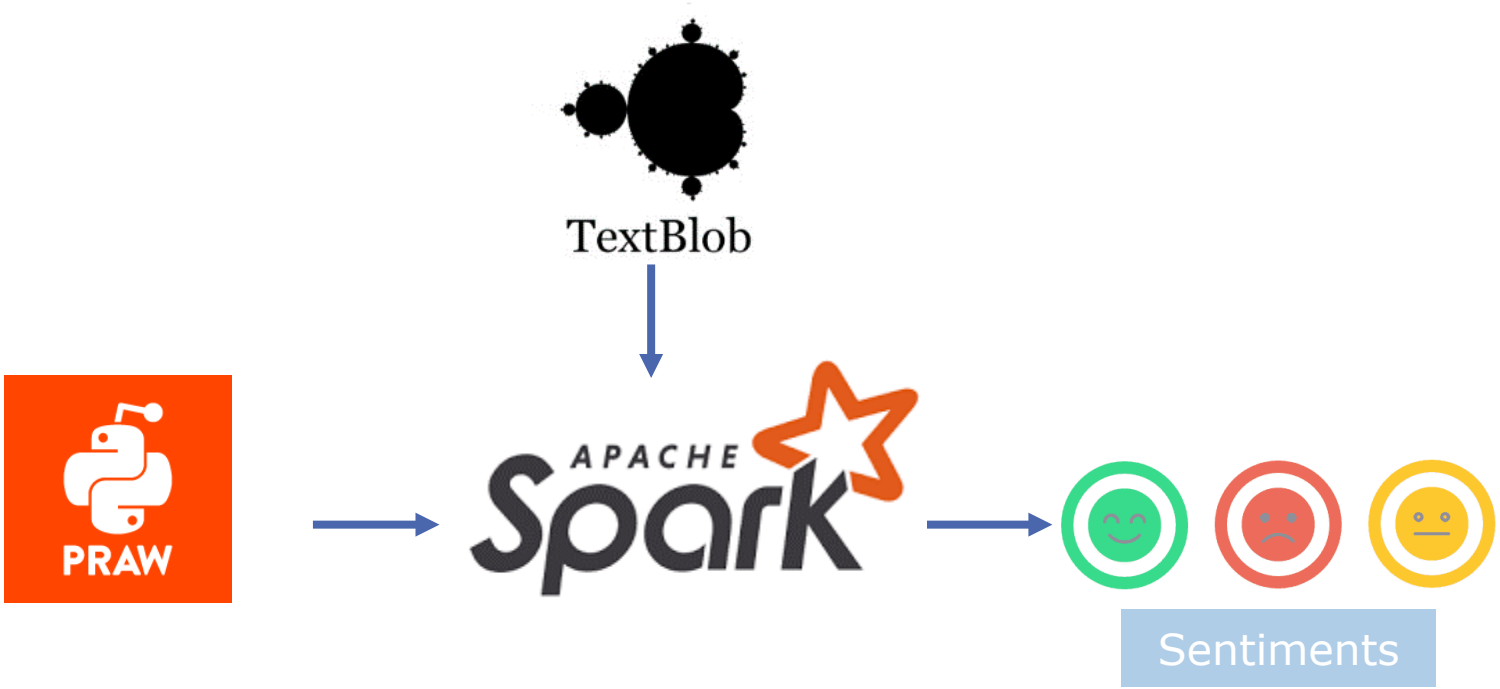
Use Case – Sentiments in Reddit Posts

- Analyse sentiments in Reddit posts
- Sentiments in r/politics



https://towardsdatascience.com/sentiment-analysis-on-streaming-twitter-data-using-spark-structured-streaming-python-fc873684bfe3

Use Case - Architecture



https://towardsdatascience.com/sentiment-analysis-on-streaming-twitter-data-using-spark-structured-streaming-python-fc873684bfe3
https://www.bmc.com/blogs/working-streaming-twitter-data-using-kafka/

Sentiment Analysis hands on exercise

The TextBlob Python API

TextBlob

Star 7,708

TextBlob is a Python (2 and 3) library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and more.

Useful Links

TextBlob @ PyPI
TextBlob @ GitHub
Issue Tracker

Table of Contents

API Reference

Blob Classes

Wrappers for various units of text, including the main **TextBlob**, **Word**, *and* **WordList** *classes.*
Example usage:

```
>>> from textblob import TextBlob
>>> b = TextBlob("Simple is better than complex.")
>>> b.tags
[(u'Simple', u'NN'), (u'is', u'VBZ'), (u'better', u'JJR'), (u'than', u'IN'), (u'co
>>> b.noun_phrases
WordList([u'simple'])
>>> b.words
WordList([u'Simple', u'is', u'better', u'than', u'complex'])
>>> b.sentiment
(0.06666666666666667, 0.41904761904761906)
>>> b.words[0].synsets()[0]
Synset('simple.n.01')
```

Changed in version 0.8.0: These classes are now imported from `textblob` *rather than* `text.blob`.

class `textblob.blob.`**`BaseBlob`**(*text, tokenizer=None, pos_tagger=None, np_extractor=None, analyzer=None, parser=None, classifier=None, clean_html=False*) *[source]*

An abstract base class that all textblob classes will inherit from. Includes words, POS tag, NP, and word count properties. Also includes basic dunder and string methods for making objects like Python strings.

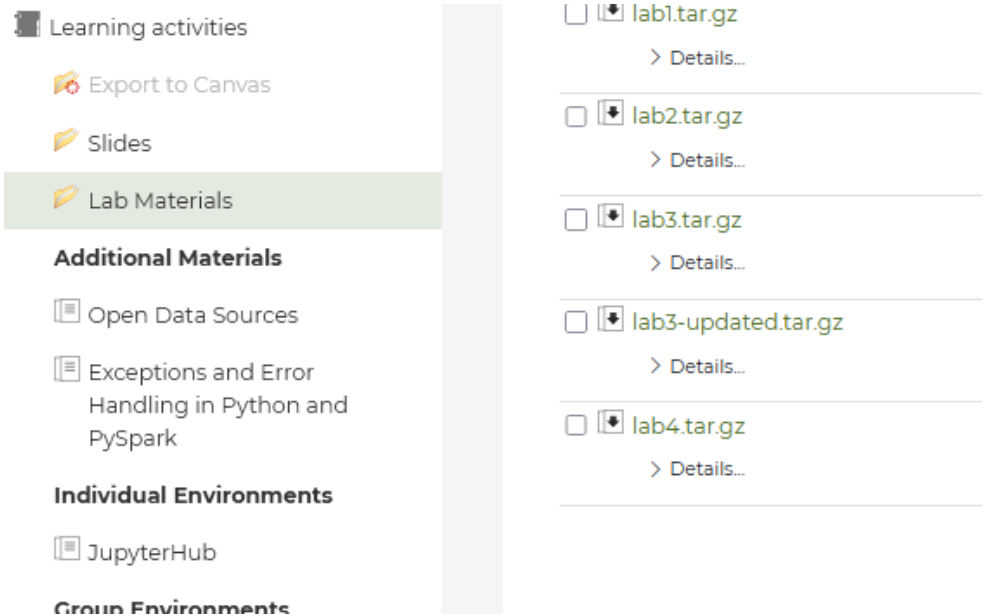
Parameters:
- ***text*** *– A string.*
- ***tokenizer*** *– (optional) A tokenizer instance. If* `None`*, defaults to* **`WordTokenizer()`**.

https://textblob.readthedocs.io/en/dev/api_reference.html

The Code

Topic modeling hands on exercise

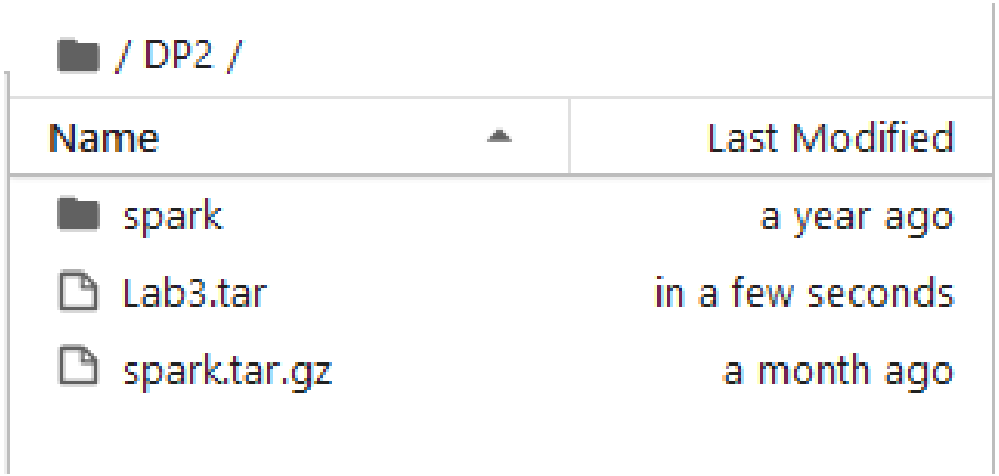
Download the code from Learn@WU



Download the code from Learn@WU

Topic modeling hands on exercise

Upload the code to your WU Jupyter notebook

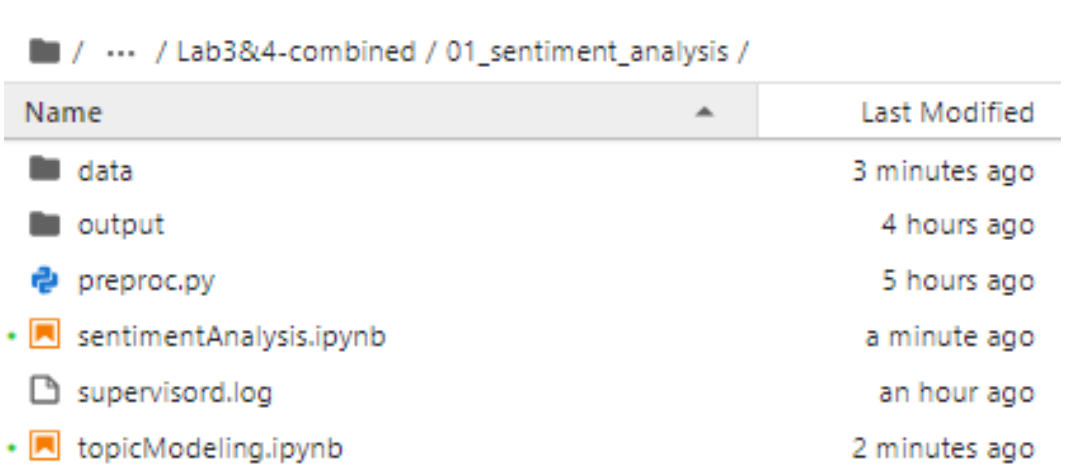


Upload the code and use the tar command to extract the code

```
jovyan@jupyter-iakaichi:~/DP2WS2024$ tar -xzvf lab4.tar.gz
```

Apache Kafka hands on exercise

Browse to the sentiment-analysis directory



Browse to the Lab3 directory

Then browse to the first directory

Apache Kafka hands on exercise

Let's take a look at the sentiment analysis

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Sentiment Analysis

Original Author: Elena Stamatelou.

Additional Info: Sentiment analysis on streaming twitter data using Spark Structured Streaming & Python. https://github.com/stamatelou/twitter_sentiment_analysis

Last Modified: By Ines Akaichi on the 19.01.2026

```
# Install textblob for the sentiment analysis
import sys
!{sys.executable} -m pip install -U textblob --no-cache-dir
```

Remember to give credit where credit is due

Apache Kafka hands on exercise

Let's take a look at the sentiment analysis

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Sentiment Analysis

Original Author: Elena Stamatelou.

Additional Info: Sentiment analysis on streaming twitter data using Spark Structured Streaming & Python. https://github.com/stamatelou/twitter_sentiment_analysis

Last Modified: By Ines Akaichi on the 19.01.2026

```
# Install textblob for the sentiment analysis
import sys
!{sys.executable} -m pip install -U textblob --no-cache-dir
```

Install the textblob library

Apache Kafka hands on exercise

Let's take a look at the sentiment analysis

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install langid --no-cache-dir
```

```
Collecting langid
  Downloading langid-1.1.6.tar.gz (1.9 MB)
     |████████████████████████████████| 1.9 MB 7.0 MB/s eta 0:00:01
Requirement already satisfied: numpy in /opt/conda/lib/python3.8/site-packages (from langid) (1.19.4)
Building wheels for collected packages: langid
  Building wheel for langid (setup.py) ... done
  Created wheel for langid: filename=langid-1.1.6-py3-none-any.whl size=1941189 sha256=7200ab7a6e1eef152ce07daa5f876af98f9cdcd6aa5
f4df4a40704196638bd5e
  Stored in directory: /tmp/pip-ephem-wheel-cache-bdhjh5pu/wheels/c5/01/a4/0160c55074707b535a6757a541842817d530d8080ca943a107
Successfully built langid
Installing collected packages: langid
Successfully installed langid-1.1.6
```

[3]:
```
# Install a pip package in the current Jupyter kernel
!{sys.executable} -m pip install nltk --no-cache-dir
```

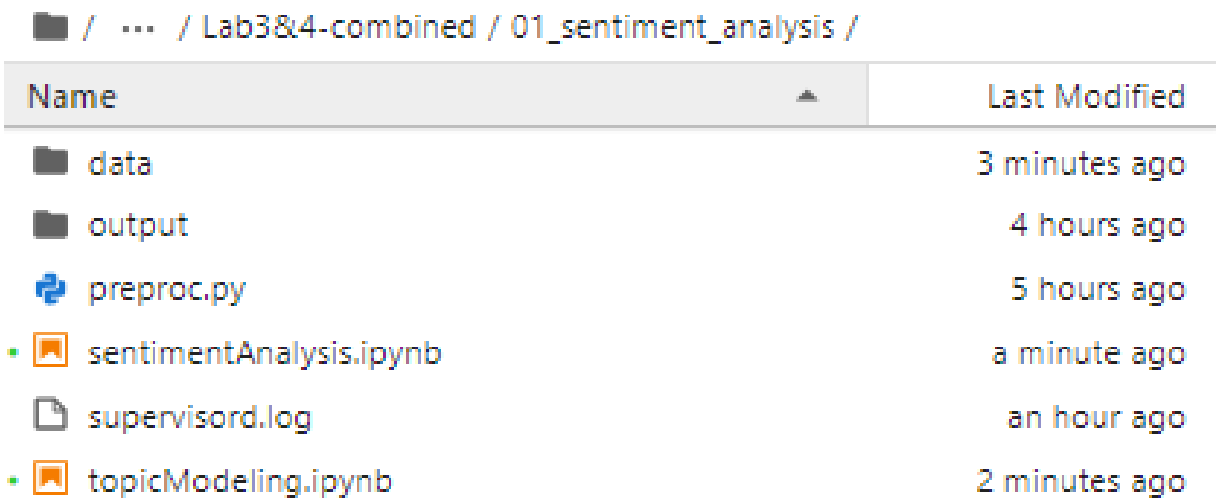
```
Requirement already satisfied: nltk in /opt/conda/lib/python3.8/site-packages (3.9.1)
Requirement already satisfied: joblib in /opt/conda/lib/python3.8/site-packages (from nltk) (0.17.0)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.8/site-packages (from nltk) (2024.11.6)
Requirement already satisfied: click in /opt/conda/lib/python3.8/site-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.8/site-packages (from nltk) (4.51.0)
```

Install the NLP related packages

[4]:
```
# Download averaged_perceptron_tagger from the nltk
import nltk
nltk.download('averaged_perceptron_tagger')
```

Apache Spark hands on exercise

Let's take a look at the sentiment analysis



Open the sentimentAnalysis notebook

Apache Spark hands on exercise

Let's take a look at the sentiment analysis


```
# Import the findspark module
import findspark

# Initialize via the full spark path
findspark.init("/usr/local/spark/")
```

```
# create Spark session
spark = SparkSession.builder \
    .master("local[*]") \
    .appName("RedditSentimentAnalysis") \
    .config("spark.executor.memory", "1gb") \
    .getOrCreate()
```

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

```python
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

# Define the schema of the dataframe
schema = StructType([
    StructField("id", StringType(), True),
    StructField("user", StringType(), True),
    StructField("comment", StringType(), True),
    StructField("parent", StringType(), True),
    StructField("stickied", StringType(), True),
    StructField("upvotes", StringType(), True),
    StructField("created", StringType(), True)
])

try:

#quote="\"" tells Spark that text inside " " is a single column even if it contains commas.
#escape="\"" allows escaping quotes inside the text.
#multiLine=true handles comments that contain line breaks (common in Reddit).

#Read data into the dataframe
    df = spark.read \
    .option("header", "true") \
    .option("sep", ",") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .option("multiLine", "true") \
    .schema(schema) \
    .csv(".data/reddit_stream_comments_politics.csv")

except:
    print("Unexpected error:", sys.exc_info()[0])
```

Define the schema of the DataFrame

- The options:
  - quote: Defines the character used to wrap fields (here "), allowing separators or newlines inside quoted values.
  - escape: Defines how to escape the quote character inside a quoted field (i.e., a string).
  - multiLine: Allows a single CSV record to span multiple lines when fields contain line breaks.

Read reddit comments from the CSV file into a Spark dataframe

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

```
# Import Sparksession and sql functions
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql import functions as F
# Import TextBlob
from textblob import TextBlob
# Import the udf, StringType, and pp modules
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

# Invoke the user-defined function `pp.check_lang` within the Spark UDF.
# Refer to https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.udf.html
# `pp.check_lang` is used to classify the language of our input text
check_lang_udf = udf(pp.check_lang, StringType())

# Invoke the user-defined function `pp.remove_stops` within the Spark UDF.
# Stop words usually refer to the most common words in a language, there is no single universal list of stop words used
# by all natural language processing tools.
# Reduces Dimensionality
# removes stop words of a single Tweets (cleaned_str/row/document)
remove_stops_udf = udf(pp.remove_stops, StringType())

# Invoke the user-defined function `pp.remove_features` within the Spark UDF.
# catch-all to remove other 'words' that I felt didn't add a lot of value
# Reduces Dimensionality, gets rid of a lot of unique urls
remove_features_udf = udf(pp.remove_features, StringType())

# Invoke the user-defined function `pp.tag_and_remove` within the Spark UDF.
# Process of classifying words into their parts of speech and labeling them accordingly is known as part-of-speech
# tagging, POS-tagging, or simply tagging. Parts of speech are also known as word classes or lexical categories. The
# collection of tags used for a particular task is known as a tagset. Our emphasis in this chapter is on exploiting
# tags, and tagging text automatically.
# http://www.nltk.org/book/ch05.html
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())

# Invoke the user-defined function `pp.lemmatize` within the Spark UDF.
# Tweets are going to use different forms of a word, such as organize, organizes, and
# organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy,
# democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these
# words to return documents that contain another word in the set.
# Reduces Dimensionality and boosts numerical measures like TFIDF
# http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
# lemmatization of a single Tweets (cleaned_str/row/document)
lemmatize_udf = udf(pp.lemmatize, StringType())

# Invoke the user-defined function `pp.check_blanks` within the Spark UDF.
# check to see if a row only contains whitespace
check_blanks_udf = udf(pp.check_blanks, StringType())
```

Import Spark, texblob and SQL functions

Preprocessing the reddit comments using the *preproc* functions

Register the *preproc* functions using Spark user defined functions (udf)

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```
from pyspark.sql.functions import coalesce, lit, concat, col
from pyspark.ml.feature import Tokenizer

# Text preprocessing

# remove columns with null

df = df.filter(col("parent").isNotNull())

# Use 'check_lang_udf' to predict language and filter out those with less than 90% chance of being English.

lang_df = df.withColumn("lang", check_lang_udf(df["parent"]))
en_df = lang_df.filter(lang_df["lang"] == "en")

# Use 'remove_stops_udf' to remove stop words to reduce dimensionality
rm_stops_df = en_df.withColumn("stop_text", remove_stops_udf(en_df["parent"]))

# Use 'remove_features_udf' to remove other non essential words, think of it as my personal stop word list
rm_features_df = rm_stops_df.withColumn("feat_text", remove_features_udf(rm_stops_df["stop_text"]))

# Use 'tag_and_remove_udf' to tag the words remaining and keep only Nouns, Verbs and Adjectives
tagged_df = rm_features_df.withColumn("tagged_text", tag_and_remove_udf(rm_features_df["feat_text"]))

# Use lemmatize_udf' to lemmatize the remaining words in order to reduce dimensionality & boost measures
lemm_df = tagged_df.withColumn("lemm_text", lemmatize_udf(tagged_df["tagged_text"]))

# Use 'check_blanks_udf' to remove all rows containing only blank spaces
check_blanks_df = lemm_df.withColumn("is_blank", check_blanks_udf(lemm_df["lemm_text"]))
no_blanks_df = check_blanks_df.filter(check_blanks_df["is_blank"] == "False")

#no_blanks_df.show(10, truncate=False)
#no_blanks_df.printSchema()

df_data=no_blanks_df.select("parent","lemm_text")
```

Apply data pre-processing steps (see Lab 2 for more details)

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```
# text classification

# Define methods from TextBlob
def polarity_detection(text):
    return TextBlob(text).sentiment.polarity

def subjectivity_detection(text):
    return TextBlob(text).sentiment.subjectivity

# We need to create user defined functions for the Textblob methods in order to use the
def text_classification(words):
    # polarity detection
    # Define as user defined fuction to embed method in the spark environment
    polarity_detection_udf = udf(polarity_detection, StringType())
    # Append polarity to dataframe
    words = words.withColumn("polarity", polarity_detection_udf("word"))

    # subjectivity detection
    # Define as user defined fuction to embed method in the spark environment
    subjectivity_detection_udf = udf(subjectivity_detection, StringType())
    # Append subjectivity to dataframe
    words = words.withColumn("subjectivity", subjectivity_detection_udf("word"))
    return words
```

Define methods from textblob we need to use for the sentiment analysis

Turn these methods into user defined functions (udf) to use them in the Spark context

Add another column for the values

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

```
# text classification to define polarity and subjectivity
cleaned_comments = text_classification(df_data)

# show results
cleaned_comments.show(10)
```

Apply the classification onto the reddit comments

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

```
# write results into CSV for inspection
cleaned_comments.write \
    .option("header", "true") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .option("multiLine", "true") \
    .mode("overwrite") \
    .csv("output/reddit_politics_sentiments_results.csv")
```

Write the results into a csv in the directory output

- The options:
  - quote: Defines the character used to wrap fields (here "), allowing separators or newlines inside quoted values.
  - escape: Defines how to escape the quote character inside a quoted field (i.e., a string).
  - multiLine: Allows a single CSV record to span multiple lines when fields contain line breaks.

Apache Spark hands on exercise

Let's take a look at the sentiment analysis

Polarity | Subjectivity

	Okay, thank you! I didn't get it, now I do.	0.625	0.5
1	Anyone and anything that does not explicitly and vocally support Trump is called "left wing".	0.0	0.0
2	To have a well established and trained armed force make war on us? That sounds terroristic or revolutionary Madge.	0.0	0.0
3	Don't forget all the contracts to dig for oil off the coast.	0.0	0.0
4	We gave y'all a chance in 2016, it's our turn now	0.0	0.0
5	We just saw it. It was a bunch of LARPers storming the capitol and threatening to hang pence	0.0	0.0
6	places from both sides, tremendous people, the best, which means we should really listen and figure out what's going on.	0.3487619047619048	0.4288571428571428
7	rt, Gosar, Cawthorn and the rest have a fan club telling them that anger and threats is good and they want to see more of it.	0.2571428571428571	0.5449200

Look into the results

The **polarity score** is a float within the range **[-1.0, 1.0]**. A score of -1 means the words are super negative, like "disgusting" or "awful." A score of 1 means the words are super positive, like "excellent" or "best."
The **subjectivity** is a float within the range **[0.0, 1.0]** where 0.0 is very objective and 1.0 is very subjective. [1]

[1] https://textblob.readthedocs.io/en/dev/quickstart.html

Topic Modeling

Topic Modeling
Introduction

- An approach used to discover **hidden semantic patterns** portrayed by a text corpus and automatically identify topics that exist inside it.

- A type of statistical modeling that leverages **unsupervised machine learning** to analyze and identify clusters or groups of similar words within a body of text.

https://www.datacamp.com/tutorial/what-is-topic-modeling

Topic Modeling
Benefits & Use Cases

- Finds hidden topics in large text collections
- Helps quickly understand the main ideas in the text
- Automates analyzing large volumes of text
- Allows searching by topics, not just keywords
- Identifies common points in customer reviews
- Helps researchers go through extensive documents efficiently

Topic Modeling

Top Topic Modeling Techniques and Algorithms

- Latent Dirichlet Allocation (LDA) is an **unsupervised algorithm** that models a dataset, typically text, as a mixture of distinct topics

- Latent Semantic Analysis (LSA) is a **natural language processing** technique used to analyze relationships between documents and the terms they contain

https://www.datacamp.com/tutorial/what-is-topic-modeling

Topic Modeling
Latent Dirichlet Allocation (LDA)

- The number of topics to be discovered is to be specified by the user to the algorithm
- The topics are assumed to be shared by all the documents within the text corpus
- The topics are a set of words

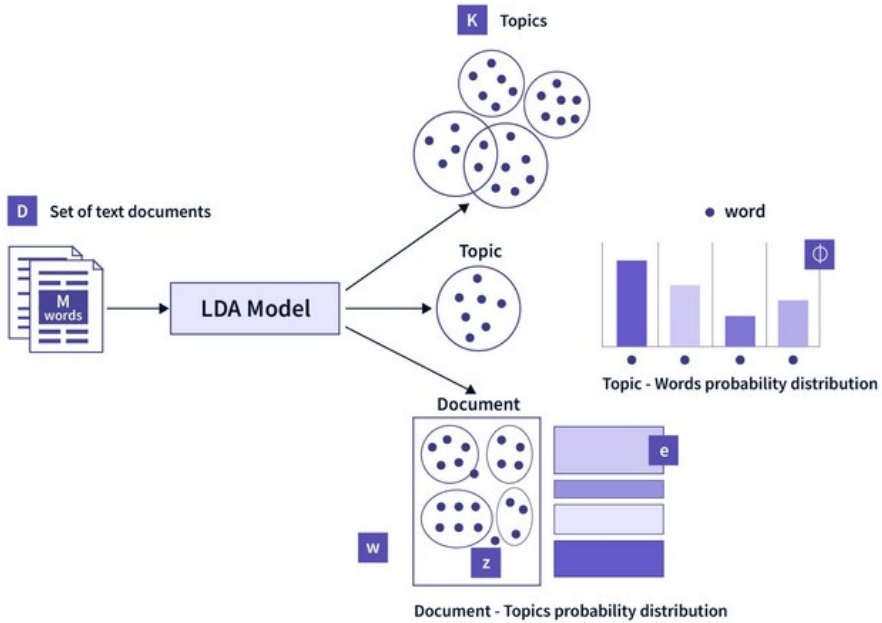
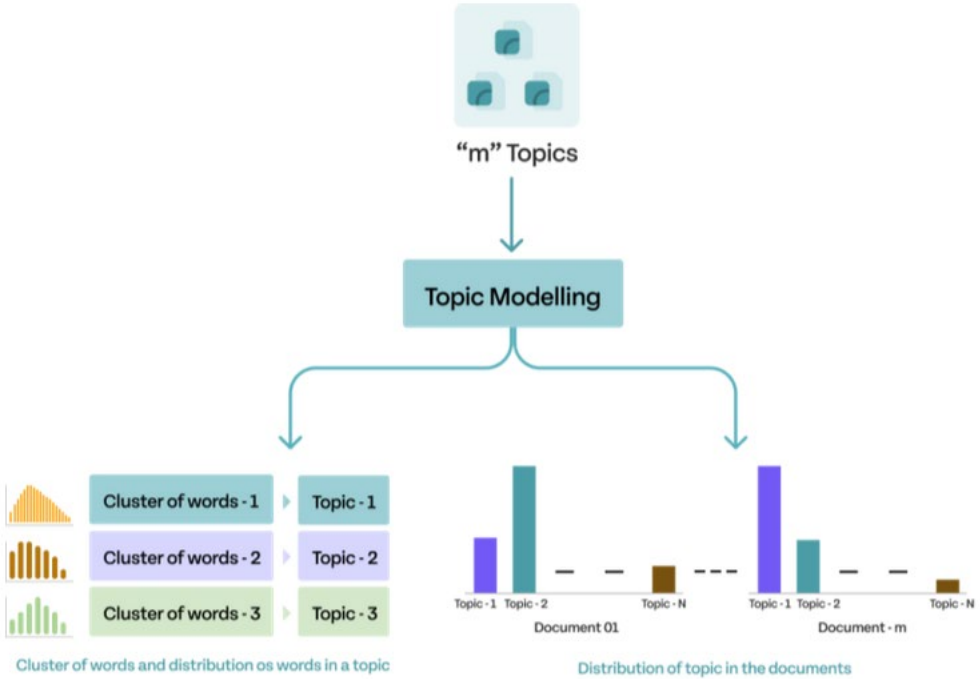


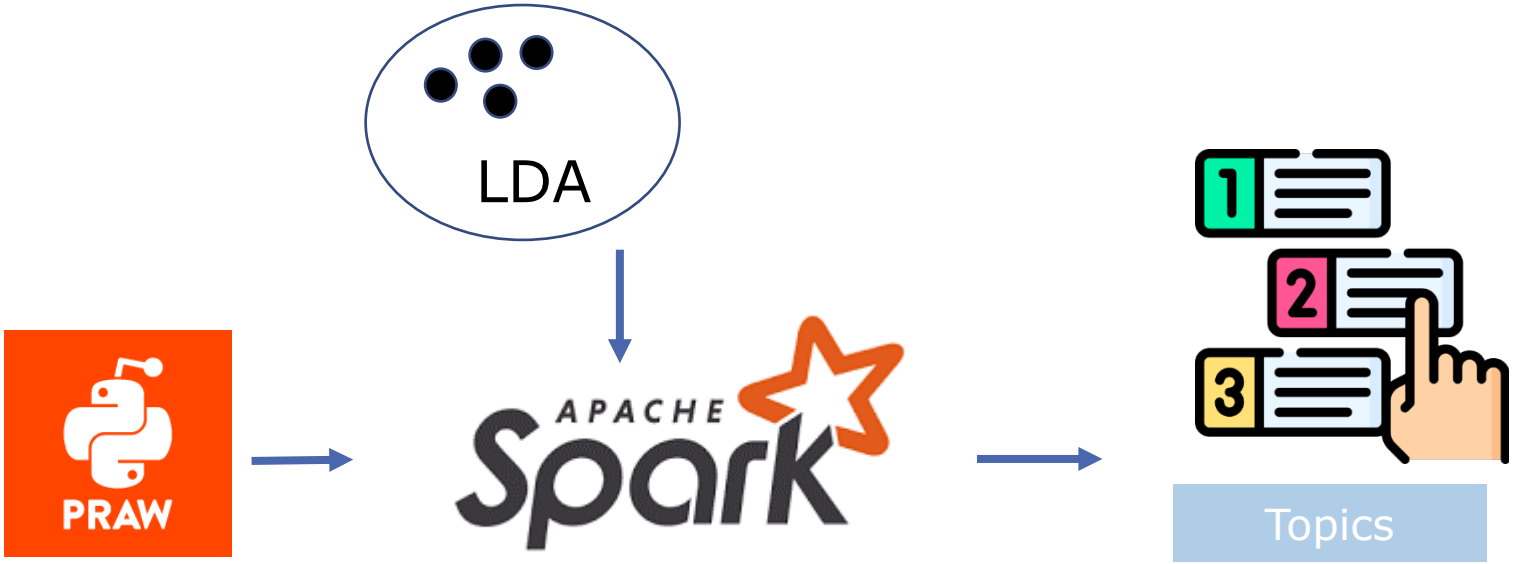
Figure is taken from https://www.scaler.com/topics/nlp/latent-dirichlet-allocation/
Explaining the algorithm behind LDA https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2

Topic Modeling
Latent Dirichlet Allocation (LDA)

- It's a generative statistical model that views each document as a mixture of various topics and, in turn, each topic as a blend of multiple words.
- This model operates under the Bayesian framework, using statistical distributions to manage the probabilities associated with topics within documents.



Use Case - Architecture

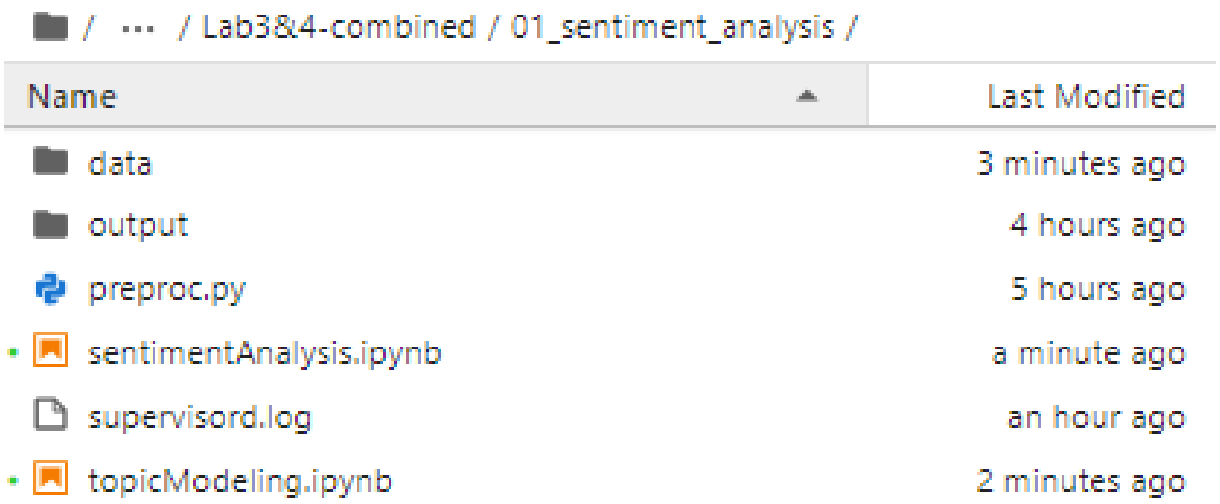


The Code

Topic Modeling with SparkML

Apache Spark hands on exercise

Let's take a look at the sentiment analysis



Open the topicMoeling notebook

Topic modeling hands on exercise

Topic modeling using LDA

Topic Analysis using Reddit Data

Original Author: Ines Akaichi

Additional Info: Natural Language Processing with Spark ML. https://github.com/dreyco676/nlp_spark.

Last Modified: By Ines Akaichi on 19.01.2026

Topic modeling hands on exercise

Topic modeling using LDA

]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install praw --no-cache-dir
```

]:
```
# Install a pip package in the current Jupyter kernel

!{sys.executable} -m pip install praw pyspark  --no-cache-dir
```

]:
```
# Install a pip package in the current Jupyter kernel
!{sys.executable} -m pip install nltk --no-cache-dir
```

]:
```
# Install a pip package in the current Jupyter kernel
!{sys.executable} -m pip install langid --no-cache-dir
```

]:
```
# Download averaged_perceptron_tagger from the nltk
import nltk
nltk.download('averaged_perceptron_tagger')
```

]:
```
# Download en averaged_perceptron_tagger from the nltk

nltk.download('averaged_perceptron_tagger_eng')
```

]:
```
# Download stopwords from the nltk

nltk.download('stopwords')
```

]:
```
# Download wordnet from the nltk

nltk.download('wordnet')
```

Topic modeling hands on exercise

Topic modeling using LDA

```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install praw --no-cache-dir
```

```
# Install a pip package in the current Jupyter kernel

!{sys.executable} -m pip install praw pyspark  --no-cache-dir
```

```
# Install a pip package in the current Jupyter kernel
!{sys.executable} -m pip install nltk --no-cache-dir
```

```
# Install a pip package in the current Jupyter kernel
!{sys.executable} -m pip install langid --no-cache-dir
```

```
# Download averaged_perceptron_tagger from the nltk
import nltk
nltk.download('averaged_perceptron_tagger')
```

```
# Download en averaged_perceptron_tagger from the nltk

nltk.download('averaged_perceptron_tagger_eng')
```

```
# Download stopwords from the nltk

nltk.download('stopwords')
```

```
# Download wordnet from the nltk

nltk.download('wordnet')
```

Topic modeling hands on exercise

Topic modeling using LDA

[ ]:
```python
# Import the findspark module
import findspark

# Initialize via the full spark path
findspark.init("/usr/local/spark/")
```

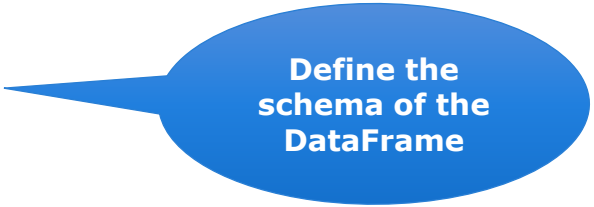
[ ]:
```python
# Import the SparkSession and SQLContext modules
from pyspark.sql import SparkSession

# Build the SparkSession
spark = SparkSession.builder \
    .master("local") \
    .appName("RedditTopicModeling") \
    .config("spark.executor.memory", "1gb") \
    .getOrCreate()
```

Create Spark session

Topic modeling hands on exercise

Topic modeling using LDA

```
# Define the schema of the dataframe
schema = StructType([
    StructField("id", StringType(), True),
    StructField("title", StringType(), True),
    StructField("selftext", StringType(), True),
    StructField("url", StringType(), True),
    StructField("author", StringType(), True),
    StructField("permalink", StringType(), True),
    StructField("score", StringType(), True),
    StructField("created_utc", StringType(), True)
])



# Read data into the dataframe
try:
    df = spark.read \
        .option("header", "true") \
        .schema(schema) \
        .csv(".data/reddit_stream_posts_politics.csv")
except:
    print("Unexpected error:", sys.exc_info()[0])

df.show (10)
```



Topic modeling hands on exercise

Topic modeling using LDA

```
# Define the schema of the dataframe
schema = StructType([
    StructField("id", StringType(), True),
    StructField("title", StringType(), True),
    StructField("selftext", StringType(), True),
    StructField("url", StringType(), True),
    StructField("author", StringType(), True),
    StructField("permalink", StringType(), True),
    StructField("score", StringType(), True),
    StructField("created_utc", StringType(), True)
])



# Read data into the dataframe
try:
    df = spark.read \
        .option("header", "true") \
        .schema(schema) \
        .csv(".data/reddit_stream_posts_politics.csv")
except:
    print("Unexpected error:", sys.exc_info()[0])

df.show (10)
```

Create a DataFrame using reddit data and the defined schema

Topic modeling hands on exercise

Topic modeling using LDA

```
# Import the udf, StringType, and pp modules
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

# Invoke the user-defined function `pp.check_lang` within the Spark UDF.
# Refer to https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.udf.html
# `pp.check_lang` is used to classify the language of our input text
check_lang_udf = udf(pp.check_lang, StringType())

# Invoke the user-defined function `pp.remove_stops` within the Spark UDF.
# Stop words usually refer to the most common words in a language, there is no single universal list of stop words used
# by all natural language processing tools.
# Reduces Dimensionality
# removes stop words of a single Tweets (cleaned_str/row/document)
remove_stops_udf = udf(pp.remove_stops, StringType())

# Invoke the user-defined function `pp.remove_features` within the Spark UDF.
# catch-all to remove other 'words' that I felt didn't add a lot of value
# Reduces Dimensionality, gets rid of a lot of unique urls
remove_features_udf = udf(pp.remove_features, StringType())

# Invoke the user-defined function `pp.tag_and_remove` within the Spark UDF.
# Process of classifying words into their parts of speech and labeling them accordingly is known as part-of-speech
# tagging, POS-tagging, or simply tagging. Parts of speech are also known as word classes or lexical categories. The
# collection of tags used for a particular task is known as a tagset. Our emphasis in this chapter is on exploiting
# tags, and tagging text automatically.
# http://www.nltk.org/book/ch05.html
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())

# Invoke the user-defined function `pp.lemmatize` within the Spark UDF.
# Tweets are going to use different forms of a word, such as organize, organizes, and
# organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy,
# democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these
# words to return documents that contain another word in the set.
# Reduces Dimensionality and boosts numerical measures like TFIDF
```

Register all the functions in Preproc with Spark Context

Topic modeling hands on exercise

Topic modeling using LDA

```
from pyspark.sql.functions import coalesce, lit, concat, col
from pyspark.ml.feature import Tokenizer

# Text preprocessing

# Use 'check_lang_udf' to predict language and filter out those with less than 90% chance of being English.

lang_df = df.withColumn("lang", check_lang_udf(df["title"]))
en_df = lang_df.filter(lang_df["lang"] == "en")

# Use 'remove_stops_udf' to remove stop words to reduce dimensionality
rm_stops_df = en_df.withColumn("stop_text", remove_stops_udf(en_df["title"]))

# Use 'remove_features_udf' to remove other non essential words, think of it as my personal stop word list
rm_features_df = rm_stops_df.withColumn("feat_text", remove_features_udf(rm_stops_df["stop_text"]))

# Use 'tag_and_remove_udf' to tag the words remaining and keep only Nouns, Verbs and Adjectives
tagged_df = rm_features_df.withColumn("tagged_text", tag_and_remove_udf(rm_features_df["feat_text"]))

# Use lemmatize_udf' to lemmatize the remaining words in order to reduce dimensionality & boost measures
lemm_df = tagged_df.withColumn("lemm_text", lemmatize_udf(tagged_df["tagged_text"]))

# Use 'check_blanks_udf' to remove all rows containing only blank spaces
check_blanks_df = lemm_df.withColumn("is_blank", check_blanks_udf(lemm_df["lemm_text"]))
no_blanks_df = check_blanks_df.filter(check_blanks_df["is_blank"] == "False")

#no_blanks_df.show(10, truncate=False)
#no_blanks_df.printSchema()

df_data=no_blanks_df.select("title","lemm_text")
```

Apply data pre-processing steps (see Lab 2 for more details)

Topic modeling hands on exercise

Topic modeling using LDA

```
# convert the column "lemm_text" to tokens and call the new column "words"
tokenizer = Tokenizer(inputCol="lemm_text", outputCol="words")
tokenizedDF = tokenizer.transform(df_data)

#tokenizedDF.printSchema()
#tokenizedDF.select("words").show(10, truncate=False)
```

Topic modeling hands on exercise

Topic modeling using LDA

```
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.clustering import LDA
from pyspark.sql.functions import col
from pyspark.sql.functions import regexp_replace

# Text preprocessing  (2)

# Convert text to feature vectors
# CountVectorizer which is used to convert a collection of text documents into a matrix of token counts (also called a document-term matrix).
# This matrix represents the frequency of terms (words) in each document (row).
# vocabSize=1000: This limits the number of most frequent words (features) to keep in the vocabulary.
# In our case, we retain the top 1000 most frequent terms across the entire dataset.
# minDF stands for minimum document frequency, and it specifies the minimum number of documents that a term must appear in to be included in the vocabulary.
vectorizer = CountVectorizer(inputCol="words", outputCol="features", vocabSize=1000, minDF=2)

# fit() trains the CountVectorizer on the data to build a model that knows how to vectorize text.
model = vectorizer.fit(tokenizedDF)

#This line is using the model that was just created by the fit() method to transform the filteredDF DataFrame into a new DataFrame (filteredDFT),
#where each document (row) is represented by a feature vector.

filteredDFT = model.transform(tokenizedDF)


filteredDFT.select("words","features").show(10, truncate=False)

filteredDFT.show(10, truncate=False)
```

Convert the tokens to feature vectors

Topic modeling hands on exercise

Topic modeling using LDA

```
# Perform LDA topic modeling
# Latent Dirichlet Allocation (LDA), a topic model designed for text documents.
# Param k defines the number of topics
# maxIter=10 means the algorithm will run for a maximum of 10 iterations when training the model.
#The purpose of the iterations is to converge to the optimal topic distribution over words.
# Param featuresCol defines the name of the input column

lda = LDA(k=4, maxIter=10, featuresCol="features")
lda_model = lda.fit(filteredDFT)

# Output topics
# describeTopics(n) is a method in the LDA model that returns the top n most significant words for each topic.
# topics will be a DataFrame that describes the topics, with each row representing a topic and containing:
#topic: The topic ID (0, 1, 2, etc.).
#termIndices: Indices of the most significant words for this topic.
#termWeights: The weight (importance) of each word in the topic.

topics = lda_model.describeTopics(10)

#model.vocabulary contains the list of words that were used to create the feature vectors for the LDA model.
#This vocabulary is ordered by the index in the feature vector.
#Each index in termIndices corresponds to a word in the vocab list. For example, if vocab[0] = "python", the index 0 will map to the word "python".

vocab = model.vocabulary
```

LDA

class pyspark.ml.clustering.**LDA**(*, featuresCol: str = 'features', maxIter: int = 20, seed: Optional[int] = None, checkpointInterval: int = 10, k: int = 10, optimizer: str = 'online', learningOffset: float = 1024.0, learningDecay: float = 0.51, subsamplingRate: float = 0.05, optimizeDocConcentration: bool = True, docConcentration: Optional[List[float]] = None, topicConcentration: Optional[float] = None, topicDistributionCol: str = 'topicDistribution', keepLastCheckpoint: bool = True) [source]

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.clustering.LDA.html

Topic modeling hands on exercise

Topic modeling using LDA

```
#loop through the topics vocabulary and return all the topics and corresponding words
print("Topics:")
for topic in topics.collect():
    words = [vocab[idx] for idx in topic['termIndices']]
    print(f"Topic {topic['topic']}: {', '.join(words)}")

# Stop Spark session
spark.stop()
```

Display topics

```
Topics:
Topic 0: inauguration, trump, biden, pardon, fauci, america, jan, live, issue, president
Topic 1: first, cryptocurrency, launch, live, coin, incoming, meme, go, state, next
Topic 2: crisis, deal, relief, major, be, disgrace, meme, think, national, come
Topic 3: trump, donald, day, say, official, end, immigrant, live, power, return
```

Expected results

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.clustering.LDA.html

Hands-on Exercise

- Load imdb_data.csv (1)
- Use TextBlob to analyse the sentiments (subjectivity) of the movie reviews
- Compare your results to the labelled data

(1) https://www.kaggle.com/datasets/columbine/imdb-dataset-sentiment-analysis-in-csv-format?resource=download

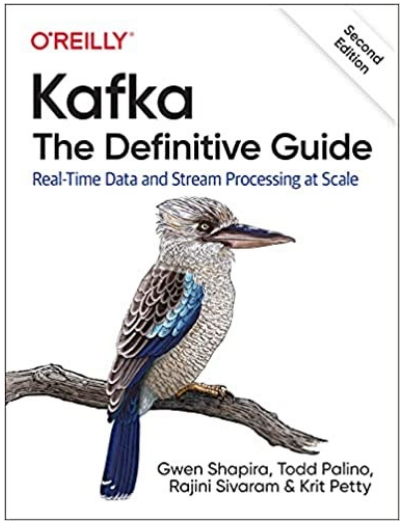
Deep dive into Apache Kafka and Stream Processing

The Apache Hadoop Ecosystem
Stream Processing

→ Concept of **DATA STREAM**

- „[...] a data stream is an abstraction representing an unbounded dataset"
- Unbounded meaning → infinite / growing / new records keep arriving

- Concept of data streams can be applied to business activities
  - Credit cards transactions
  - Stock trades
  - Package deliveries
  - Network events
  - Emails sent
  - Web server log entries



O'Reilly Book

https://kafka.apache.org/intro

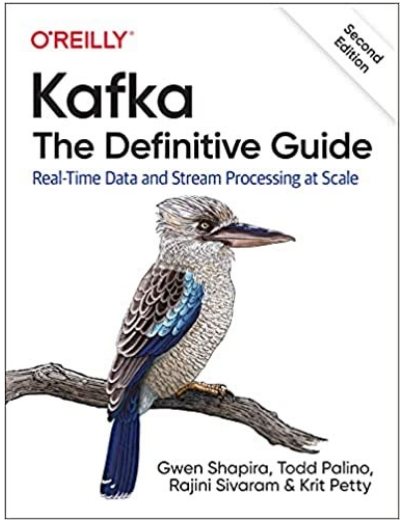
The Apache Hadoop Ecosystem
Stream Processing

→ Concept of **BATCH PROCESSING**

- "high-latency/high-throughput option"
- Read all required input (fixed input dataset) – do transformations/calculations – write all required output

→ Concept of **STREAM PROCESSING**

- „continous and nonblocking option"
- Continously read new data and compute a result



O'Reilly Book

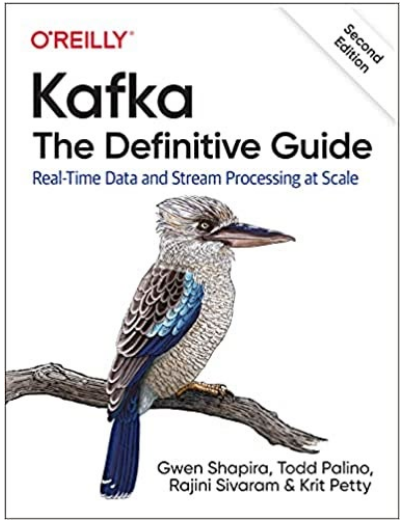
https://kafka.apache.org/intro

The Apache Hadoop Ecosystem
Stream Processing Use Cases

→ **Notifications and Alerting**
- Triggered by event (message) and leads to some action taken

→ **Real-time Reporting**
- Feed real-time system data to dashboards for monitoring

→ **Incremental ETL**
- Reduce latency when moving data into a data warehouse

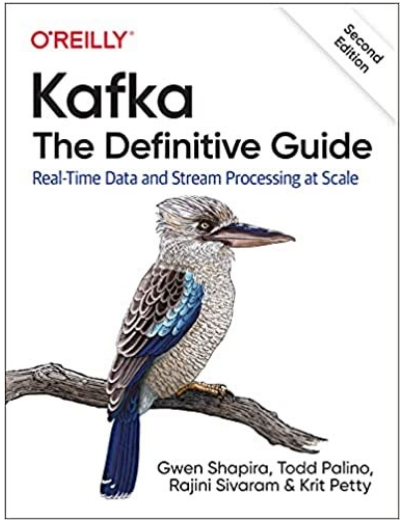


O'Reilly Book

https://kafka.apache.org/intro

The Apache Hadoop Ecosystem
Kafka

- Concept of Publish / Subscribe Messaging
- Pattern characterized by sender = **publisher** = *PRODUCER* of a piece of data (= message)
- Message is not directed to receiver
- Receiver = **subscriber** = *CONSUMER* connects to receive certain messages



O'Reilly Book

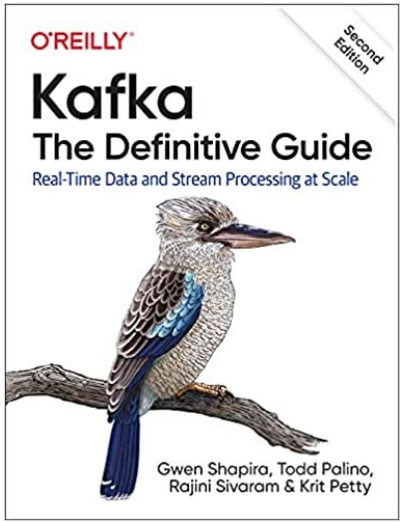
https://kafka.apache.org/intro

The Apache Hadoop Ecosystem
Kafka

→ Concept of a **MESSAGE**
- „*The unit of data within Kafka is called a message*"
- Similar to a record or row in a database table

→ Concept of a **TOPIC**
- Messages are categorized into topics
- Similar to a database table or folder in file system

→ Concept of **PARTITION**
- Messages are written to partition in append-only fashion
- A topic may consist of multiple partitions (scalability & redundancy)



O'Reilly Book

https://kafka.apache.org/intro

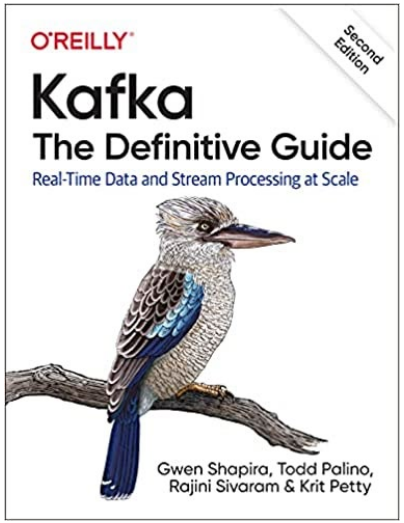
The Apache Hadoop Ecosystem
Kafka

→ Concept of a **PRODUCER**

- „[...] create[s] new messages"
- A message is produced to a certain topic
- Balanced over all partitions in topic

→ Concept of **CONSUMER**

- "[...] read messages"
- Subscribes to a topic



O'Reilly Book

https://kafka.apache.org/intro

The Apache Hadoop Ecosystem
Kafka

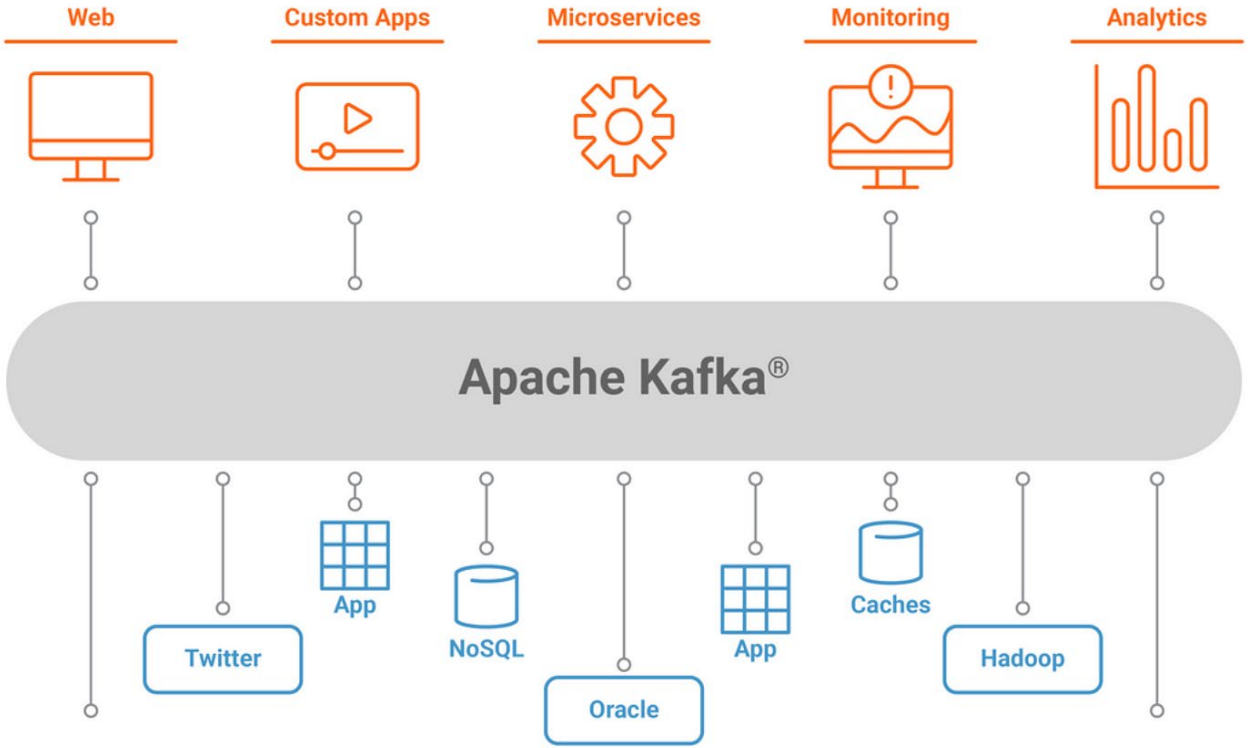
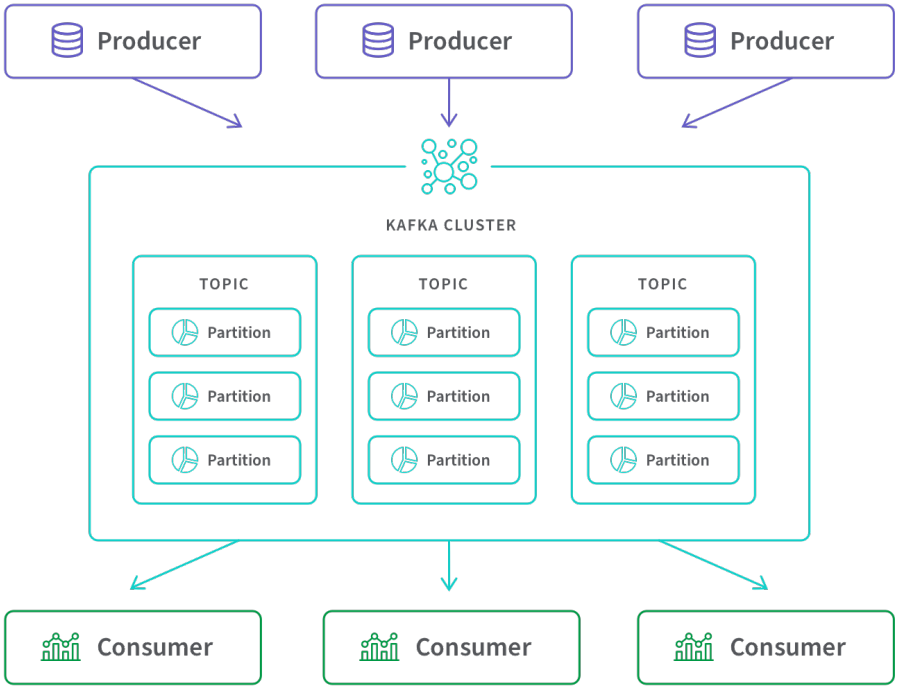


Image Source: https://medium.com/inspiredbrilliance/kafka-basics-and-core-concepts-5fd7a68c3193

The Apache Hadoop Ecosystem
Kafka

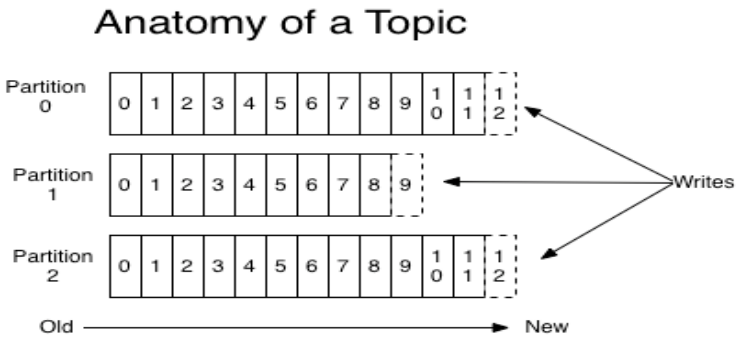


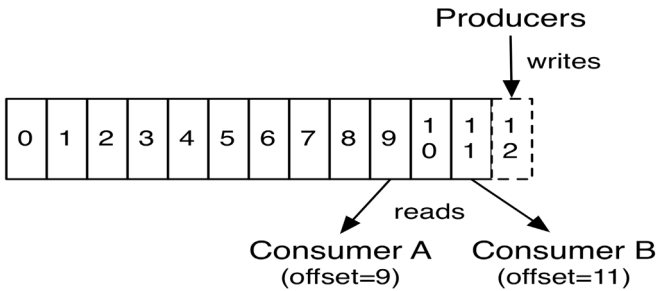
- The **Producer API** allows an application to publish a stream of records to one or more Kafka topics.
- The **Consumer API** allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The **Streams API** provides higher-level functions to process event streams, including transformations, stateful operations like aggregations and joins, windowing, processing based on event-time, and more.
- The **Connector API** allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

https://kafka.apache.org/intro

Image Source: https://www.qlik.com/us/streaming-data/apache-kafka

The Apache Hadoop Ecosystem
Kafka

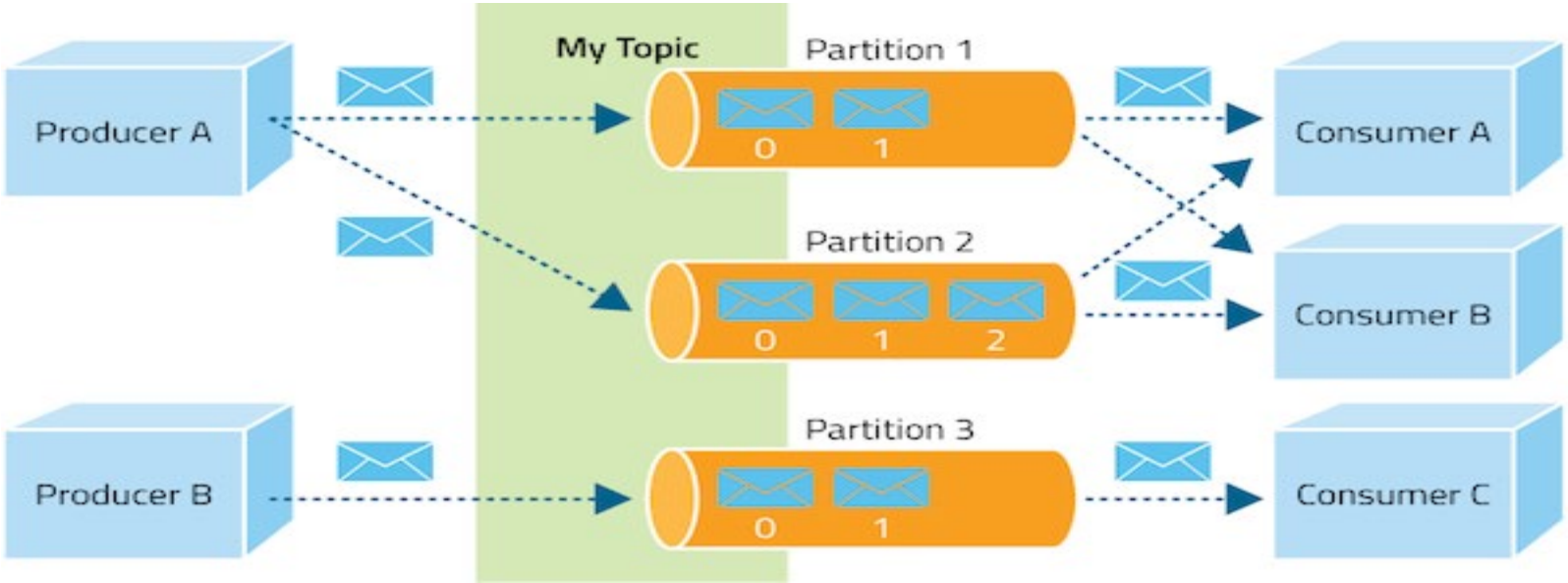




- Kafka is run as a cluster on one or more servers that can span multiple datacenters.
- The Kafka cluster stores streams of records in categories called topics.
- Each topic consists of partitions, which can be distributed to different serves (scalable)
- Messages, called records, are written by producers, get signed to a topic (automatically or by key), and are distributed to the partitions
- Each record consists of a key, a value, and a timestamp.
- Consumers read records of topics/partitions they are assigned to
- Consumers are responsible to remember an offset of the last message they read

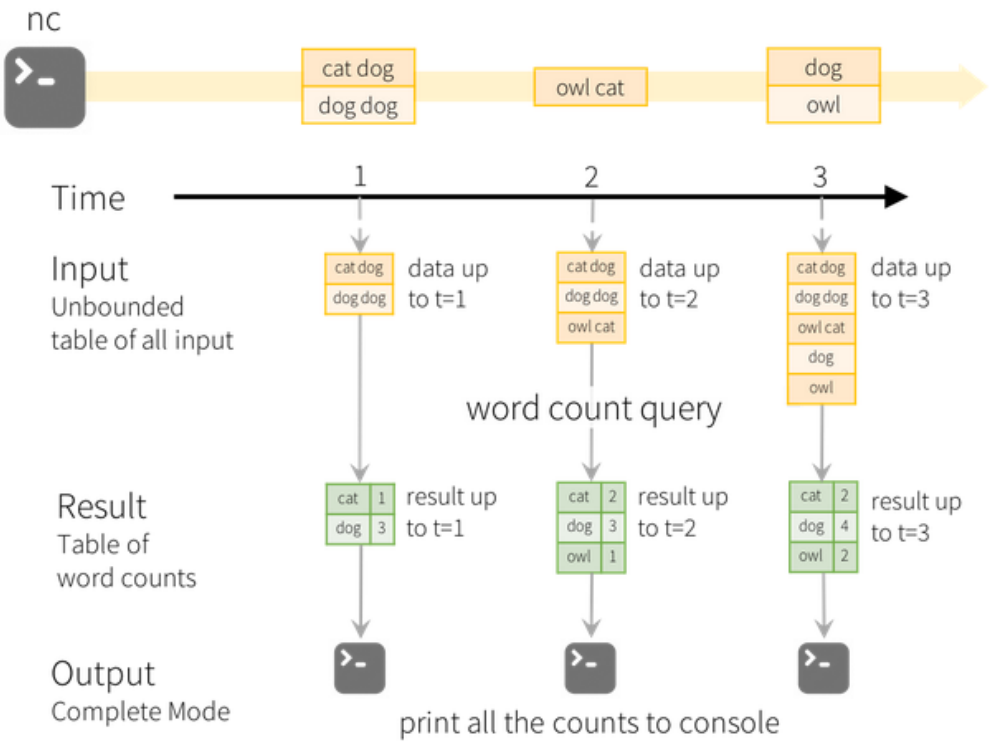
https://kafka.apache.org/intro

The Apache Hadoop Ecosystem
Kafka



https://www.cloudera.com/documentation/kafka/1-2-x/topics/kafka.html

The Apache Hadoop Ecosystem
Kafka & Spark



https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

The Apache Hadoop Ecosystem
Kafka – real world examples

- **LinkedIn** – initially created Kafka to address data pipeline problems – collect & provide user interaction and system metrics for real time recommendations and actions

- **Netflix** - apply recommendations in real-time while watching shows

- **Uber** - gather user, taxi and trip data in real-time to calculate forecast demand, and according pricing in real-time

- **Geizhals** – gather price data and update in real-time

https://kafka.apache.org/powered-by

Apache Kafka hands on exercise
The Kafka API

DOCUMENTATION

Kafka 2.7 Documentation

Prior releases: 0.7.x, 0.8.0, 0.8.1.X, 0.8.2.X, 0.9.0.X, 0.10.0.X, 0.10.1.X, 0.10.2.X, 0.11.0.X, 1.0.X, 1.1.X, 2.0.X, 2.1.X, 2.2.X, 2.3.X, 2.4.X, 2.5.X, 2.6.X.

1. GETTING STARTED

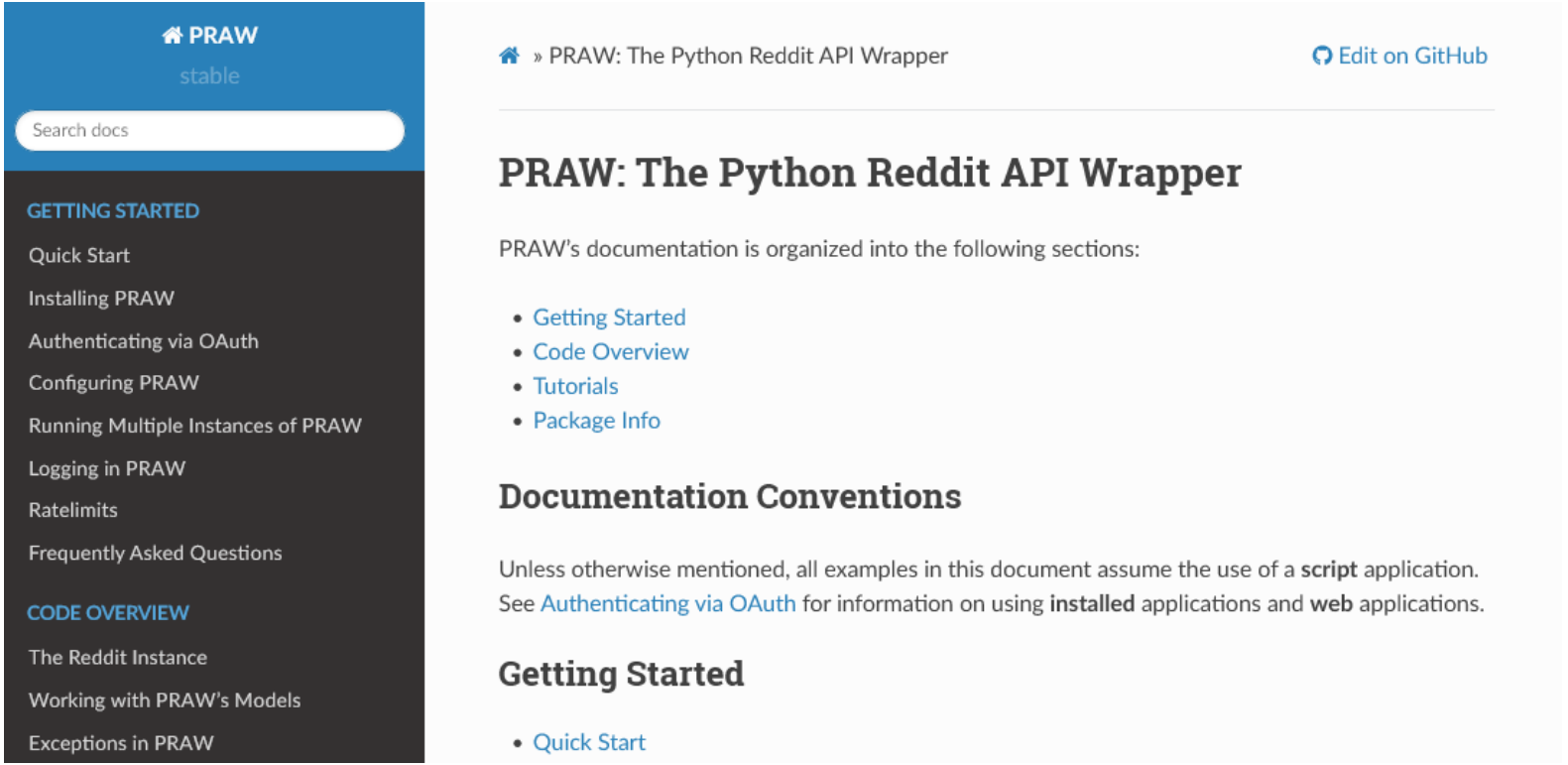
1.1 Introduction

What is event streaming?

https://kafka.apache.org/documentation/

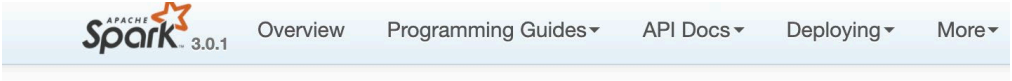
Apache Kafka hands on exercise

PRAW: The Python Reddit API Wrapper



https://praw.readthedocs.io/en/stable/

Apache Kafka hands on exercise
Spark Structured Streaming



Structured Streaming Programming Guide

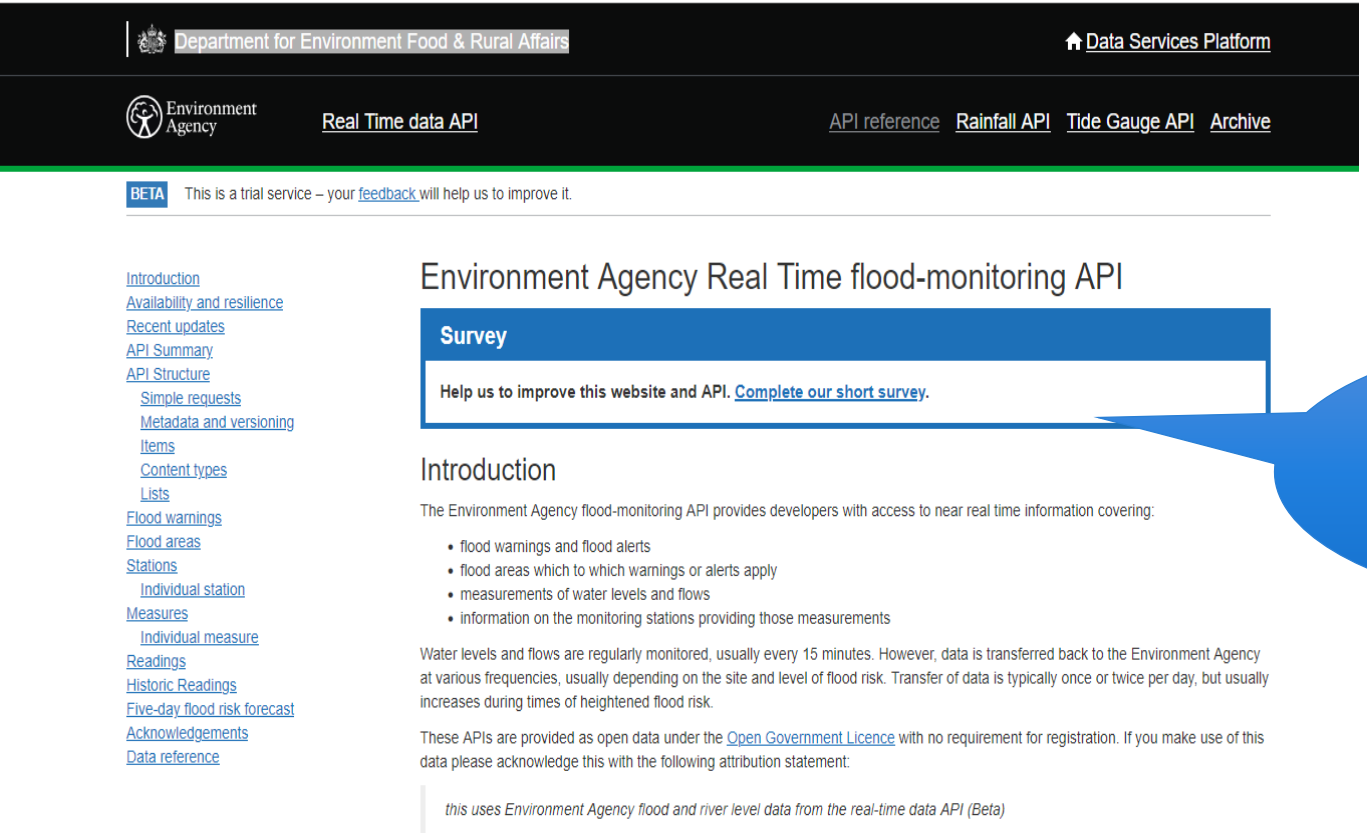
https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#overview

Kafka and Python dealing with real-time data hands on

Goal: we want to connect to the **Environment Agency Real Time flood-monitoring API** and stream the data

Apache Kafka hands on exercise

Environment Agency Real Time flood-monitoring

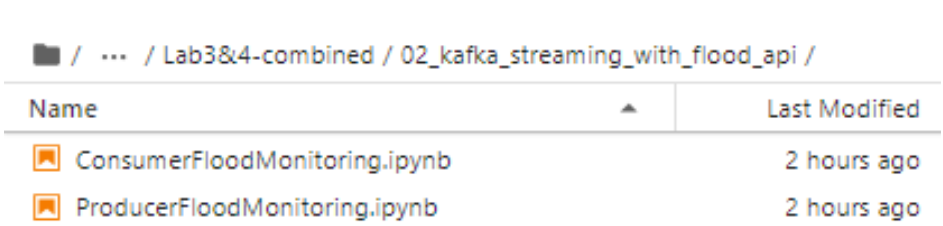


We can use Kafka to process data about flood warning, measurements of water level, etc. in real-time

The Code

Apache Kafka hands on exercise

Browse to the Kafka directory



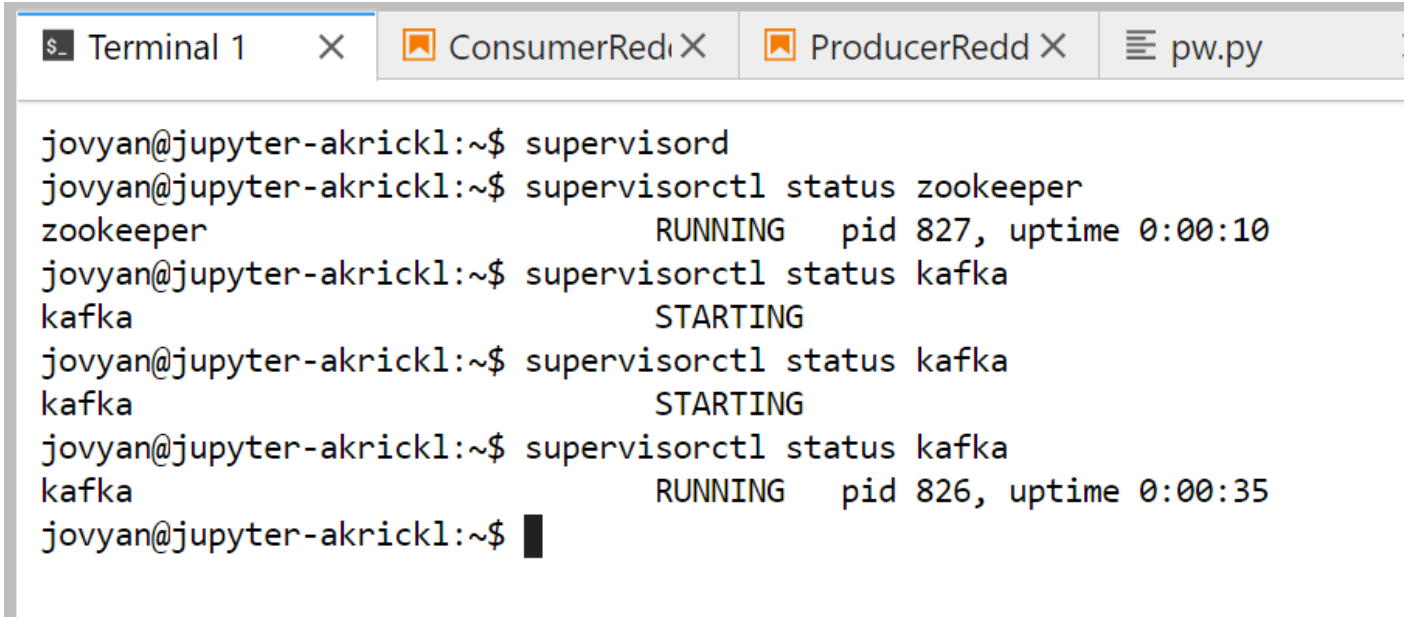
Browse to the second directory

Starting Kafka and Zookeeper

Apache Kafka hands on exercise
Starting Zookeeper and Kafka

Start via Terminal: supervisord

You need to start zookeeper and kafka

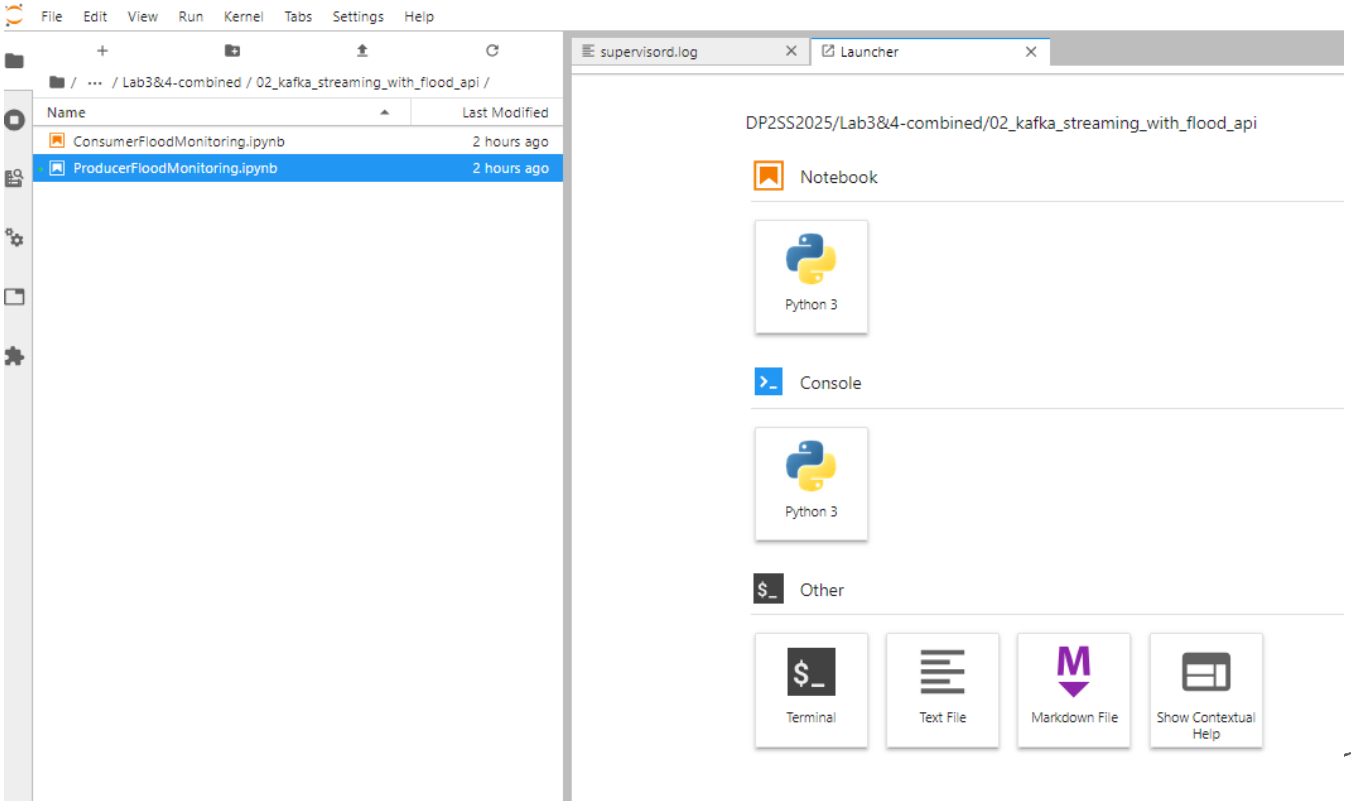


Producer


WU
WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Apache Kafka hands on exercise

Let's first take a look at the Producer



Apache Kafka hands on exercise
Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Producer

Original Author: Ines Akaichi.

Additional Info: The official website of the Environment Agency Real Time flood-monitoring API can be found here https://environment.data.gov.uk/flood-monitoring/doc/reference

Last Modified: By Ines Akaichi on the 19.01.2026

Remember to give credit where credit is due

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install kafka-python --no-cache-dir
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.8/site-packages (2.0.2)
```

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Producer

Original Author: Ines Akaichi.

Additional Info: The official website of the Environment Agency Real Time flood-monitoring API can be found here https://environment.data.gov.uk/flood-monitoring/doc/reference

Last Modified: By Ines Akaichi on the 19.01.2026

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install kafka-python --no-cache-dir
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.8/site-packages (2.0.2)
```

Install kafka-python using pip

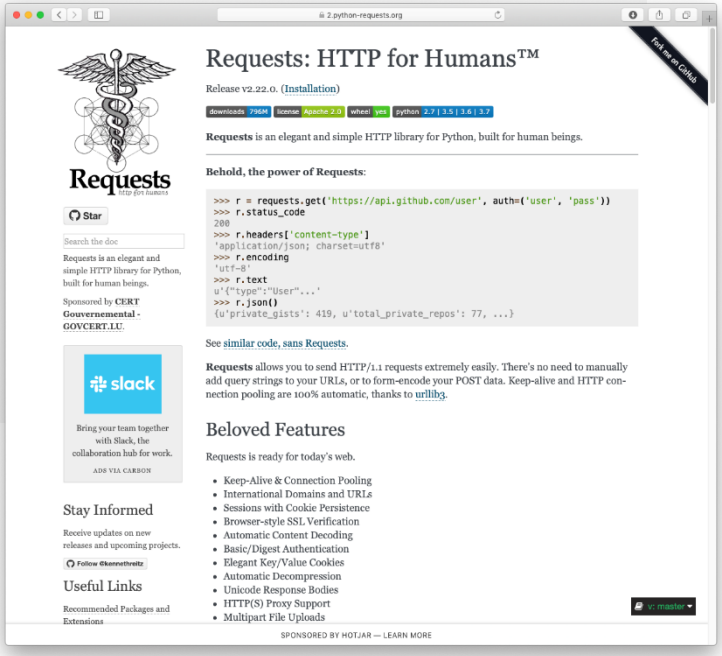
Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```
import json
import time
import requests
from kafka import KafkaProducer
from datetime import datetime

# EA Flood Monitoring API latest readings
API_URL = "https://environment.data.gov.uk/flood-monitoring/data/readings?latest"


# call the API and fetch exposed json data
# response.raise_for_status() checks the HTTP response code and raises an exception
# If the request was successful (status code 200-299), it does nothing and the code
def fetch_flood_data():
    response = requests.get(API_URL)
    response.raise_for_status()
    return response.json()
```

Import the requests module



Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

```
import json
import time
import requests
from kafka import KafkaProducer
from datetime import datetime

# EA Flood Monitoring API latest readings
API_URL = "https://environment.data.gov.uk/flood-monitoring/data/readings?latest"


# call the API and fetch exposed json data
# response.raise_for_status() checks the HTTP response code and raises an exception if the request failed (e.g. 4xx or 5xx errors).
# If the request was successful (status code 200-299), it does nothing and the code continues normally.
def fetch_flood_data():
    response = requests.get(API_URL)
    response.raise_for_status()
    return response.json()
```

Setup the URL for the API

The API is open to everyone No need for authentication

Return json data

Apache Kafka hands on exercise
Streaming flood-monitoring data: Producer

```python
import json
import time
import requests
from kafka import KafkaProducer
from datetime import datetime

# EA Flood Monitoring API latest readings
API_URL = "https://environment.data.gov.uk/flood-monitoring/data/readings?latest"


# call the API and fetch exposed json data
# response.raise_for_status() checks the HTTP response code and raises an exception if the request failed (e.g. 4xx or 5xx errors).
# If the request was successful (status code 200-299), it does nothing and the code continues normally.
def fetch_flood_data():
    response = requests.get(API_URL)
    response.raise_for_status()
    return response.json()
```

Introduction

The Environment Agency flood-monitoring API provides developers with access to near real time information covering:

- flood warnings and flood alerts
- flood areas which to which warnings or alerts apply
- measurements of water levels and flows
- information on the monitoring stations providing those measurements

You can find details about the API here

https://environment.data.gov.uk/flood-monitoring/doc/reference

Apache Kafka hands on exercise
Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Setup Kafka Producer

- Specify where Kafka is running (localhost and port)
- Pick a topic Name
- Start Stream where all flood monitoring data gets send to

[ ]:
```python
import json
import time
import requests
from kafka import KafkaProducer
from datetime import datetime

# EA Flood Monitoring API latest readings
API_URL = "https://environment.data.gov.uk/flood-monitoring/data/readings?latest"


# call the API and fetch exposed json data
# response.raise_for_status() checks the HTTP response code and raises an exception if the request failed (e.g. 4xx or 5xx errors).
# If the request was successful (status code 200-299), it does nothing and the code continues normally.
def fetch_flood_data():
    response = requests.get(API_URL)
    response.raise_for_status()
    return response.json()


# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, value_serializer=lambda v: json.dumps(v).encode('utf-8'))
```

Setup Kafka producer on port 9092

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```python
# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTR                    .encode('utf-8'))


# Poll every 15 minutes
POLL_INTERVAL = 15 * 60  # seconds

while True:

    try:
        # fetch exposed json data
        data = fetch_flood_data()
        items = data.get("items", [])
        # store the current time
        timestamp = datetime.utcnow().isoformat()
        # extract the different properties

        for item in items:
            message = {
                "datetime": item.get("dateTime"),
                "measure": item.get("measure"),
                "value": item.get("value")
            }

            #print (message)

            # send the data to kafka
            producer.send(KAFKA_TOPIC, value=message)

        producer.flush()
        # print the number of sent items at a specific time
        print(f"Sent {len(items)} records at {timestamp}")

    except Exception as e:
        print("Error:", e)

    # sleep for 15 mins
    time.sleep(POLL_INTERVAL)
```

Fetch data every 15 mins

Start streaming latest readings of water level

Note on crawling

A common requirement is for an application to maintain a copy of all the latest level/flow values from some (or all) of the measurement stations. The efficient way to do this is to issue a single call every 15 mins:

https://environment.data.gov.uk/flood-monitoring/data/readings?latest

This will retrieve the latest value for all measurements - from which you can pick the ones you are interested. Since the data changes at most every 15 mins then this single call four times an hour will ensure a complete track of all of the data.

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```python
# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, value_serializer=lambda v: json.dumps(v).encode('utf-8'))


# Poll every 15 minutes
POLL_INTERVAL = 15 * 60  # seconds

while True:

    try:
        # fetch exposed json data
        data = fetch_flood_data()
        items = data.get("items", [])
        # store the current time
        timestamp = datetime.utcnow().isoformat()
        # extract the different properties

        for item in items:
            message = {
                "datetime": item.get("dateTime"),
                "measure": item.get("measure"),
                "value": item.get("value")
            }

            #print (message)

            # send the data to kafka
            producer.send(KAFKA_TOPIC, value=message)

        producer.flush()
        # print the number of sent items at a specific time
        print(f"Sent {len(items)} records at {timestamp}")

    except Exception as e:
        print("Error:", e)

    # sleep for 15 mins
    time.sleep(POLL_INTERVAL)
```

Writing interesting data into json

EFMD EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBA ACCREDITED

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```python
# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, value_serializer=lambda v: json.dumps(v).encode('utf-8'))


# Poll every 15 minutes
POLL_INTERVAL = 15 * 60  # seconds

while True:

    try:
        # fetch exposed json data
        data = fetch_flood_data()
        items = data.get("items", [])
        # store the current time
        timestamp = datetime.utcnow().isoformat()
        # extract the different properties

        for item in items:
            message = {
                "datetime": item.get("dateTime"),
                "measure": item.get("measure"),
                "value": item.get("value")
            }

            #print (message)

            # send the data to kafka
            producer.send(KAFKA_TOPIC, value=message)

        producer.flush()
        # print the number of sent items at a specific time
        print(f"Sent {len(items)} records at {timestamp}")

    except Exception as e:
        print("Error:", e)

    # sleep for 15 mins
    time.sleep(POLL_INTERVAL)
```

Print to see what we receive

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```python
# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, value_serializer=lambda v: json.dumps(v).encode('utf-8'))


# Poll every 15 minutes
POLL_INTERVAL = 15 * 60  # seconds

while True:

    try:
        # fetch exposed json data
        data = fetch_flood_data()
        items = data.get("items", [])
        # store the current time
        timestamp = datetime.utcnow().isoformat()
        # extract the different properties

        for item in items:
            message = {
                "datetime": item.get("dateTime"),
                "measure": item.get("measure"),
                "value": item.get("value")
            }

            #print (message)

            # send the data to kafka
            producer.send(KAFKA_TOPIC, value=message)

        producer.flush()
        # print the number of sent items at a specific time
        print(f"Sent {len(items)} records at {timestamp}")

    except Exception as e:
        print("Error:", e)

    # sleep for 15 mins
    time.sleep(POLL_INTERVAL)
```

Send data to kafka topic

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

```python
# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, value_serializer=lambda v: json.dumps(v).encode('utf-8'))


# Poll every 15 minutes
POLL_INTERVAL = 15 * 60  # seconds

while True:

    try:
        # fetch exposed json data
        data = fetch_flood_data()
        items = data.get("items", [])
        # store the current time
        timestamp = datetime.utcnow().isoformat()
        # extract the different properties

        for item in items:
            message = {
                "datetime": item.get("dateTime"),
                "measure": item.get("measure"),
                "value": item.get("value")
            }

            #print (message)

            # send the data to kafka
            producer.send(KAFKA_TOPIC, value=message)

        producer.flush()
        # print the number of sent items at a specific time
        print(f"Sent {len(items)} records at {timestamp}")

    except Exception as e:
        print("Error:", e)

    # sleep for 15 mins
    time.sleep(POLL_INTERVAL)
```

Catch exceptions

Apache Kafka hands on exercise

Streaming flood-monitoring data: Producer

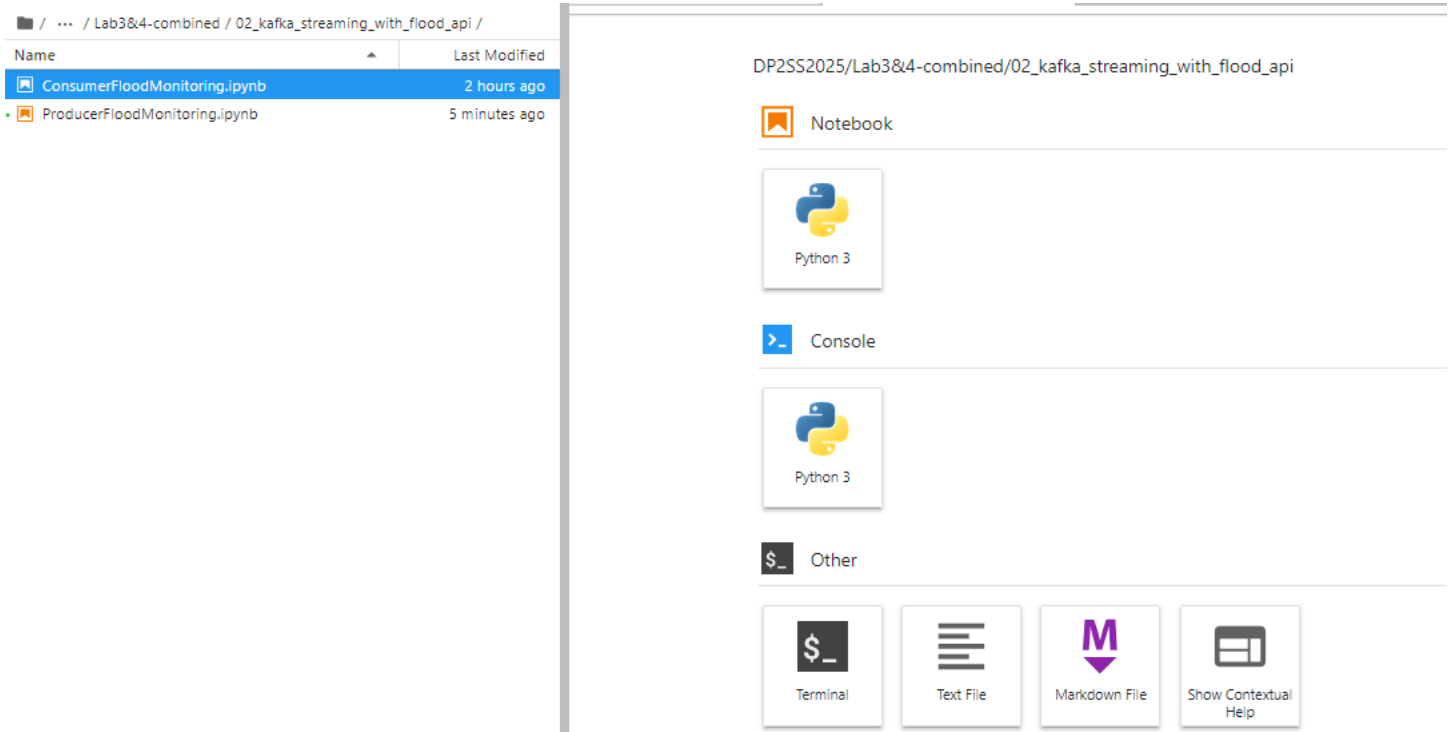
```python
# Kafka config
KAFKA_TOPIC = "flood"
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, value_serializer=lambda v: json.dumps(v).encode('utf-8'))


# Poll every 15 minutes
POLL_INTERVAL = 15 * 60  # seconds

while True:

    try:
        # fetch exposed json data
        data = fetch_flood_data()
        items = data.get("items", [])
        # store the current time
        timestamp = datetime.utcnow().isoformat()
        # extract the different properties

        for item in items:
            message = {
                "datetime": item.get("dateTime"),
                "measure": item.get("measure"),
                "value": item.get("value")
            }

            #print (message)

            # send the data to kafka
            producer.send(KAFKA_TOPIC, value=message)

        producer.flush()
        # print the number of sent items at a specific time
        print(f"Sent {len(items)} records at {timestamp}")

    except Exception as e:
        print("Error:", e)

    # sleep for 15 mins
    time.sleep(POLL_INTERVAL)
```

Fetch data every 15 mins

Data Consumer

Apache Kafka hands on exercise

Let's take a look at the Consumer



Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Use kafka python library

[1]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install kafka-python --no-cache-dir
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.8/site-packages (2.0.2)
```

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install findspark --no-cache-dir
```

```
Requirement already satisfied: findspark in /opt/conda/lib/python3.8/site-packages (1.4.2)
```

[3]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install pyspark --no-cache-dir
```

```
Requirement already satisfied: pyspark in /opt/conda/lib/python3.8/site-packages (3.5.8)
Requirement already satisfied: py4j<0.10.9.10,>=0.10.9.7 in /opt/conda/lib/python3.8/site-packages (from pyspark) (0.10.9.9)
```

[4]:
```
# Import the os module
import os

# Set the PYSPARK_SUBMIT_ARGS to the appropriate spark-sql-kafka package
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1 pyspark-shell'
```

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

[1]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install kafka-python --no-cache-dir
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.8/site-packages (2.0.2)
```

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install findspark --no-cache-dir
```

```
Requirement already satisfied: findspark in /opt/conda/lib/python3.8/site-packages (1.4.2)
```

[3]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install pyspark --no-cache-dir
```

```
Requirement already satisfied: pyspark in /opt/conda/lib/python3.8/site-packages (3.5.8)
Requirement already satisfied: py4j<0.10.9.10,>=0.10.9.7 in /opt/conda/lib/python3.8/site-packages (from pyspark) (0.10.9.9)
```

[4]:
```
# Import the os module
import os

# Set the PYSPARK_SUBMIT_ARGS to the appropriate spark-sql-kafka package
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1 pyspark-shell'
```

Install findspark using pip

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

[1]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install kafka-python --no-cache-dir
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.8/site-packages (2.0.2)
```

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install findspark --no-cache-dir
```

```
Requirement already satisfied: findspark in /opt/conda/lib/python3.8/site-packages (1.4.2)
```

[3]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install pyspark --no-cache-dir
```

```
Requirement already satisfied: pyspark in /opt/conda/lib/python3.8/site-packages (3.5.8)
Requirement already satisfied: py4j<0.10.9.10,>=0.10.9.7 in /opt/conda/lib/python3.8/site-packages (from pyspark) (0.10.9.9)
```

Install pyspark using pip

[4]:
```
# Import the os module
import os

# Set the PYSPARK_SUBMIT_ARGS to the appropriate spark-sql-kafka package
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1 pyspark-shell'
```

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

[1]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install kafka-python --no-cache-dir
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.8/site-packages (2.0.2)
```

[2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install findspark --no-cache-dir
```

```
Requirement already satisfied: findspark in /opt/conda/lib/python3.8/site-packages (1.4.2)
```

[3]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install pyspark --no-cache-dir
```

```
Requirement already satisfied: pyspark in /opt/conda/lib/python3.8/site-packages (3.5.8)
Requirement already satisfied: py4j<0.10.9.10,>=0.10.9.7 in /opt/conda/lib/python3.8/site-packages (from pyspark) (0.10.9.9)
```

[4]:
```
# Import the os module
import os

# Set the PYSPARK_SUBMIT_ARGS to the appropriate spark-sql-kafka package
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1 pyspark-shell'
```

Spark and Kafka specific configuration

Apache Kafka hands on exercise
Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Setup Spark

- **Load Spark with the respective path** Set path where Spark is installed (differences in between paths with windows / linux)
- **Create Spark Session** Either take the existing session or create a new one if there is none. Create session based on the parameters.
- **Load SparkContext** Context as main entry point for Spark functionality, which we will need later

[ ]:
```
import time
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType


spark = SparkSession.builder \
    .master("local[*]") \
    .appName("FloodMonitoringConsumer") \
    .config("spark.executor.memory", "1gb") \
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")
```

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Setup Spark Stream to read data from Kafka

- **Start Spark Stream** Initialize stream with location and to which topic to subscribe
- **Write into Spark Stream** Read data from Kafka, process it and write the results into new Spark Stream
- **Inspect Spark Stream Results** See the new results every 10 seconds

Load a kafka stream

```
6]: # Encapsulate the code in try except blocks
try:

# readStream is an interface used to load a streaming :class:`DataFrame <pyspark.sql.DataFrame>` from external
# storage systems (e.g. file systems, key-value stores, etc).
# With the option ("startingOffsets" equal to "latest"), when the query would resume, it always picks up from where the query left of
    kafka_df = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "127.0.0.1:9092") \
        .option("subscribe", "flood") \
        .option("startingOffsets", "earliest") \
        .load()
except:
    # Print the error
    print("Unexpected error:", sys.exc_info()[0])
```

```
7]: kafka_df.printSchema()
```

This is how kafka_df looks like

```
root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```
# Encapsulate the code in try except blocks
try:
    # We are only interested in the value column that contains our Reddit data that we streamed in the beginning
    # selectExpr () projects a set of SQL expressions and returns a new DataFrame.
    # In this case, the SQL expression ("CAST(value AS STRING) as json_data") is casting the 'value' column to a string type and renaming the resulting column as 'json_
    json_df = kafka_df.selectExpr("CAST(value AS STRING) as json_data")

    schema = StructType([
        StructField("datetime", StringType()),
        StructField("measure", StringType()),
        StructField("value", StringType())
    ])

    # extract the schema info from json_df
    # from_json  converts the JSON string in the column "json_data" into a Spark struct (a nested row) using the provided schema
    # alias () names the parsed struct column data
    # .select("data.*")  explodes the struct into separate columns.
    parsed_df = json_df.select(from_json("json_data", schema).alias("data")).select("data.*")

    # Interface used to write a streaming :class:`DataFrame <pyspark.sql.DataFrame>` to external
    # storage systems (e.g. file systems, key-value stores, etc).
    # Write the above data into memory. Consider the entire analysis in all iteration (output mode = complete). and let the trigger runs in every 10 secs.

    writeDF = parsed_df.writeStream. \
            format("memory"). \
            queryName("floodQuery2"). \
            trigger(processingTime='10 seconds'). \
            start()

    # Print banner text
    print("----- streaming is running -------")

    #writeDF.awaitTermination()

except:
    # Print the error
    print("Unexpected error:", sys.exc_info())
```

Extract the data stream and write it to memory

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```
# Encapsulate the code in try except blocks
try:
    # We are only interested in the value column that contains our Reddit data that we streamed in the beginning
    # selectExpr () projects a set of SQL expressions and returns a new DataFrame.
    # In this case, the SQL expression ("CAST(value AS STRING) as json_data") is casting the 'value' column to a string type and renaming the resulting column as 'json_data'
    json_df = kafka_df.selectExpr("CAST(value AS STRING) as json_data")

    schema = StructType([
        StructField("datetime", StringType()),
        StructField("measure", StringType()),
        StructField("value", StringType())
    ])

    # extract the schema info from json_df
    # from_json  converts the JSON string in the column "json_data" into a Spark struct (a nested row) using the provided schema
    # alias () names the parsed struct column data
    # .select("data.*")  explodes the struct into separate columns.
    parsed_df = json_df.select(from_json("json_data", schema).alias("data")).select("data.*")

    # Interface used to write a streaming :class:`DataFrame <pyspark.sql.DataFrame>` to external
    # storage systems (e.g. file systems, key-value stores, etc).
    # Write the above data into memory. Consider the entire analysis in all iteration (output mode = complete). and let the trigger runs in every 10 secs.

    writeDF = parsed_df.writeStream. \
            format("memory"). \
            queryName("floodQuery2"). \
            trigger(processingTime='10 seconds'). \
            start()

    # Print banner text
    print("----- streaming is running -------")

    #writeDF.awaitTermination()

except:
    # Print the error
    print("Unexpected error:", sys.exc_info())
```

Start the Stream

Apache Kafka hands on exercise

Streaming flood-monitoring data: Consumer

```
#clear cached results for a specific table

spark.catalog.uncacheTable("floodQuery2")

# Encapsulate the code in try except blocks
try:
    for x in range(20):
        spark.sql("SELECT * from floodQuery2").show()
        time.sleep(10)
    spark.stop()
except:
    # Print the error
    print("Unexpected error:", sys.exc_info())
```

Select the data from your in memory view

```
+--------+-------+-----+
|datetime|measure|value|
+--------+-------+-----+
+--------+-------+-----+
```

Data APIs

Other data APIs
Reddit

- **Policy Update:** Approval is now required under the new *Responsible Builder Policy* [1].
- This policy applies to **developers (commercial use)**, **researchers (non-commercial use)**, and **bots**.
- The approval process typically takes **1 to 4 weeks**.
- Researchers can submit their approval request using the following form: request-type-researcher

[1] https://support.reddithelp.com/hc/en-us/articles/42728983564564-Responsible-Builder-Policy

Other data APIs

Twitter: Access Levels and Versions

Twitter API access levels and versions

While the Twitter API v2 is the primary Twitter API, the platform currently supports previous versions (v1.1, Gnip 2.0) as well. We recommend that all users start with v2 as this is where all future innovation will happen.

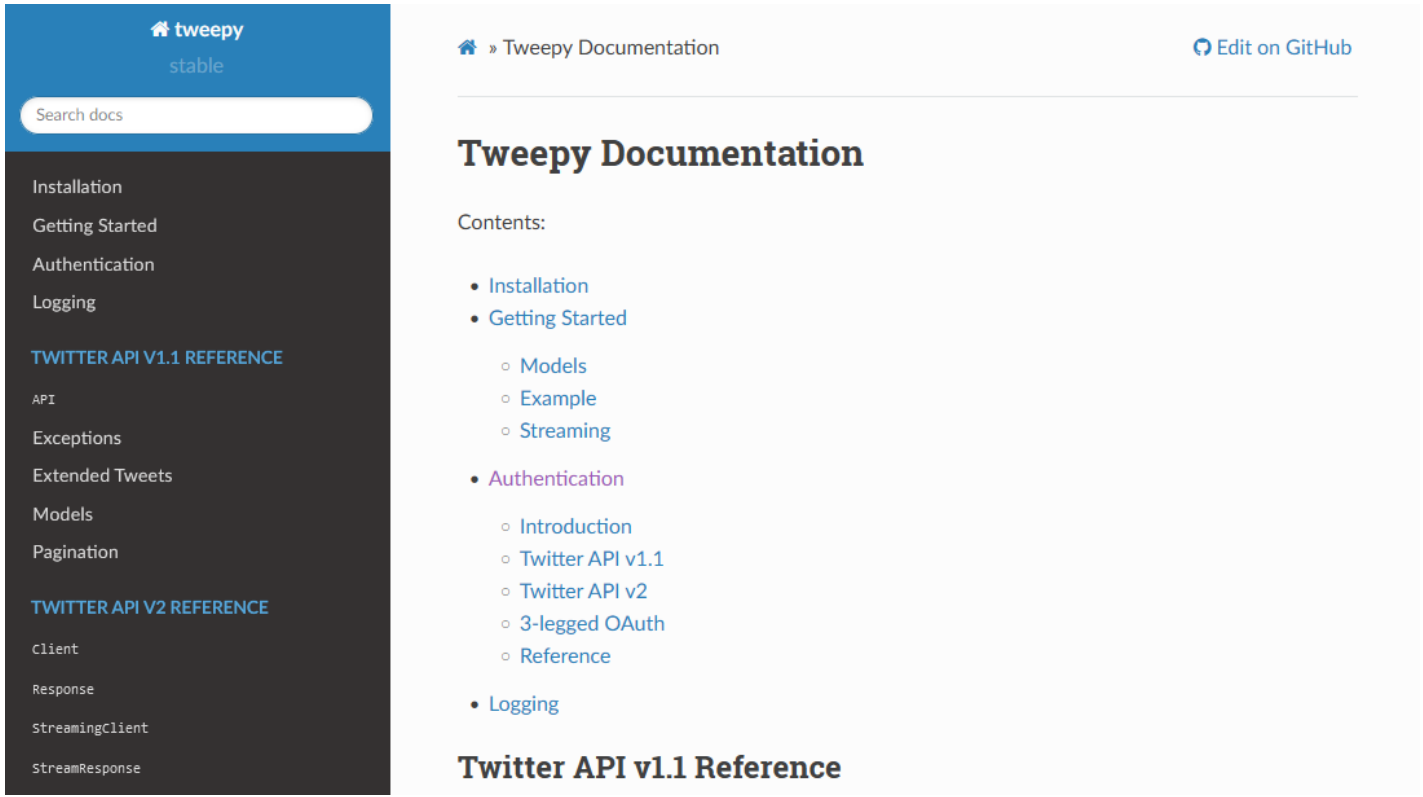
The Twitter API v2 includes a few access levels to help you scale your usage on the platform. In general, new accounts can quickly sign up for Basic access. Should you want additional access, you may choose to apply for Enterprise access.

	Free	Basic	Pro	Enterprise
Getting access	Get Started	Get Started	Get Started	Get Started
Price	Free	$100/month	$5000/month	
Access to Twitter API v2	✔ (Only Tweet creation)	✔	✔	
Access to standard v1.1	✔ (Only Media Upload, Help, Rate Limit, and Login with Twitter)	✔ (Only Media Upload, Help, Rate Limit, and Login with Twitter)	✔ (Only Media Upload, Help, Rate Limit, and Login with Twitter)	
Project limits	1 Project	1 Project	1 Project	
App limits	1 App per Project	2 Apps per Project	3 Apps per Project	
Tweet caps - Post	1,500	3,000	300,000	
Tweet caps - Pull	✘	10,000	1,000,000	
Filteres stream API	✘	✘	✔	
Access to full-archive search	✘	✘	✔	
Access to Ads API	✔	✔	✔	

https://developer.twitter.com/en/docs/twitter-api/getting-started/about-twitter-api

Other data APIs

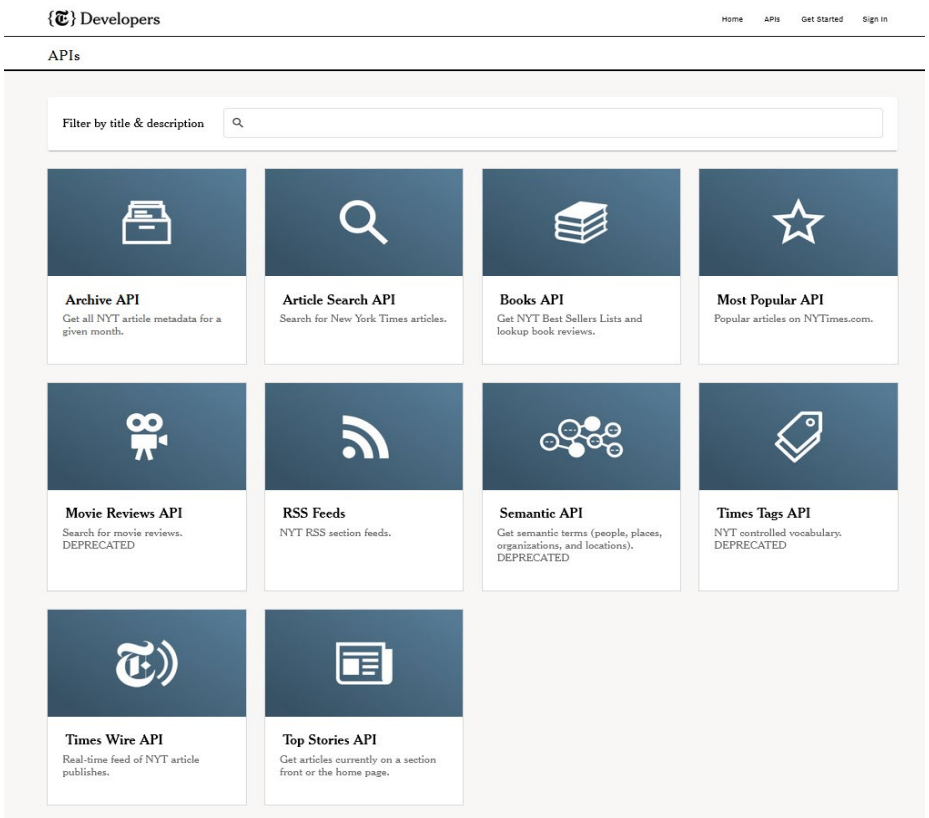
Tweepy: the Python Tweeter API



https://docs.tweepy.org/en/stable/

Other data APIs

NyTimes: a Variety of APIs



https://developer.nytimes.com/apis

Other data APIs

Pynytimes: the Python Nytimes API

README | Code of conduct | MIT license | Security

pynytimes

Run full tests failing | python 3.9 | 3.10 | 3.11 | 3.12 | pypi v0.10.0 | downloads 47k | DOI 10.5281/zenodo.7852730

Use all (actually most) New York Times APIs, get all the data you need from the Times!

Documentation

Extensive documentation is available at: https://pynytimes.michadenheijer.com/.

Installation

There are multiple options to install and ugprade pynytimes, but the easiest is by just installing it using `pip` (or `pip3`).

Linux and Mac

```
pip install --upgrade pynytimes
```

Windows

```
python -m pip install --upgrade pynytimes
```

Usage

You can easily import this library using:

```
from pynytimes import NYTAPI
```

Then you can simply add your API key (get your API key from The New York Times Dev Portal):

```
nyt = NYTAPI("Your API key", parse_dates=True)
```

Make sure that if you commit your code to GitHub you don't accidentially commit your API key.

https://github.com/michadenheijer/pynytimes

DataCamp Reinforcement Learning

DataCamp

You need to copy and paste the link in your browser for it to work!

- Assign yourself for the course via these links

Group 1236:

https://www.datacamp.com/groups/shared_links/eef5b498ab5d09f6e05573715548063b0dbdede3bcd92465030caa11e31a8ec2

Group 0842:

https://www.datacamp.com/groups/shared_links/1820dcd7cdf8e410d8b6e584ddede069d45a9412189790156caaf34c206be3a5

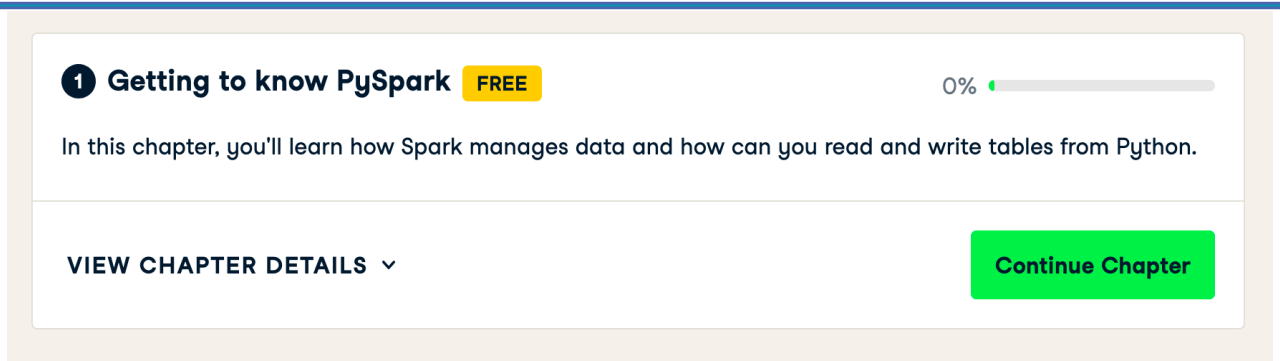
Please be sure to use one of your student email addresses

- username@s.wu.ac.at
- h12345@wu.ac.at

- Please make sure to add your **first name** and **last name** when signing up for the first time

https://campus.datacamp.com/courses/introduction-to-pyspark/

DataCamp Homework

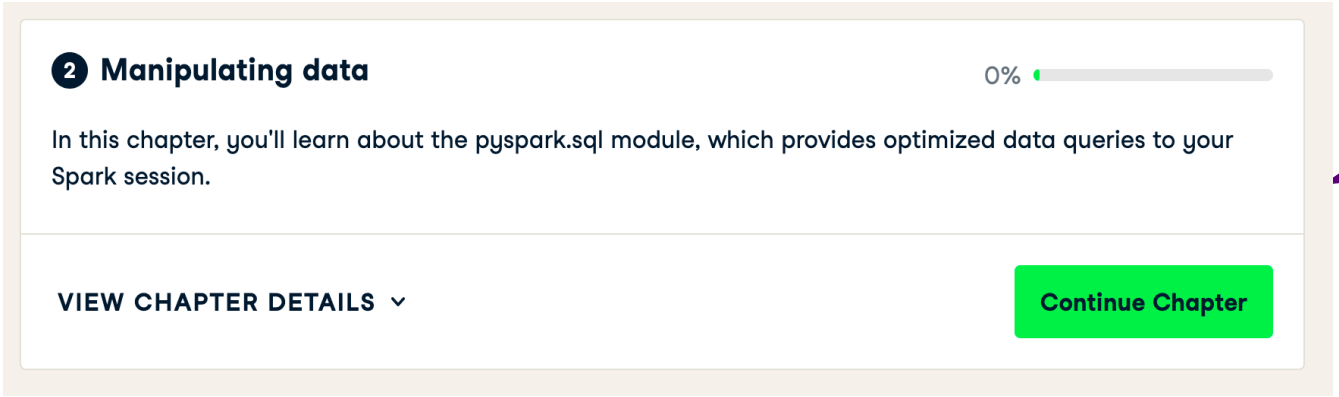


In this chapter you will be introduced to the spark dataframe

Topics include:
What is Spark?
Using Spark in Python
Examining The SparkContext
Using DataFrames
Creating a SparkSession
Viewing tables
Running queries

https://campus.datacamp.com/courses/introduction-to-pyspark/

DataCamp Homework

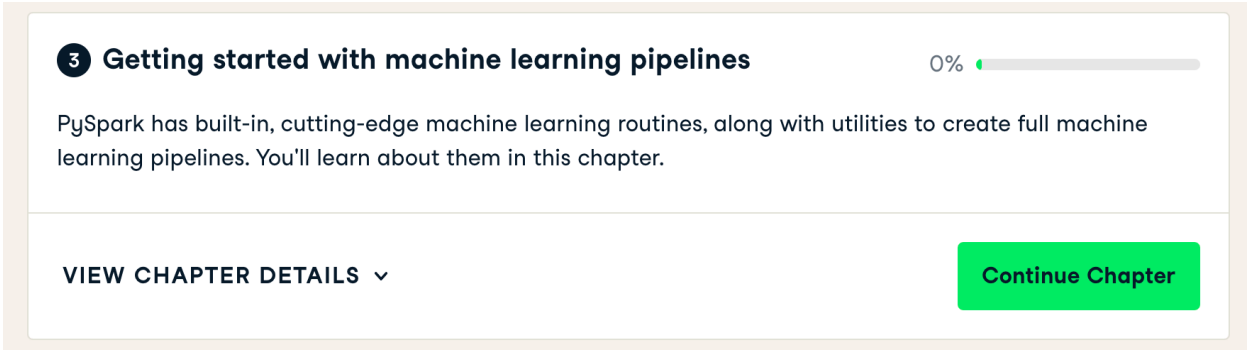


In this chapter you will be introduced to the spark SQL

Topics include:
Creating columns
SQL in a nutshell
Filtering Data
Selecting
Grouping and Aggregating
Joining

https://campus.datacamp.com/courses/introduction-to-pyspark/

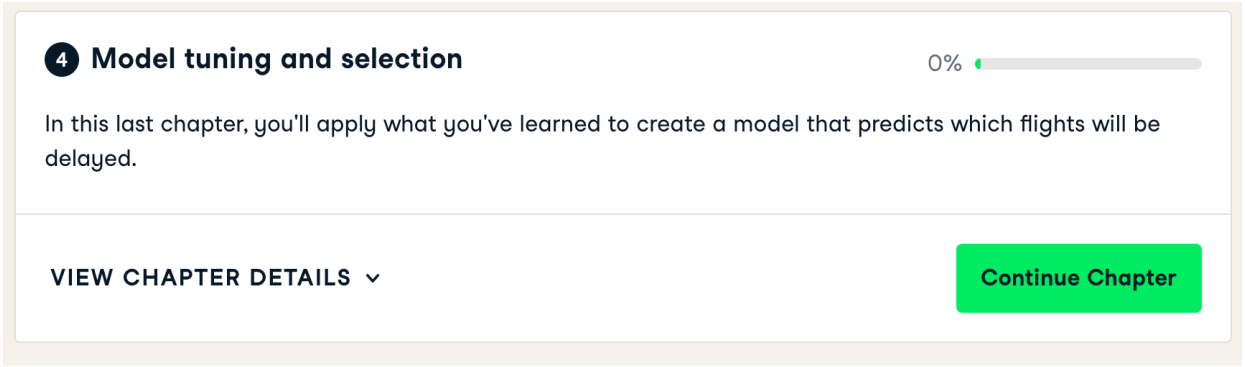
DataCamp Homework



In this chapter you will be introduced to machine learning pipelines

Topics include:
Machine learning piplines
Data types
Strings and factors
Test vs Train

DataCamp Homework



In this chapter you will be introduced to model tuning and selection

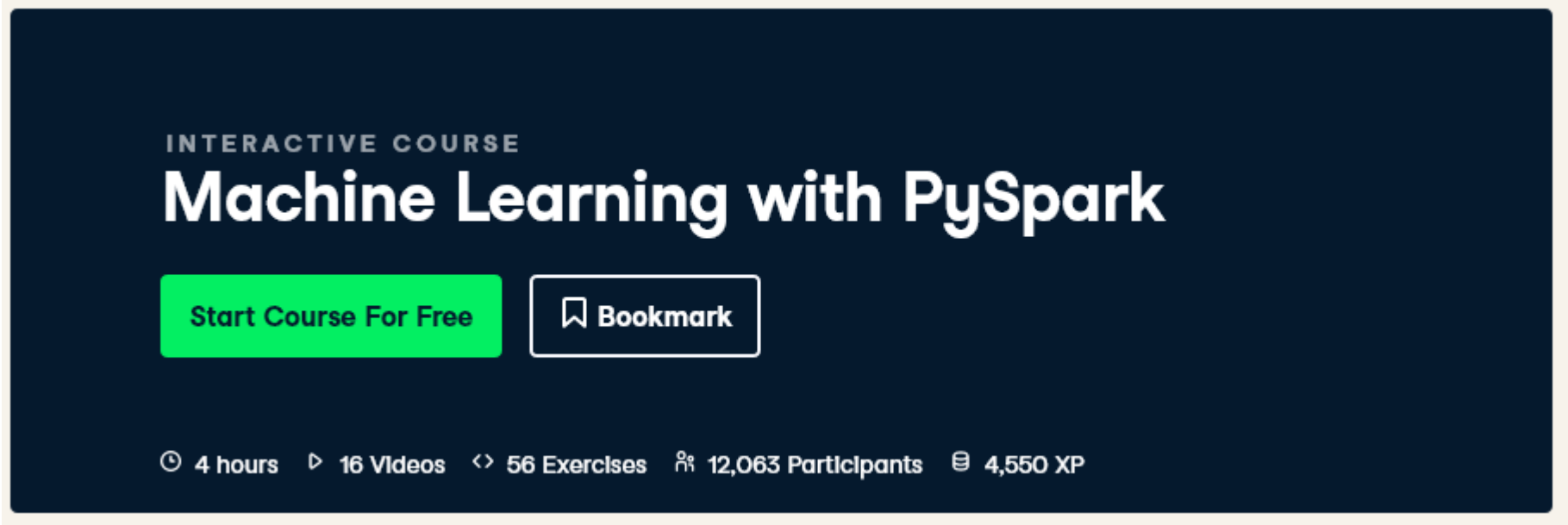
Topics include:
Logistic regression
Cross validation
Evaluating binary classifiers

DataCamp Homework

Earn 4 Bonus points in case you complete the 4 Datacamp chapters

Please make sure to add your **first name** and **last name to** your datacamp profile!

Highly Recommended



https://learn.datacamp.com/courses/machine-learning-with-pyspark

Final Project
Grading Schema

	NOT MEETING BASIC EXPECTATION IN THIS CATEGORY.	MEETING BASIC EXPECTATION.	TENDING TO EXCEED EXPECTATION; SOME MINOR ERRORS.	SIGNIFICANTLY EXCEEDING EXPECTATION.
ENGAGEMENT	THE TEAM DID NOT ENGAGE WITH THE PROBLEM / TASK	THE TEAM ENGAGED WITH THE PROBLEM / TASK TO A LIMITED EXTENT	THE TEAMS ENGAGEMENT IS EVIDENCED BY SOME OF THE SUBMITTED ARTEFACTS	THE SUBMITTED ARTEFACTS CLEARLY DEMONSTRATE THAT THE TEAM HAS ENGAGED FULLY WITH THE PROBLEM / TASK
ELEGANT SOLUTION	THE SOLUTION / ALGORITHM DOES NOT USE SCALABLE TECHNOLOGIES	THE SOLUTION / ALGORITHM SOMETIMES USES SCALABLE TECHNOLOGIES	THE SOLUTION / ALGORITHM USES SCALABLE TECHNOLOGIES ALTHOUGH THE SOLUTION IS NOT ELEGANT	THE SOLUTION / ALGORITHM IMPLEMENTED IS ELEGANT (I.E. USE OF APPROPRIATE BIG DATA FRAMEWORKS AND LIBRARIES)
WORKS AS EXPECTED	THE PROGRAM DOES NOT WORK	THE PROGRAM SOMETIMES DOES NOT WORK AS EXPECTED	THE PROGRAM SOMETIMES DOES NOT WORK AS EXPECTED OR DOES NOT INCLUDE APPROPRIATE EXCEPTION/ERROR HANDLING	THE PROGRAM WORKS AS EXPECTED AND INCLUDES APPROPRIATE EXCEPTION/ERROR HANDLING
BEST PRACTICES	NO EVIDENCE OF BEST PRACTICES	SOME EVIDENCE OF BEST PRACTICES	BEST PRACTICES ARE ALMOST ALWAYS OBSERVED	BEST PRACTICES ARE OBSERVED IN TERMS OF DESIGN, STRUCTURE, READABILITY AND MAINTAINABILITY OF CODE
MARKDOWN	NO MARKDOWN	MARKDOWN USED IN SOME INSTANCES	MARKDOWN IS USED TO DESCRIBE THE PROPOSED SOLUTION BUT SOME INFO IS MISSING	MARKDOWN IS USED TO DESCRIBE THE PROPOSED SOLUTION: INCLUDING THE ARCHITECTURE, LIBRARIES, AND ALGORITHM(S) USED
PROJECT OVERVIEW	PROJECT OVERVIEW IS MISSING	PROJECT OVERVIEW PROVIDED BUT LACKS EXPECTED DETAIL	PROJECT OVERVIEW PROVIDED BUT SOME INFO IS MISSING	DETAILED PROJECT OVERVIEW PROVIDED; BUSINESS / SOCIETAL CONTEXT PROVIDED; JUSTIFICATION PROVIDED
LEGAL AND ETHICAL GUIDELINES & CHALLENGES	LEGAL AND ETHICAL GUIDELINES AND CHALLENGES ARE MISSING	SOME LEGAL AND ETHICAL GUIDELINES AND CHALLENGES MENTIONED BUT LACKS EXPECTED DETAIL	LEGAL AND ETHICAL GUIDELINES AND CHALLENGES DISCUSSED BUT SOME INFO IS MISSING	LEGAL AND ETHICAL GUIDELINES AND CHALLENGES EXPLAINED IN DETAIL AND SUPPORTED VIA EVIDENCE
CHALLENGES ENCOUNTERED	TECHNICAL CHALLENGES ARE MISSING	SOME TECHNICAL CHALLENGES MENTIONED BUT LACKS EXPECTED DETAIL	SOME TECHNICAL CHALLENGES DISCUSSED	TECHNICAL CHALLENGES DISCUSSED IN DETAIL AND SUPPORTED VIA EVIDENCE
EXPERIENCE GAINED	EXPERIENCE GAINED IS MISSING	SOME EXPERIENCES GAINED MENTIONED BUT LACKS EXPECTED DETAIL	SOME EXPERIENCES GAINED DISCUSSED	EXPERIENCE GAINED DISCUSSED IN DETAILED AND SUPPORTED VIA EVIDENCE
FUTURE WORK	FUTURE WORK IS MISSING	SOME FUTURE WORK MENTIONED BUT LACKS EXPECTED DETAIL	SOME FUTURE WORK DISCUSSED	CONCRETE RECOMMENDATIONS FOR FUTURE WORK BASED ON THE RESULTS OF THE PROJECT AND THE EXPERIENCE GAINED
TOTAL POINTS POSSIBLE	POINTS 0-4	POINTS 5-6	POINTS 7-8	POINTS 9-10

70% of your final grade

Final Project
Deliverables

- **Deliverables (upload as 1 ZIP):**
  - Jupyter notebook(s) + data + Readme File (check that it runs, leave cell outputs inside notebook)
  - Jupyter notebook export(s) as HTML
  - Project report in a PDF format.
    Maximum 8 pages excluding references. Font 12pt; Times New Roman; Single Column; Single Space.

Deadline
13.02.2026

Final Project
Declaration of Generative AI in Scientific Writing

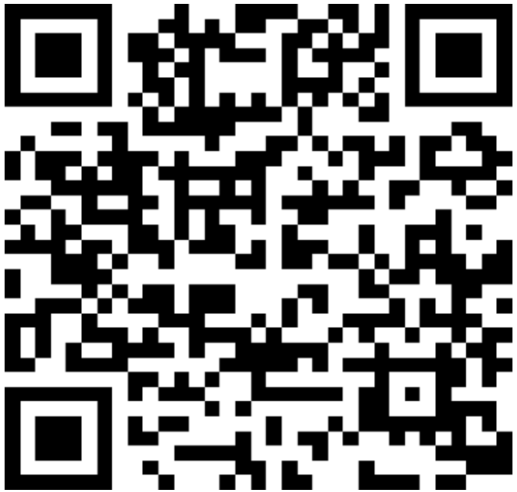
- **Disclosure instructions**
  - Students must disclose the use of generative AI and AI-assisted technologies in the writing process by adding a statement at the end of their report, before the References list. The statement should be placed in a new section entitled 'Declaration of Generative AI and AI-assisted technologies in the writing process'.
  - *Statement: During the preparation of this work the author(s) used [NAME TOOL / SERVICE] in order to [REASON]. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.*
  - This declaration does not apply to the use of basic tools for checking grammar, spelling, references, etc. If there is nothing to disclose, there is no need to add a statement.

Evaluation

- **Course: 1236 Data Processing 2: Scalable Data Processing, Legal & Ethical Foundations of Data Science**

Link to evaluation for students

https://eval.wu.ac.at/lva/28533315



Evaluation

- **Course: 0842 Data Processing 2: Scalable Data Processing, Legal & Ethical Foundations of Data Science**

Link to evaluation for students

https://eval.wu.ac.at/lva/28517501

Thank you / contact details

Department of Information Systems & Operations
Institute for Complex Networks
Welthandelsplatz 1, 1020 Vienna, Austria

Ines Akaichi

Telephone:+43-1-31336-6655
Email: ines.akaichi [at] wu.ac.at