Python libraries for rich data collection

Welcome to today's webinar.

Real-world case study

Real-time user activity tracking at 'BuyOnline'

- **Goal:** Enhance user experience by tracking customer activities in real-time

Key steps:

1. **Define Schemas**: Create Avro schemas for events
2. **Serialize Events**: Use Avro for data serialization
3. **Produce to Kafka**: Send serialized events with kafka-python

Benefits to business:

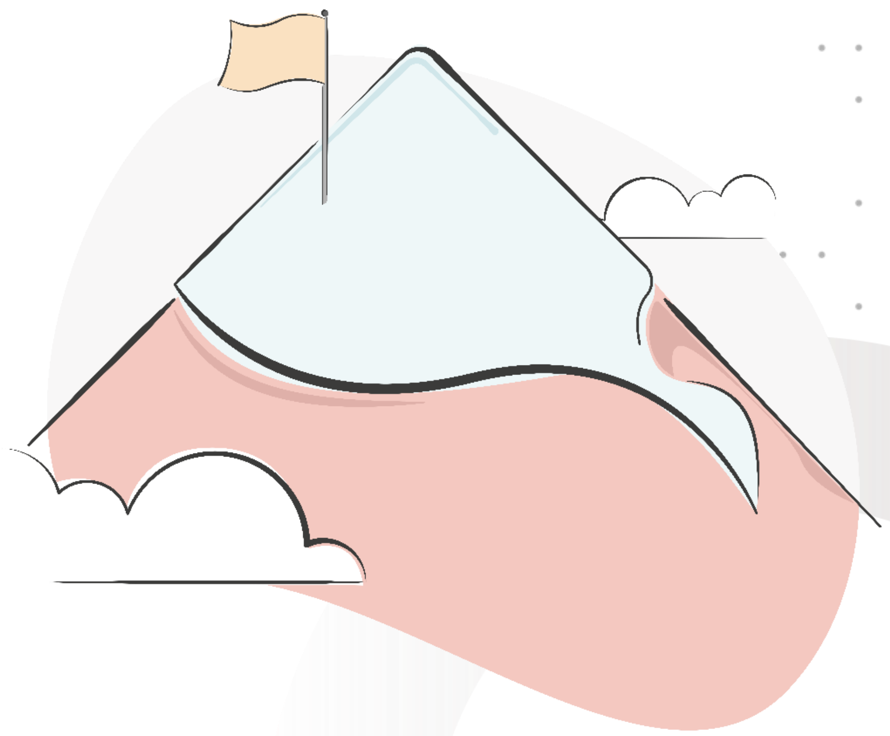
- **Consistency**: Standardised event format
- **Efficiency**: Reduced network overhead
- **Flexibility**: Add new fields without breaking consumers

With the use of Python, 'BuyOnline' were able to improve customer satisfaction with personalised recommendations

Session aim and objectives

By the end of this session, you should be able to:

- Utilise Python libraries such as kafka-python, Avro, and interact with schema registries.
- Integrate Scrapy for data collection from web sources.
- Implement data validation and processing using schema registries.
- Collaborate effectively on data collection projects using best practices and version control systems.



Knowledge check poll

Building Careers Through Education

Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

You are working on an e-commerce project where you need to track user activities in real-time using kafka-python and Avro.

Which of the following steps is NOT part of the process for tracking user activities?

A. Define Avro schemas for different event types.

B. Use Avro to serialize event data according to the defined schemas.

C. Use kafka-Python to send serialized events to specific topics.

D. Use Scrapy to extract user activity data from the website.

Correct answer: D - Scrapy is used for web scraping, not for tracking user activities in real-time using kafka-python and Avro.

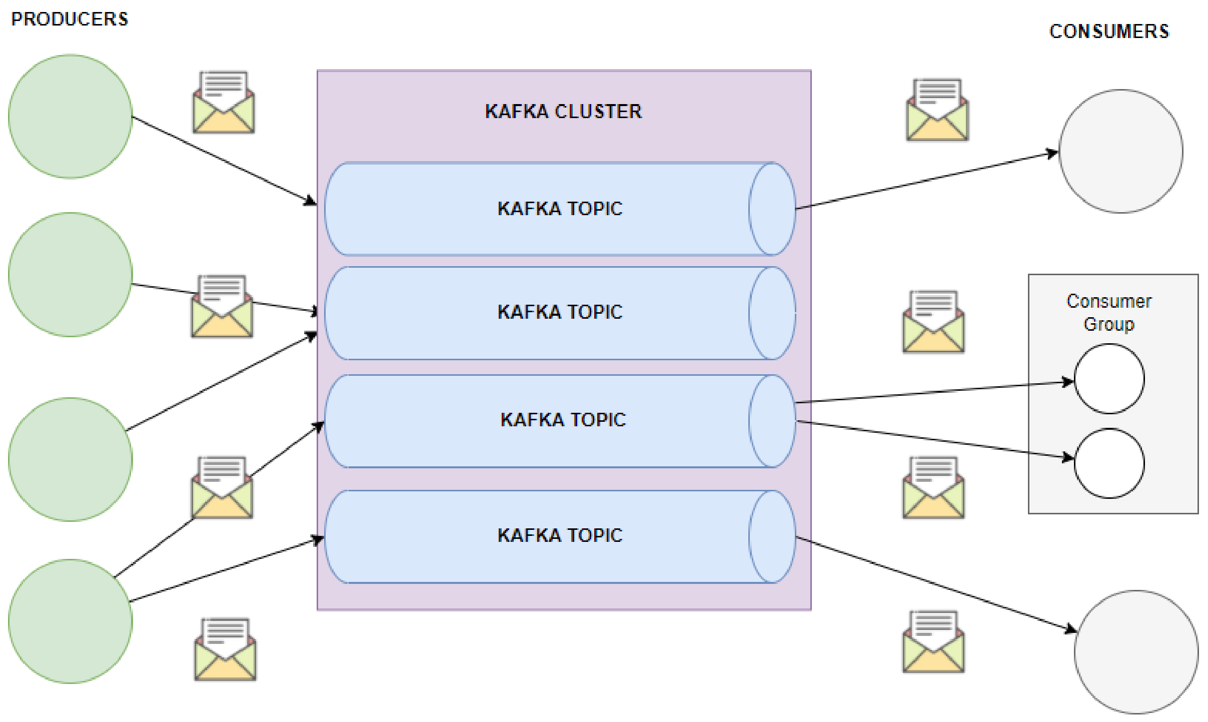
Submit your responses to the chat or turn on your microphone!

BPP

Introduction to kafka-python

Pure Python client for Apache Kafka

- **Pure Python Client**: Easy-to-use interface for Apache Kafka
- **Producer and Consumer APIs**: Simplifies interaction with Kafka topics
- **Support for Kafka Protocols**: Ensures compatibility with Kafka clusters
- **Thread Safety**: Allows safe multi-threaded access for high-performance applications



Kafka and Python integration, image source: [Medium]

Using kafka-python

Producing and consuming messages

```
from kafka import KafkaProducer

producer =
KafkaProducer(bootstrap_servers='localhost:9092')
producer.send('my_topic', b'Hello, Kafka!')
producer.flush()
```

The code for sending messages to a Kafka topic

```
from kafka import KafkaProducer

producer =
KafkaProducer(bootstrap_servers='localhost:9092')
producer.send('my_topic', b'Hello, Kafka!')
producer.flush()
```

The code for consuming messages with Kafka-python

Introduction to Apache Avro

A data serialisation system

What is Apache Avro?

- **Data Serialization**: Compact, fast, binary format
- **Uses JSON**: Defines data types and protocols
- **Efficient**: Compact binary serialization

The benefits of Using Apache Avro

- **Compact**: Reduces data size, improves speed
- **Evolving Schemas**: Allows changes without breaking compatibility
- **Language Agnostic**: Supports interchange between different languages

Defining Avro Schemas

A user record schema

- **Example Schema**: User record schema

- **Key Elements**:
    - Fields
    - Types
    - Default values

```
{
 "namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
 {"name": "username", "type": "string"},
 {"name": "email", "type": ["null",
"string"], "default": null},
 {"name": "age", "type": "int"}
 ]
}
```

An example schema for a user record

Serializing data with Avro

Using avro-python3 library

- **Integration**: Sending serialized messages to Kafka

Code Snippet:

- **Import Modules**: Avro and I/O library
- **Parse Schema**: Load Avro schema from file
- **Setup Writer/Encoder**: Initialize DatumWriter and BinaryEncoder
- **Define Data**: User data to serialize
- **Serialize Data**: Write data to encoder, get serialized bytes

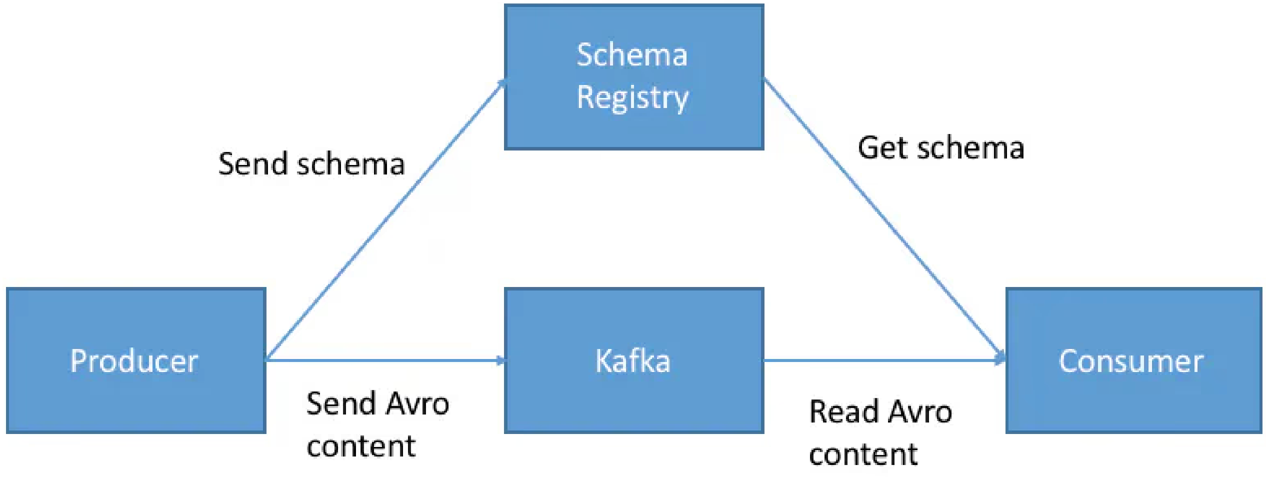
```
from avro.io import DatumWriter, BinaryEncoder
from avro.schema import Parse
import io

schema = Parse(open("user.avsc", "r").read())
writer = DatumWriter(schema)
bytes_writer = io.BytesIO()
encoder = BinaryEncoder(bytes_writer)

user_data = {"username": "john_doe", "email":
"john@example.com", "age": 30}
writer.write(user_data, encoder)
raw_bytes = bytes_writer.getvalue()
```

An example using the avro-python3 library to serialize data

Introduction to Schema Registries

A central repository for schemas

- Stores and retrieves schemas for data serialization.
- Central repository for schemas.
- Ensures agreement on data structure between producers and consumers.



An illustration of the function of Schema registries, condukter.io/blog: Link

Benefits of schema registries:

- Schema management
- Compatibility checks
- Reducing overheads

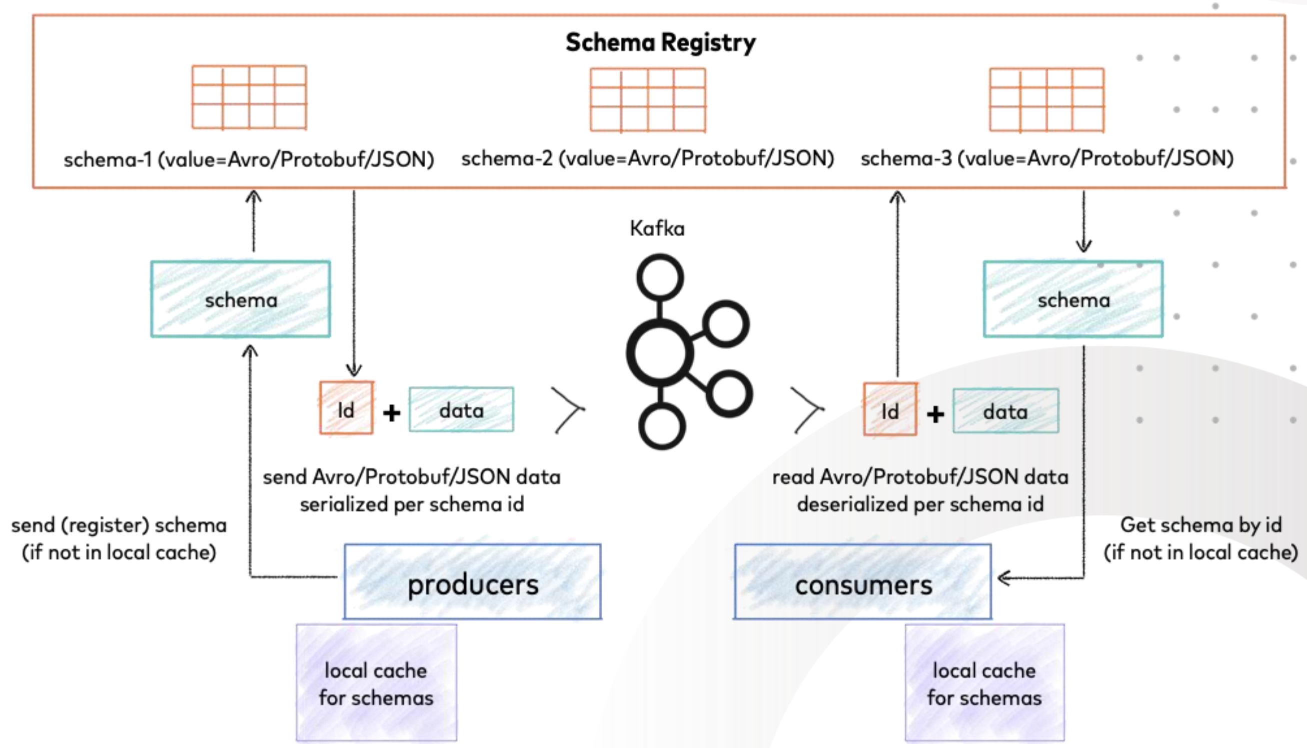
Introducing Confluent Schema Registry

A popular open-source schema registry

Confluent Schema Registry is a popular open-source schema registry that supports Avro, JSON Schema, and Protobuf.

Key features include:

- **RESTful interface:** Interact with the registry using HTTP calls
- **Multi-format support:** Handles different serialisation formats.
- **Compatibility modes:** Enforce schema evolution rules (e.g., backward, forward, full).



Schema Registry
schema-1 (value=Avro/Protobuf/JSON)
schema-2 (value=Avro/Protobuf/JSON)
schema-3 (value=Avro/Protobuf/JSON)
Kafka
schema
schema
Id
data
Id
data
send Avro/Protobuf/JSON data serialized per schema id
read Avro/Protobuf/JSON data deserialized per schema id
send (register) schema (if not in local cache)
Get schema by id (if not in local cache)
producers
consumers
local cache for schemas
local cache for schemas

Confluent registry overview, image source, Confluent documentation: Link

Integrating Scrapy for web data collection

Scrapy: A web scraping framework

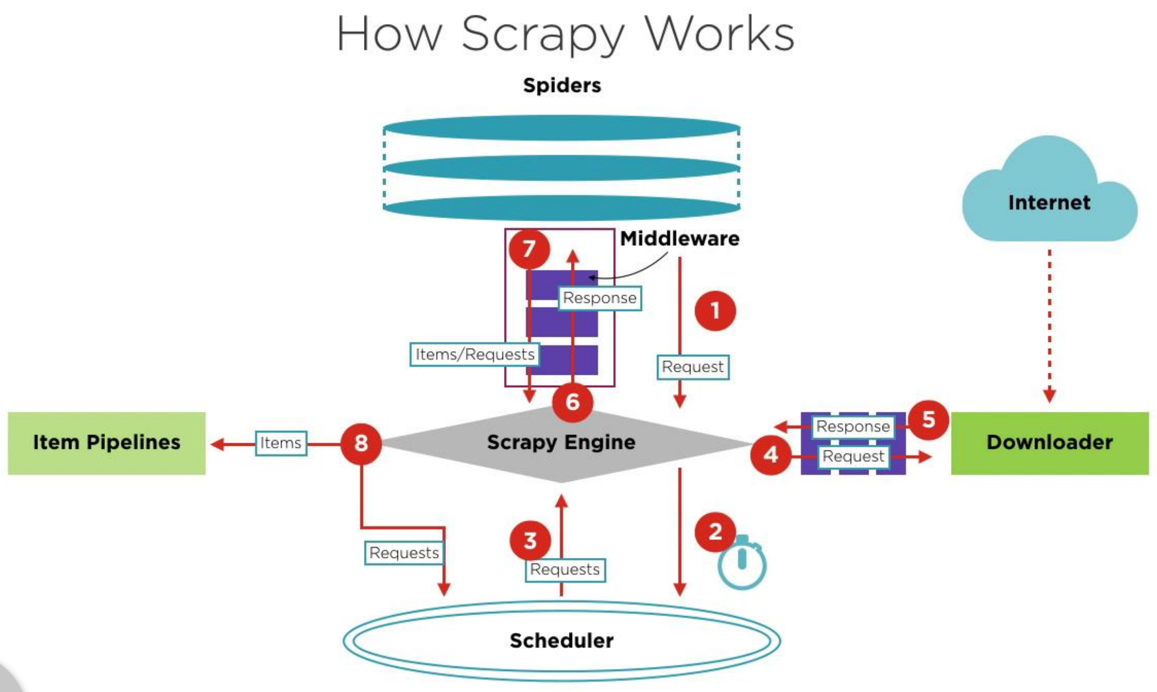
Scrapy is a powerful, open-source Python framework for extracting data from websites, designed for speed, efficiency, and extensibility.

Key features include:

- Asynchronous processing
- Selective data extraction
- Extensibility

Importance functions in Scrapy:

Installation, creating a new project, defining a Spider, running the spider – **these will be practiced during the practical lab.**



How a Scrapy works image source, Pluralsight: Link

Data validation and processing

Ensuring data consistency

Data Ingestion Challenges

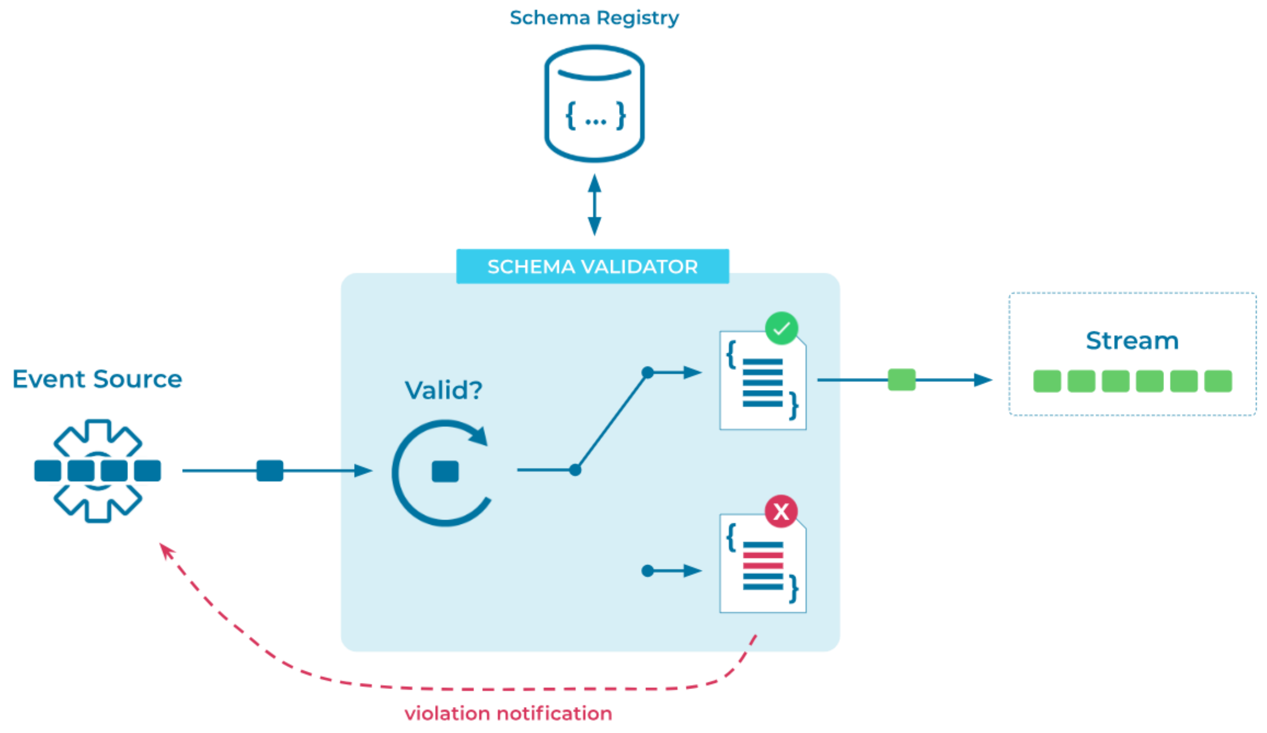
- Inconsistencies and errors.
- Need for validation.

Role of Schema Registries

- Store schemas, enforce rules.
- Reject invalid data.

Validation Process

- **Define Schemas**: Data types, required fields, defaults, ranges/patterns.
- **Set Compatibility Modes**: Enforce compatibility, prevent changes.
- **Validate in Producers**: Check data before sending, catch errors early.

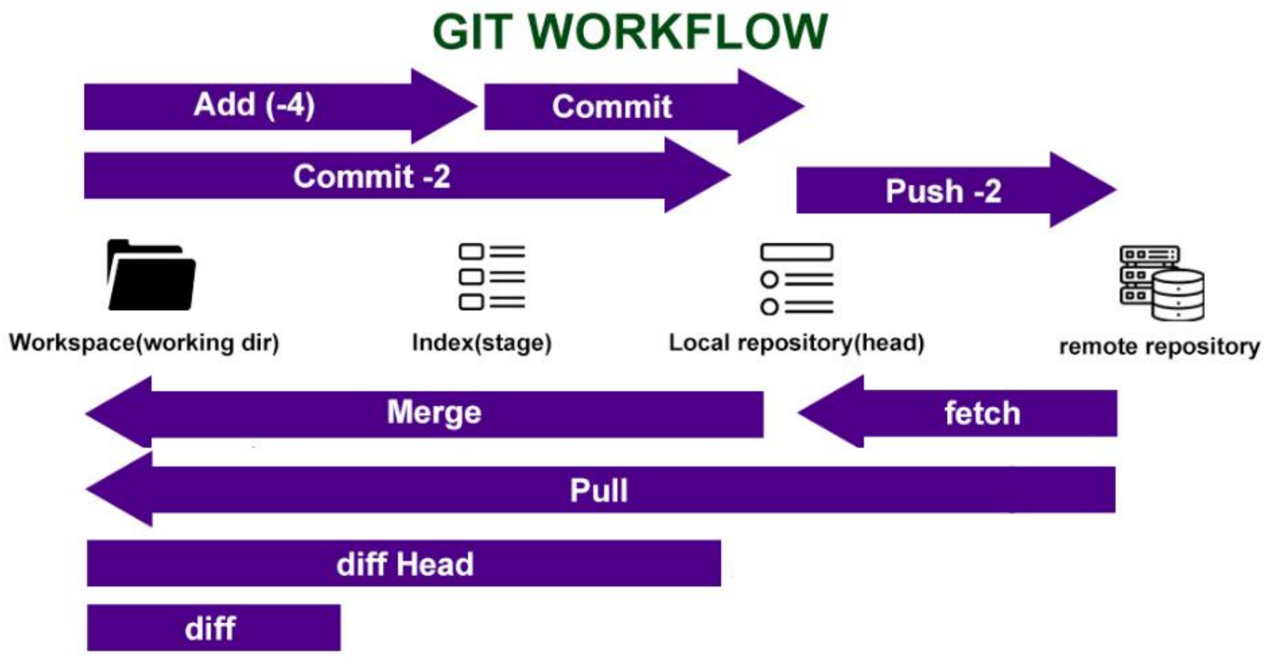


An illustration of a schema validator, image source, Confluent Developer: [Link]

Collaboration and best practices

Version control using Git

- Repository structure.
- Branching strategies.
- Code reviews.
- CI/CD.



Best Practices

- Define roles and responsibilities.
- Establish coding standards.
- Use collaborative tools (e.g., Jira, Slack).
- Implement CI/CD pipelines.

How Git version control works, image source: GeeksforGeeks, link

Do you already follow some of these best practices when completing your own data projects?

Time for the practical lab!

Your tutor will provide guidance as required…

Practical lab activities detailed in this document: Lab activities

Building Careers Through Education

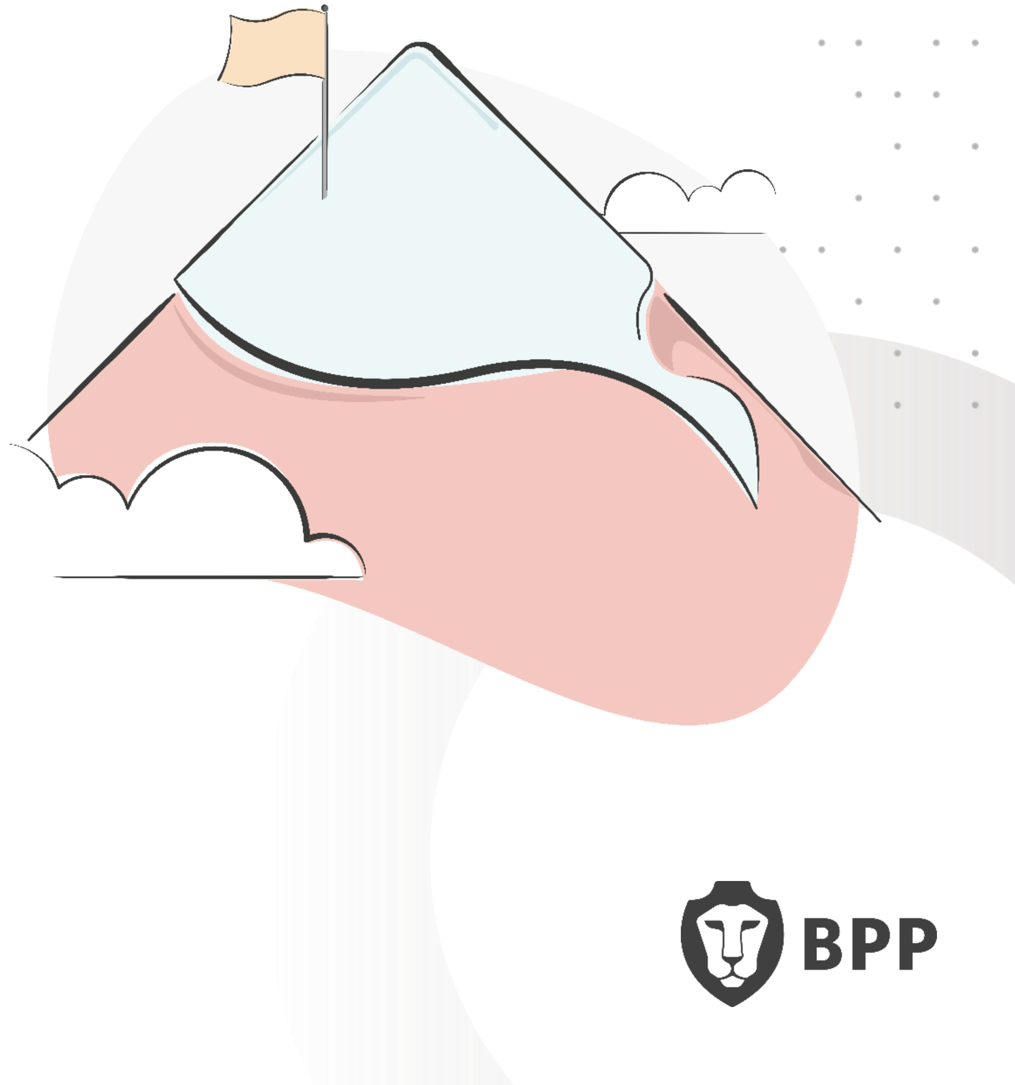
Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

Session aim and objectives

Building Careers Through Education
Everybody Matters.
Stronger Together.
Student, Learner and Client Centric.
Trust and Respect.
Embrace Change.

You should now be able to:

- Utilise Python libraries such as kafka-python, Avro, and interact with schema registries.
- Integrate Scrapy for data collection from web sources.
- Implement data validation and processing using schema registries.
- Collaborate effectively on data collection projects using best practices and version control systems.



BPP

Key Learning Summary

Here is a summary of the key learning points for this topic:

- **kafka-python:** A pure Python client for Apache Kafka, providing easy-to-use producer and consumer APIs.
- **Apache Avro:** A data serialization system that uses JSON for defining data types and protocols and serializes data into a compact binary format.
- **Schema Registries:** Central repositories for storing and retrieving schemas, ensuring data consistency and compatibility.
- **Scrapy:** An open-source Python framework for web scraping, designed for speed and efficiency.
- **Data Validation:** Ensures data conforms to expected formats and values before processing, reducing errors and inconsistencies.
- **Version Control:** Using systems like Git to manage code changes, collaborate effectively, and maintain code quality.
- **Best Practices:** Implementing branching strategies, code reviews, and CI/CD pipelines to ensure high-quality outcomes in data collection projects.

BPP

Thank you

Do you have any questions, comments, or feedback?