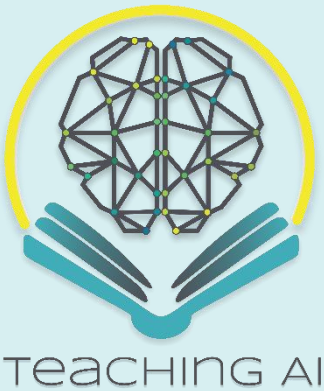

Report WP4-A2:

Overview of Different Python Libraries

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency. (EACEA). Neither the European Union nor EACEA can be held responsible for them.



Result

A descriptive overview of Python libraries and their possible uses in AI education across numerical, image, audio and video data.

Related to

WP4-A2: Overview of different Python libraries

Statement of originality

This report contains original unpublished work, except where clearly indicated otherwise.

Disclaimer

This report contains material, which is the copyright of TAI Consortium Parties. All TAI Consortium Parties have agreed that the content of the report is licensed under a Creative Commons Attribution Non-Commercial Share Alike 4.0 International License. TAI Consortium Parties does not warrant that the information contained in the Report is capable of use, or that use of the information is free from risk and accept no liability for loss or damage suffered by any person or any entity using the information.

Copyright notice

© 2024-2027 TAI Consortium Parties

Note

For anyone interested in having more information about the project, please see the website at: https://tai-erasmus.eu/

This publication is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License (CC BY-NC-SA 4.0).

Document version history

Version	Date	Author	Comment
1.0	09.11.2026.	Mirza Pašić, UNSA	Initial version
1.1	17.11.2026.	Mirza Pašić, UNSA	Comparative library evaluation, links with WP4-A1 and indicative recommendations
1.2	17.11.2026.	Mirza Pašić, UNSA	Descriptive presentation; citations and reference list removed

Table of contents

1. Introduction 4

2. Scope and organisation of the overview 4

3. Common Python libraries and deep learning frameworks 6

4. Numerical and tabular data 7

5. Image data 9

6. Audio and sound data 11

7. Video data 13

8. Evaluation, interpretation and interactive learning 14

9. Comparative assessment and educational applicability 15

10. Technical requirements and reproducibility 22

11. Conclusion 23

Annex 1. Library options for selected datasets 24

1. Introduction

The Teaching Artificial Intelligence (TAI) project is implemented within the Erasmus+ Programme, Key Action 2 – Cooperation Partnerships in Higher Education. Its educational activities cover numerical and tabular data, images, audio and sound, and video. These modalities require different forms of data preparation, modelling and evaluation, but share a substantial part of the Python scientific-computing ecosystem.

This report presents an overview of Python libraries that can be used to prepare data, implement AI models, evaluate results and develop educational examples. It describes their main functions, strengths, limitations and possible combinations across numerical, image, audio and video workflows.

The scope covers the full range of project data modalities. Numerical regression, classification, time series prediction and clustering are considered alongside image classification, object detection, segmentation, sound classification, speech-related tasks, video classification and temporal analysis. Libraries supporting visualisation, interpretation and interactive learning are also included because these functions influence the educational usefulness of the modelling tools.

The libraries are presented as possible tools for different tasks and teaching needs. The overview can support the preparation of practical examples, while the final choice of libraries will be made when work begins with the actual data and the requirements of each implementation become clear.

1.1 Data and modelling context

Data formats, annotations, scale and preparation requirements provide the context for considering Python libraries. Standard tables, labelled images, WAV recordings and video-related collections do not require identical data loaders or model interfaces. A library that is useful for one stage may need to be combined with another for the complete workflow.

The model families discussed in WP4-A1 provide useful context for the range of tasks considered here. The present overview describes Python tools that can support these tasks, including preprocessing, modelling, evaluation and notebook-based learning. Section 9 illustrates possible connections between model families and libraries.

1.2 Organisation of the report

The report first outlines the scope of the overview and the common Python ecosystem. Separate sections then describe numerical, image, audio and video libraries. Section 9 compares the main learning libraries, illustrates their use with different model families and presents possible combinations for each modality. The final sections discuss execution conditions and reproducibility. Annex 1 provides examples of library roles for selected datasets.

2. Scope and organisation of the overview

The overview is descriptive. It explains what different Python libraries can be used for and how they may be combined in practical AI workflows. The discussion considers data preparation, model

implementation, evaluation and educational use. It provides a starting point for choosing tools when the characteristics of the actual data and the intended exercises are known.

2.1 Scope of the overview

The overview covers general scientific libraries, machine-learning packages, deep-learning frameworks and tools specific to image, audio and video data. Supporting packages are included where they address data formats, annotations, model interpretation or interactive learning. The examples represent common task families without attempting to list every Python package or every implementation of each architecture.

The discussion distinguishes between a library, a framework, a model architecture and a pretrained checkpoint. CNN, LSTM, U-Net and YOLO describe model families or architectures. Their implementation may use a general learning framework, a specialised package or an additional model definition.

2.2 Analysis dimensions

The descriptions are organised around the dimensions in Table 2.1. The same considerations apply across modalities, although their practical importance varies. For video, decoding and memory use may dominate; for a small numerical dataset, transparent preprocessing and evaluation may matter more.

Table 2.1. Dimensions used to describe the libraries

Dimension	Questions addressed
Functionality	Which tasks, model types and preparation steps are supported?
Data compatibility	Which formats, array structures and annotation types can be handled?
Ease of use	How much setup and programming knowledge does the interface require?
Educational applicability	Can the main steps be explained and inspected in practical learning?
Documentation	Are official guides, examples and API descriptions available?
Interoperability	How does the package connect to other libraries and data structures?
Resources and maintenance	What dependencies, memory demands and release changes affect use?
Conditions of use	What software, checkpoint and data conditions require checking?

2.3 Interpreting the overview

Educational applicability depends on the interface, prerequisite knowledge and workflow complexity. Terms such as introductory or advanced describe possible learning uses. The same library can be used

at different levels by adjusting the dataset, model and amount of implementation detail shown to learners.

The Python ecosystem changes over time. Model availability, supported formats, platform support and licensing may differ between releases. The functions and dependencies of the chosen release should therefore be checked when an implementation is prepared.

3. Common Python libraries and deep learning frameworks

3.1 Numerical computation and data structures

NumPy provides multidimensional arrays and numerical operations that underlie many Python AI workflows. It can represent numerical features, image pixels, audio waveforms and extracted video features. Its educational value lies in making array shapes, vectorised operations and mathematical transformations visible. It does not, by itself, provide the full set of data-management or deep-learning functions needed for every application.

pandas provides labelled tabular structures and tools for importing, inspecting and transforming data. It can support missing-value analysis, category handling, joins, time-indexed observations and metadata tables. These functions are relevant to numerical datasets and to the label or annotation files accompanying multimedia collections. Converting a table into an array or tensor requires attention to column order and data types.

SciPy adds scientific operations such as optimisation, statistical functions, sparse matrices and signal-processing routines. These can support feature preparation and analytical exercises across modalities. Its functions generally operate below the level of a complete teaching workflow, so their role depends on the transformation or calculation being demonstrated.

3.2 Data access and storage

The format of a dataset may require tools beyond a standard CSV reader. h5py provides an interface to HDF5 storage and can access portions of array-based data without reading the entire file into memory. Xarray adds labelled multidimensional data structures, which can be relevant to environmental observations and data indexed by time or spatial dimensions. Neither replaces the need to understand the stored variables and units.

Hugging Face Datasets provides facilities for loading, processing and streaming datasets and for connecting them to several numerical and learning frameworks. It can be useful where a dataset is distributed through a supported repository or format. Availability through a software interface does not determine the licence of the dataset or remove access conditions imposed by its provider.

3.3 TensorFlow and Keras

TensorFlow provides tensor operations, automatic differentiation and facilities for building and training neural networks. Its data-pipeline tools can support batches of numerical observations, images or sequences. It can therefore be considered across the four modalities, while specialised loaders and task-specific model definitions may still be needed.

Keras offers a higher-level interface for constructing, training and evaluating neural networks. Keras 3 supports multiple backends, including TensorFlow, JAX and PyTorch; it should not be described solely as an independent synonym for TensorFlow. A concise high-level interface may help students follow the model structure, although backend-specific operations and third-party components can reduce portability.

For teaching, this ecosystem can accommodate dense networks, convolutional networks, recurrent architectures and encoder–decoder designs. The existence of suitable layer types does not mean that all architectures or pretrained weights are supplied in the same package. Tensor shapes, output activation, loss function and evaluation procedure remain specific to the problem.

3.4 PyTorch and JAX

PyTorch provides tensors, automatic differentiation, neural-network components and data-loading tools. It can support both standard training workflows and custom model definitions. Explicit training loops can expose optimisation steps and gradient calculations to students, while requiring more attention to device placement, model modes and implementation details. Its use can extend across numerical, image, audio and video tasks.

JAX combines array computation with transformations such as differentiation, vectorisation and compilation. It offers another route for computational and neural-network work, including use as a Keras backend. Its transformation-based programming model introduces additional concepts, so its educational suitability depends on whether those concepts form part of the intended learning activity.

These frameworks overlap in purpose but differ in programming style, ecosystem and execution requirements. The report treats them as alternatives or complementary options for particular tasks. It does not identify a single framework as the required foundation for all partners.

4. Numerical and tabular data

Numerical datasets can contain continuous measurements, categories, timestamps and target variables. The relevant library depends on the analytical task and on whether the learning objective concerns predictive accuracy, statistical interpretation, temporal dependence or data structure. The same dataset can support more than one of these perspectives.

4.1 Regression and classification

scikit-learn provides a common estimator interface for many regression and classification methods, together with preprocessing, pipelines, model selection and evaluation. Its scope includes linear models, support-vector methods, nearest neighbours, trees, ensembles and multilayer perceptrons. The shared interface can make method comparison easier to follow in teaching, while specialised deep architectures and sequence models generally require a different framework.

XGBoost, LightGBM and CatBoost provide additional gradient-boosting implementations. They can support regression and classification on structured data, with different approaches to training, parameterisation and categorical features. They offer alternatives for comparing models on the same dataset.

TensorFlow/Keras, PyTorch and JAX-based implementations provide options for neural regression and classification. These tools offer greater control over architecture and training, while increasing the number of design choices that students must understand. Binary, multiclass and continuous outputs require corresponding target encodings and loss functions; this distinction applies independently of the framework.

4.2 Statistical modelling and time series

statsmodels provides statistical models, estimation and diagnostic tools, including regression and time-series analysis. It can support exercises in which coefficient interpretation, uncertainty, residual analysis or a classical forecasting model is part of the learning objective. Statistical assumptions and diagnostic checks remain necessary when these functions are used.

sktime offers a task-oriented ecosystem for time-series analysis, including forecasting workflows. It can provide a different perspective from manually constructing lagged features or neural-network windows. Its range of estimators and optional dependencies needs to be considered for the particular task rather than treated as one small installation with uniform requirements.

Sequence models can also be constructed in TensorFlow/Keras or PyTorch, while lagged numerical features can be used with scikit-learn or boosting libraries. These routes address temporal information differently. Chronological splitting, forecast horizon and the availability of past target values must be specified before comparing their outputs. A naive or seasonal reference forecast can be implemented with basic array and table operations.

4.3 Clustering, dimensionality reduction and imbalance

For unsupervised learning, scikit-learn includes clustering and decomposition methods that can support exploratory exercises and feature preparation. UMAP provides a further option for nonlinear dimensionality reduction and visualisation. A two-dimensional representation can help inspect structure, but visual separation alone does not establish a reliable clustering result or a predictive relationship.

Autoencoders and other learned representations can be implemented in the general deep-learning frameworks. Specialised methods may require an additional package or a separate implementation. Keeping this distinction explicit helps prevent model names from being presented as if they were interchangeable software libraries.

imbalanced-learn provides resampling methods and related pipeline support for class-imbalanced problems. It can be considered for tasks such as loan classification, where the distribution of classes may affect learning and evaluation. Resampling belongs inside the training procedure; performing it before splitting the data can introduce leakage.

Table 4.1. Library options for numerical and tabular tasks

Task	Possible libraries	Main consideration
Regression and classification	scikit-learn; XGBoost; LightGBM; CatBoost	Estimator interface, encoding and parameterisation
Neural models	TensorFlow/Keras; PyTorch; JAX-based tools	Architecture, training procedure and resources
Statistical analysis	statsmodels; SciPy	Assumptions, interpretation and diagnostics
Time series	statsmodels; sktime; general learning frameworks	Temporal structure and forecast horizon
Clustering and representation	scikit-learn; UMAP; neural frameworks	Distance, scaling and meaning of the representation
Class imbalance	imbalanced-learn	Resampling within the training procedure

5. Image data

Image workflows involve more than choosing a classifier. File decoding, colour representation, resizing, annotation handling and augmentation influence the information presented to a model. Classification labels, bounding boxes and segmentation masks require different preparation and evaluation procedures.

5.1 Image loading and processing

Pillow provides image reading, conversion and manipulation functions that can support standard formats and simple preparation tasks. OpenCV covers a wider range of computer-vision operations, including image processing, geometric transformations, video input and feature-based analysis. Their array conventions need to be understood; for example, colour-channel ordering must be checked when moving between processing tools and a model.

scikit-image provides image-processing algorithms within the scientific Python ecosystem. It can support exercises involving filtering, morphology, segmentation and feature extraction. These methods can be used independently of deep learning or as part of a preparation workflow, making the relationship between classical vision and learned models visible.

5.2 Classification and transfer learning

TensorFlow/Keras and PyTorch can implement convolutional classifiers and transfer-learning workflows. Torchvision adds datasets, transforms and model components within the PyTorch ecosystem. Available pretrained architectures can reduce the amount of task-specific training,

although their input normalisation, image size, class definitions and weight provenance must match the intended use.

For datasets such as CIFAR-10 and logo collections, these tools provide several possible routes from image files and labels to model inputs. Dataset size alone does not determine complexity: variable image dimensions, repeated identities and uneven class distributions can make a small collection methodologically demanding.

Transformers also provides interfaces to supported image and vision–language models, including vision transformers. This offers another route for examining pretrained representations, alongside convolutional classification. Support must be checked at the model level because classification, image–text alignment and caption generation have different inputs and objectives.

5.3 Object detection and segmentation

Ultralytics provides a Python interface to supported YOLO implementations and related computer-vision tasks. Torchvision supplies model components for detection and segmentation, while general deep-learning frameworks allow custom implementations of architectures such as U-Net or fully convolutional networks. Task support depends on the model and release being used.

The Segment Anything repository offers code and checkpoints for promptable segmentation. It represents a further option for examining pretrained visual representations and mask generation. It should be distinguished from a supervised semantic-segmentation model trained to assign the project dataset's own class labels.

Annotation processing may require additional tools. pycocotools supports operations associated with COCO-style annotations and evaluation. XML files can be parsed using Python's standard library, while other schemas require explicit conversion. A supported image format does not imply that its annotations are immediately compatible with every training interface.

5.4 Augmentation and conditions of use

Torchvision transforms, framework preprocessing layers and the Albumentations ecosystem provide different augmentation options. Geometric changes must remain consistent with bounding boxes, masks and keypoints. Augmentation also needs to preserve the meaning of the label: a transformation that is harmless for one image task may change the target in another.

Software identity and licence should be recorded precisely. Current Albumentations documentation distinguishes AlbumentationsX and its licensing options; older package names or releases should not be assumed to have identical terms. Ultralytics also publishes conditions governing its software. This report records these as factors in assessing suitability and does not make a consortium-wide licensing decision.

Table 5.1. Library options for image workflows

Function	Possible libraries	Main consideration
Reading and processing	Pillow; OpenCV; scikit-image	Format, channel order and numerical range
Classification and transfer learning	TensorFlow/Keras; PyTorch; Torchvision	Input conventions and pretrained weights
Detection	Ultralytics; Torchvision	Bounding-box schema and class mapping
Segmentation	General frameworks; Torchvision; Segment Anything code	Semantic classes, masks and prompting
Augmentation	Framework transforms; Albumentations ecosystem	Consistent treatment of labels and geometry
Annotations and evaluation	pycocotools; schema-specific parsing tools	Annotation format and evaluation protocol

6. Audio and sound data

Audio datasets differ in sampling rate, channel count, duration and label structure. A spoken digit, an environmental sound and an instrument recording can all be stored as WAV files while requiring different preparation and modelling decisions. The review therefore distinguishes decoding, feature extraction and learning tools.

6.1 Loading, standardisation and feature extraction

SoundFile provides a Python interface for reading and writing audio through libsndfile. librosa offers functions for audio analysis, including time–frequency representations and features such as mel spectrograms and MFCCs. These tools can support the conversion of recordings into representations used by classical or neural models. Format support and resampling behaviour depend on the selected functions and their underlying dependencies.

SciPy can support filtering and other signal-processing calculations. TorchAudio provides audio transformations and other PyTorch-oriented components, while TorchCodec addresses media decoding and encoding within the PyTorch ecosystem. Their roles should be checked against the relevant release: the official documentation describes changes to the division of responsibilities between these projects.

6.2 Sound classification and anomaly detection

For fixed-length feature vectors, scikit-learn and boosting libraries can support classification or anomaly-detection workflows. Spectrogram-based classifiers and waveform models can be

implemented with TensorFlow/Keras or PyTorch. The choice of representation determines whether the model receives a numerical vector, a two-dimensional time–frequency array or a sequence.

FSDD, Sopele Sounds and ESC-50 provide different contexts for these approaches. Spoken-digit recordings allow discussion of class labels and speaker separation; instrument recordings introduce timbre-related features; environmental sounds introduce event variability. The suitable library combination depends on the target and the representation used.

6.3 Speech recognition and pretrained audio models

SpeechBrain provides a PyTorch-based toolkit for speech and related audio research. OpenAI's Whisper repository provides an implementation and pretrained models for speech recognition and associated tasks. Hugging Face Transformers offers interfaces to supported pretrained audio and speech models, including model families such as Wav2Vec2 and Whisper. Availability of weights, language coverage and task support must be checked for the particular checkpoint.

Spoken-digit classification should be distinguished from full speech transcription. Similarly, a speech-recognition model does not automatically solve emotion recognition or audio anomaly detection. These tasks require suitable labels, evaluation measures and, where necessary, an additional classifier or task-specific adaptation.

6.4 Educational implications

The audio representation is an important part of the learning content. Students can compare waveform duration, sampling rate and spectrogram resolution before considering model complexity. Separating recordings by speaker or source can be more important for a fair experiment than increasing the number of model layers. Feature extraction parameters and segmentation rules need to be recorded together with the learning library.

Table 6.1. Library options for audio workflows

Function	Possible libraries	Main consideration
Decoding and storage	SoundFile; TorchCodec	Codec support, sampling rate and channels
Features and signal processing	librosa; SciPy; TorchAudio	Windowing, resampling and feature settings
Feature-based learning	scikit-learn; boosting libraries	Aggregation and training-set separation
Neural audio learning	TensorFlow/Keras; PyTorch	Waveform or time–frequency input
Speech and pretrained models	SpeechBrain; Whisper; Transformers	Language, task and checkpoint conditions

7. Video data

Video adds a temporal dimension to visual data and may also contain audio. Libraries for frame decoding, image processing, temporal sampling and sequence modelling therefore serve different functions. A detector applied to individual frames does not, by itself, model motion or recognise an action across time.

7.1 Decoding and frame preparation

OpenCV provides interfaces for reading video and working with frames. PyAV exposes FFmpeg-based media functions in Python, allowing access to streams and timing information. TorchCodec provides decoding functionality associated with PyTorch workflows. Supported codecs and execution behaviour depend on the installed software and the media file, so a file extension alone does not establish compatibility.

Image-processing libraries remain relevant after decoding. Frames can require resizing, colour conversion and normalisation before they are assembled into clips or passed to a detector. Frame rate, timestamps and the sampling strategy should remain explicit when the learning objective involves temporal behaviour.

7.2 Video classification and action recognition

TensorFlow/Keras and PyTorch can support temporal models, including convolutional features combined with recurrent layers and three-dimensional convolutional architectures. PyTorchVideo and MMAction2 provide additional video-oriented implementations and processing components. Their educational use may involve pretrained inference, fine-tuning or examination of a model configuration rather than training a large model from the beginning.

Hugging Face Transformers provides interfaces to supported video models, including VideoMAE implementations. Model-specific processors define how frames are sampled and transformed. Memory use depends on clip length, spatial resolution, batch size and architecture; the size of the compressed video file is therefore an incomplete guide to the training requirement.

7.3 Detection, tracking, segmentation and captioning

Frame-level object detection can use tools such as Ultralytics or Torchvision. Tracking adds associations between observations across frames; the original Deep SORT repository illustrates this additional stage. A tracker depends on detections, appearance features and temporal assumptions and should not be treated as equivalent to the detector itself. Repository-based research implementations may need adaptation to a current software environment.

Segmentation across video can combine image-level mask generation with temporal methods. Captioning introduces a further language-generation component and may use supported multimodal models through Transformers. These tasks are broader than ordinary frame classification and require corresponding annotations, evaluation procedures and checkpoint conditions. No single listed package guarantees all stages of every video task.

7.4 Working with the supplied assets

Collections such as Chinese Hand numerals, Traffic Signs Detection Europe Computer Vision Model and Popeye Cartoon may expose frames, annotations, sequences or encoded video. Library needs depend on the actual representation made available for an exercise. Annex 1 therefore describes possible roles for the libraries according to the supplied files and intended task.

Table 7.1. Library options for video workflows

Function	Possible libraries	Main consideration
Decoding and sampling	OpenCV; PyAV; TorchCodec	Codec, frame rate and timestamps
Temporal classification	General frameworks; PyTorchVideo; MMAction2; Transformers	Clip length, temporal input and memory
Detection and tracking	Ultralytics; Torchvision; Deep SORT code	Frame detections and cross-frame association
Segmentation	Image and temporal model implementations	Mask definitions and temporal consistency
Captioning	Supported multimodal models in Transformers	Video–language inputs and reference captions

8. Evaluation, interpretation and interactive learning

8.1 Evaluation and visualisation

Evaluation functions are distributed across general machine-learning libraries and modality-specific tools. scikit-learn can support many numerical and classification measures; detection and segmentation packages provide additional task-specific evaluation. Metrics should be selected according to the prediction target and annotation type rather than because a library displays them by default.

Matplotlib supports detailed control of static figures. Seaborn provides statistical visualisation built on Matplotlib, while Plotly supports interactive charts. These libraries can display distributions, confusion matrices, learning curves, predictions and errors, but the validity of the comparison depends on the underlying experiment. Interactive charts can also introduce rendering or export requirements that differ from static figures.

8.2 Explainability and experiment support

SHAP provides model-explanation methods and visualisations based on feature attribution. Captum provides interpretation methods for PyTorch models across different input types. These can support discussion of model behaviour and sensitivity, but an attribution should not be presented as evidence of causation or as an automatic confirmation that a prediction is correct.

Optuna provides facilities for hyperparameter optimisation. It can be considered where the educational activity includes search strategies or resource allocation across trials. A larger search is not necessarily more informative for teaching, and the final test data should remain separate from the data used to choose the parameters.

Joblib supports persistence, caching and parallel computation in Python workflows. It can help retain intermediate arrays or fitted estimators, while model-specific formats may be more appropriate for deep-learning models. Saving a model does not capture every dependency, and loading pickle-based objects requires a trusted source.

8.3 Notebook environments and interactive controls

Jupyter Notebook and JupyterLab provide environments that combine explanations, executable cells and outputs. They are working environments rather than model-training libraries. ipywidgets adds controls such as sliders, selectors and output panels that can be connected to Python functions. Possible educational uses include changing a preprocessing parameter, inspecting a sample or comparing stored predictions.

A live widget normally requires a compatible frontend and an active execution environment. Exporting a notebook to a static document does not guarantee the same behaviour. Precomputed results can reduce response time, while retraining inside a widget callback can require substantial resources. The appropriate design depends on the intended exercise and available computing resources.

9. Comparative assessment and educational applicability

9.1 Common and modality-specific functions

The review shows a common foundation of arrays, tables, model interfaces and visualisation tools, combined with specialised components for media decoding, annotations and task-specific models. This layered structure explains why a complete workflow often contains several libraries. It does not imply that all libraries discussed here need to be installed or used together.

Table 9.1. Comparative overview across the four modalities

Modality	Typical preparation	Learning options	Main technical issue
Numerical and tabular	Types, missing values, encoding, scaling, temporal windows	Estimator libraries, statistical tools, neural frameworks	Leakage, category treatment and split design
Image	Decoding, resizing, labels, boxes or masks	Classical vision and neural vision tools	Annotation consistency and image resolution
Audio and sound	Decoding, resampling, segmentation, spectral features	Feature-based models and audio/speech frameworks	Duration, sampling rate and speaker/source separation

Modality	Typical preparation	Learning options	Main technical issue
Video	Stream decoding, frame or clip sampling, temporal annotations	Frame models, tracking and temporal frameworks	Timing, memory and sequence-level separation

9.2 Comparative evaluation of the main learning libraries

Tables 9.2 and 9.3 describe the main learning libraries using the dimensions defined in Section 2. TensorFlow and Keras are presented separately because the execution framework and the higher-level model interface serve different roles, although they can be used together. Preparation and visualisation packages provide complementary functions within the same workflow.

Table 9.2. Functionality, ease of use and compatibility of the main learning libraries

Library	Main functionality	Ease of use	Model and data compatibility
TensorFlow	Tensor computation, automatic differentiation, training and data pipelines.	High-level workflows are accessible; custom execution requires more framework knowledge.	Dense, convolutional and recurrent models across all four modalities, with suitable input pipelines.
Keras	High-level model definition, training, evaluation and callbacks.	Concise interfaces support short teaching examples; custom components add complexity.	Runs on TensorFlow, JAX or PyTorch backends; portability depends on the operations and components used.
PyTorch	Tensor computation, automatic differentiation and custom neural-network training.	Explicit training steps aid inspection but require more programming decisions.	Neural architectures across all modalities; companion packages supply task-specific components.
scikit-learn	Classical estimators, preprocessing, pipelines, model selection and metrics.	A consistent estimator interface simplifies method comparison.	Tabular or extracted feature matrices; not a general CNN, recurrent-network or video-training framework.
XGBoost	Gradient-boosted trees for supervised prediction.	Familiar estimator-style use; tuning requires understanding boosting.	Structured features for regression and classification, including features extracted from media.
LightGBM	Tree-based gradient boosting with resource-oriented training options.	Estimator-style workflows; tree-growth settings need explanation.	Structured numerical or suitably represented categorical features; temporal tasks require an explicit feature design.

Library	Main functionality	Ease of use	Model and data compatibility
CatBoost	Tree boosting with categorical-feature handling.	Can reduce manual category encoding; feature declarations still require care.	Mixed numerical and categorical tables; additional supported feature types depend on the selected interface.
statsmodels	Statistical estimation, diagnostics and classical time-series models.	Accessible model interfaces, with statistical knowledge needed for interpretation.	Regression and forecasting with specified assumptions; not a general deep-learning engine.
sktime	Task-oriented time-series estimation and workflow composition.	A common task interface helps comparison; optional dependencies add setup.	Forecasting and other supported time-series tasks; estimator and data-format compatibility must be checked.
Transformers	Interfaces, processors and checkpoints for supported pretrained model families.	Convenient entry points for inference; adaptation requires model-specific knowledge.	Supported image, speech, video and multimodal architectures; backend support varies by model and release.

Table 9.3. Strengths, limitations and educational applicability of the same libraries

Library	Strengths	Limitations	Educational applicability
TensorFlow	Integrated training and input-pipeline tools.	Environment setup and specialised models can add dependencies.	Suitable for following a complete neural training workflow and then extending it.
Keras	Readable architectures and reusable training controls.	High-level calls can hide steps; backend-specific code reduces portability.	A suitable starting option for explaining layers, losses and learning curves.
PyTorch	Flexible control over model behaviour and optimisation.	Custom loops increase code and the scope for implementation errors.	Useful when learners need to inspect gradients, batches and optimisation explicitly.
scikit-learn	Comparable estimators and explicit preprocessing pipelines.	Requires feature extraction for raw media; deep architecture coverage is limited.	A suitable starting option for classical baselines, validation and evaluation.
XGBoost	A specialised alternative for nonlinear structured-data prediction.	Tuning and feature preparation remain necessary; no guaranteed dataset advantage.	Useful for comparing boosted trees with linear, forest and neural baselines.

Library	Strengths	Limitations	Educational applicability
LightGBM	Options intended for efficient tree-boosting workflows.	Efficiency and generalisation depend on data, settings and hardware.	Useful for discussing tree growth, resources and overfitting.
CatBoost	Integrated handling of declared categorical variables.	Does not remove leakage risks or the need to understand feature meanings.	Useful for contrasting internal category handling with explicit encoding.
statsmodels	Model summaries and diagnostic procedures.	Assumptions and temporal structure must be appropriate.	Useful for coefficient interpretation and statistical forecasting references.
sktime	Time-series task abstractions and estimator composition.	Capabilities and installation needs differ between estimators.	Useful for comparing temporal methods within a consistent workflow.
Transformers	Access to supported pretrained representations and processors.	Checkpoint availability, resources, labels and licences constrain use.	Useful for explaining pretrained inference, transfer learning and task adaptation.

For a classical baseline, scikit-learn is a suitable initial candidate because preprocessing and evaluation can be inspected within a consistent workflow. For a concise neural-network example, Keras with a compatible backend is a suitable initial candidate; PyTorch is an alternative where explicit training steps or a particular model ecosystem are important. TensorFlow provides a broader execution and data-pipeline environment when that functionality is needed. These are educational judgements, not evidence that one library is universally easier or more accurate.

9.3 Model families and possible Python libraries

Table 9.4 illustrates which Python libraries can be considered for different model families and tasks. The examples can help identify a starting point for an exercise. An architecture may require a separate model definition, companion package or pretrained checkpoint even when the underlying framework supplies its building blocks.

Table 9.4. Examples of model families and possible Python libraries

Methods or model families	Candidate libraries	Reason for considering the libraries
Tabular regression and classification: linear and logistic models, random forests, ensembles and stacking	scikit-learn; statsmodels where statistical inference is required	Estimator interfaces and pipelines support comparable baselines; statistical tools add coefficient diagnostics.

Methods or model families	Candidate libraries	Reason for considering the libraries
Gradient boosting: XGBoost and LightGBM; related structured-data alternatives	XGBoost; LightGBM; CatBoost as an additional alternative	Dedicated tree-boosting implementations support nonlinear structured-data prediction; category handling and tuning needs distinguish the options.
Artificial and deep neural networks for numerical prediction	Keras with a compatible backend; TensorFlow; PyTorch	Configurable dense networks support continuous, binary and multiclass targets with task-appropriate outputs and losses.
Time series: RNN, LSTM, GRU, temporal CNNs and transformer-based approaches	TensorFlow/Keras or PyTorch; a task-specific implementation for the selected architecture	Sequence-building and neural components support temporal models; data windows, horizon and architecture must be defined explicitly.
Time series: ARIMA/SARIMA, exponential smoothing and naive references	statsmodels; sktime; NumPy/pandas for simple reference forecasts	Statistical and task-oriented interfaces support forecasting comparisons, including simpler reference methods.
Clustering and representation: k-means, hierarchical clustering, DBSCAN, mixtures, PCA, t-SNE, UMAP and autoencoders	scikit-learn; UMAP; TensorFlow/Keras or PyTorch for autoencoders	Classical transformations and clustering have ready estimator interfaces; learned encoders require a neural implementation.
Images: CNNs, transfer learning, ResNet, VGG, Inception and EfficientNet	TensorFlow/Keras; PyTorch with Torchvision; appropriate pretrained implementations	Neural frameworks and model collections support image classifiers and transfer learning with model-specific preprocessing.
Images: YOLO, R-CNN-family models, FCNs, U-Net and DeepLab	Ultralytics for supported YOLO models; Torchvision for supported detectors and segmentation models; general frameworks for custom definitions	Task-specific interfaces can reduce implementation effort; the architecture and required annotation schema must match the package.
Audio: MFCC-based approaches, CNNs, LSTM/GRU, Wav2Vec and Whisper	librosa/SoundFile with scikit-learn; TensorFlow/Keras or PyTorch; Transformers, SpeechBrain or Whisper where appropriate	The tools cover distinct stages: audio access, representations, classification and supported pretrained speech tasks. MFCC is a feature representation, not a classifier.

Methods or model families	Candidate libraries	Reason for considering the libraries
Video: CNN plus recurrent models, 3D/spatio-temporal CNNs, video transformers and VideoMAE	TensorFlow/Keras; PyTorch with PyTorchVideo or MMAction2; Transformers for supported video models	General frameworks support temporal architectures, while specialised implementations can supply model definitions and clip processing.
Video detection, tracking, segmentation and captioning	Ultralytics or Torchvision; a tracker such as Deep SORT where required; supported temporal or multimodal implementations	Frame detection, association, masks and language outputs are different stages and require compatible annotations and model components.

9.4 Indicative recommendations for the four data modalities

The following recommendations identify library combinations to consider when work begins with the datasets. They describe suitable roles and alternatives rather than a final software selection. The choice will be made after examining the actual files, targets, annotations, data volume, split design, hardware and teaching objectives. It is not necessary to install or use all the listed options, and no single framework is prescribed for all partners.

Numerical and tabular data. NumPy and pandas provide an initial basis for arrays, tables and metadata, with SciPy for scientific calculations. scikit-learn is a suitable starting option for preprocessing, classical regression/classification, clustering and evaluation because it keeps these stages in a consistent interface. XGBoost, LightGBM and CatBoost can be considered as boosting alternatives. statsmodels and sktime are options for statistical modelling and temporal workflows; TensorFlow/Keras, PyTorch or a JAX-based implementation can be considered for neural models. UMAP and imbalanced-learn may be added for dimensionality reduction or training-set resampling. h5py or Xarray is relevant where the actual storage format or labelled dimensions justify it. These choices depend on whether the task emphasises predictive comparison, statistical interpretation or temporal structure.

Image data. Pillow or OpenCV can provide image access and transformations, while scikit-image can support classical processing and feature extraction. A suitable neural starting route is TensorFlow/Keras or PyTorch with Torchvision, selected according to the required classifier, pretrained weights and teaching style. For detection or segmentation, Ultralytics, supported Torchvision models or a custom framework implementation can be considered. Transformers offers supported vision and vision–language models; Segment Anything is an additional option for promptable masks rather than an automatic substitute for semantic labels. Framework transforms or the Albumentations ecosystem can support augmentation, with pycocotools where COCO-style annotations are used. This combination separates file preparation from learning and annotation-specific evaluation.

Audio and sound data. SoundFile with librosa is a suitable initial combination for inspecting recordings and constructing waveform, spectral or MFCC representations; SciPy can add signal

processing. scikit-learn or boosting libraries can then provide feature-based baselines. TensorFlow/Keras or PyTorch can be considered for neural waveform, spectrogram or recurrent models, with TorchAudio and TorchCodec where their release-specific roles fit a PyTorch workflow. SpeechBrain, Transformers or Whisper can be considered when supported pretrained audio or speech tasks are actually required. The target must distinguish sound or digit classification from transcription, emotion recognition and anomaly detection. Recording duration, segmentation and speaker/source separation should guide the choice of representation and implementation.

Video data. OpenCV, PyAV or TorchCodec can be considered for decoding and frame/clip access, with image tools used for frame preparation. TensorFlow/Keras or PyTorch is a suitable candidate for custom CNN–recurrent or 3D models; PyTorchVideo, MMAction2 or supported Transformers models can provide specialised temporal implementations. Ultralytics or Torchvision can support frame-level detection, with a separately evaluated tracking stage where needed. Segmentation and captioning require task-specific temporal or multimodal components and suitable annotations. This combination is appropriate because decoding, frame analysis and temporal learning are separate functions. The final choice depends on whether the supplied assets are still frames or actual sequences, and on clip length, timestamps and available memory.

Across all four modalities, Matplotlib, Seaborn or Plotly can support visualisation; Jupyter Notebook/JupyterLab with ipywidgets can support interactive exercises. SHAP or Captum may be considered for suitable explanation tasks, Optuna for parameter-search exercises and Joblib for compatible estimator or intermediate-data persistence. Hugging Face Datasets may help access supported collections. These supporting choices should follow the selected workflow rather than determine the learning model.

Before confirming a combination, a representative data sample should be used to check loading, preprocessing, target/annotation compatibility, a small model run and evaluation outputs. Installation, resource needs, software and checkpoint licences should also be checked. The selected libraries, versions and reasons for selection can then be documented in the implementation activities. A successful small run establishes basic feasibility; it does not establish comparative model performance.

9.5 Applicability at different educational levels

At an introductory level, a small dataset, a visible transformation and a short model workflow can help students connect inputs, targets and outputs. NumPy, pandas, a limited estimator interface or a concise neural-network definition can each support such an exercise. Suitability depends on the amount of preparation that the learner must understand, rather than on a permanent classification of one library as introductory.

Intermediate activities can connect several stages: preprocessing, baseline modelling, parameter variation and interpretation. Image augmentation, audio feature extraction and chronological validation introduce modality-specific decisions. Advanced activities can examine custom architectures, multimodal representations, large-scale data access or specialised annotations. The same framework can serve all three levels when the task is scoped differently.

9.6 Trade-offs between alternative approaches

High-level interfaces reduce the amount of code required to produce an initial result, but may conceal steps that an exercise intends to explain. Lower-level interfaces expose those steps while increasing implementation demands. Pretrained models reduce some training requirements, but introduce assumptions about preprocessing, label space, source data and checkpoint licences.

A broader package may simplify access to several models, while a specialised implementation may provide closer control over one method. Neither property establishes universal superiority. The relevant comparison is between complete workflows for the same educational task, including preparation, evaluation, installation and explanation.

9.7 Implications for the project

The overview provides possible starting points and alternatives for different task families and data characteristics. The final choice will be made when work begins with the actual data, after checking the target, annotations, preprocessing needs, model implementation and execution environment. Partners can choose the combination that fits their tasks and teaching needs.

As implementation proceeds, the libraries actually used, their versions and their roles can be recorded alongside the code and interactive examples.

10. Technical requirements and reproducibility

10.1 Dependencies and execution environments

A Python library may depend on compiled components, media codecs, numerical runtimes or another learning framework. The installation requirement is therefore broader than the import statement shown in a notebook. Operating system, processor architecture, Python version and accelerator support can affect whether a particular combination is available.

Python's venv module provides isolated environments, while pip supports package installation and requirements specifications. These are possible mechanisms for recording and reproducing dependencies. They do not remove the need to check compatibility or ensure identical numerical results on different hardware.

10.2 Version information and resource requirements

Reproducibility information can include the Python version, package versions, operating system, CPU or accelerator, relevant drivers, random seeds and data-preparation settings. Dataset identity, splits and checkpoint provenance are equally important. This report does not provide a final common requirements file because the indicative choices in Section 9 must be refined for the separate implementations after the actual data and execution requirements have been examined.

Small examples can often be structured for CPU execution, while large image, audio and especially video models may require more memory and acceleration. Pretrained inference and full training have

different requirements. A GPU-capable library does not guarantee that a GPU is available or that a particular model fits in its memory.

10.3 Licences and model assets

Software licences, dataset licences and pretrained-weight conditions are separate records. An open-source library may be used with a checkpoint or dataset that has additional restrictions. Conditions can also differ between related package distributions and releases. The technical review therefore identifies licence checking as a criterion without certifying that all possible combinations are suitable for every form of distribution.

Dataset access and redistribution conditions should be checked separately from software licences. An accessible download or a working loader does not, by itself, establish permission to redistribute institutional data, recordings or model assets.

10.4 Use of the overview

The descriptions provide an initial guide to available tools and their possible roles. When a workflow is implemented, a representative data sample can be used to check compatibility, resource needs and the required outputs before the full exercise is developed.

11. Conclusion

The Python ecosystem provides multiple routes for preparing data, implementing AI methods and presenting results across numerical, image, audio and video applications. General scientific libraries support shared operations, learning frameworks support a broad range of model structures, and specialised packages address the formats, annotations and temporal characteristics of particular modalities.

The central finding is that library suitability is task-dependent. Functionality, interface complexity, compatibility, computational demands and conditions of use need to be considered together. No single package covers every stage equally, and the presence of a documented implementation does not establish its suitability for a particular dataset or teaching objective.

The overview brings these options into a common descriptive structure. The comparison of libraries and the possible combinations for each modality can support initial planning. Final decisions will be made when work begins with the actual data and the requirements of each implementation become clear. The selected libraries and their configuration can then be recorded with the developed code and educational examples.

Annex 1. Library options for selected datasets

The following examples illustrate how selected datasets and their data characteristics can be connected to Python tools. The actual files, annotations and intended tasks determine which functions are relevant. Library choices can be adjusted as these requirements become clear.

Table A1. Possible library roles for the 12 datasets

Partner and modality	Dataset	Possible library roles	Preparation or compatibility issue
UNILJ / video	Chinese Hand numerals	OpenCV or Pillow for frames; neural frameworks for visual classification; video tools if sequences are supplied	Confirm whether the supplied assets are images, sequences or encoded video and identify labels.
UNILJ / video	Traffic Signs Detection Europe Computer Vision Model	OpenCV; Ultralytics or Torchvision; annotation parsers	Confirm resource type, box schema and class mapping; temporal decoding only for video inputs.
UNILJ / video	Popeye Cartoon	OpenCV or PyAV for video; Pillow for frames; general visual or temporal frameworks	Identify the supplied representation, task and annotation level before choosing a model interface.
UoM / audio	FSDD – Free Spoken Digit Dataset	SoundFile; librosa; scikit-learn or neural frameworks	Separate digit classification from transcription and account for speaker identity.
UoM / audio	Sopele Sounds	SoundFile; librosa; SciPy; feature-based or neural classifiers	Check duration, sampling rate, recording groups and instrument labels.
UoM / audio	ESC-50	SoundFile; librosa; TorchAudio; general learning frameworks	Preserve the documented evaluation structure and prevent related recordings from crossing splits.
UNSA / numerical	Hourly meteorological and air-quality data for Sarajevo	pandas; NumPy; statsmodels; sktime; neural or feature-based forecasting tools	Define timestamps, targets and horizon; preserve temporal order and institutional access conditions.
UNSA / numerical	High-performance concrete mix designs and compressive strength	pandas; scikit-learn; boosting libraries; neural frameworks	Check spreadsheet or table format, units, target definition and preprocessing.

Partner and modality	Dataset	Possible library roles	Preparation or compatibility issue
UNSA / numerical	Synthetic/enriched loan-applicant dataset	pandas; scikit-learn; CatBoost or other boosting tools; imbalanced-learn; neural frameworks	Check categories, class balance and split design; evaluate fairness in the context of the task.
UNBG / image	CarLogos and CarIDs	Pillow; OpenCV; Torchvision; TensorFlow/Keras or PyTorch	Identify labels and whether the intended task is logo, class or identity recognition.
UNBG / image	Brand logos and icons dataset	Pillow; OpenCV; image-classification frameworks; detection tools if box labels exist	Check folder labels, image variation and annotation availability.
UNBG / image	CIFAR-10 image classification	Dataset loaders; NumPy; TensorFlow/Keras; PyTorch and Torchvision	Use the supplied split and label definitions and document input normalisation.

The examples remain open to equivalent implementations. Where a task requires a method not supplied directly by a listed package, a general learning framework or an additional implementation may provide an alternative. Final choices can be made and documented during work with the actual data.