Here is the hierarchically formatted knowledge base based on the content you provided:

**1. Python Programming Basics (🔑 High Priority)**

* **Data types and Data structures**
  + **Known:**
    - Numeric types → int, float
    - Boolean type → bool
    - Creating lists → [], list()
    - Indexing in list
    - Searching → in operator
    - Creating tuples → tuple()
    - String concatenation (+)
    - Changing case in strings → upper(), lower()
    - Whitespace handling in strings → strip()
    - Splitting in strings → split()
    - Replacement in strings → replace()
    - Formatting strings → format(), % formatting
    - Common built-ins in string → len()
    - String methods for starts/ends → startswith(), endswith()
    - Type checking → type()
    - NoneType → None
    - Built-in functions interacting with data types → len()
    - Creating dictionaries → {}
    - Accessing values dictionary → dict[key], get()
    - Iterating dictionary → keys(), values(), items()
    - Creating sets → { }, set()
    - Membership tests in sets → in operator
    - Type casting/conversion → int(), float(), str(), list(), dict(), set()
  + **Unknown:**
    - Numeric types → complex
    - Sequence types → str, list, tuple, range
    - Slicing list
    - Creating lists → list()
    - Adding elements to list → append(), extend(), insert()
    - Removing elements from list → remove(), pop(), clear()
    - Searching list → index(), count()
    - Sorting and reversing list → sort(), sorted(), reverse()
    - Copying lists → shallow vs deep copy, copy() method
    - Common built-ins in list → len(), sum(), min(), max(), any(), all()
    - Creating tuples → tuple()
    - Tuple immutability (contrast with lists)
    - count() and index() methods tuples
    - Use cases of tuples (immutable data, dictionary keys)
    - String repetition (\*)
    - String Searching → in operator, find(), index(), rfind()
    - Changing case in string → title(), capitalize(), swapcase()
    - Splitting and joining → splitlines(), join()
    - Checking conditions in strings → isdigit(), isalpha(), isalnum(), isnumeric(), isspace()
    - Formatting strings → format(), % formatting
    - Encoding and decoding → encode(), decode()
    - String immutability
    - Common built-ins in string → len(), ord(), chr()
    - Raw strings (r"...")
    - Escape sequences in strings (\n, \t, etc.)
    - Partitioning strings → partition(), rpartition()
    - Justification/padding in string → ljust(), rjust(), center(), zfill()
    - Mapping type → dict
    - Accessing values → get()
    - Adding/updating key dictionary-value pairs
    - Removing items from dictionary → pop(), popitem(), clear()
    - Checking membership → in operator (keys)
    - Copying dictionaries → copy(), deepcopy
    - Default values → defaultdict (collections)
    - Merging dictionaries (| operator in Python 3.9+)
    - Common built-ins → len(), sorted()
    - Adding/removing elements to set → add(), remove(), discard(), pop(), clear()
    - Set operations → union(), intersection(), difference(), symmetric\_difference()
    - Copying sets → copy()
    - Set comprehension
    - frozenset → immutability, use cases
    - Built-in functions in sets → len(), min(), max()
    - Binary types → bytes, bytearray, memoryview
    - Type checking → isinstance()
    - Immutability vs mutability (str vs list, tuple vs dict/set)
    - Built-in functions interacting with data types → sum(), min(), max(), sorted(), etc.
    - Data type hierarchy in Python (everything is an object)
* **Loops & conditionals** → for, while, if/else
  + **Known:**
    - if, elif, else statements → syntax and flow
    - Nested conditionals
    - Loop control statements → break, continue, pass
    - Nested loops
    - Iterating with range()
    - Iterating with enumerate()
  + **Unknown:**
    - Ternary operator (inline if) → <value\_if\_true> if condition else <value\_if\_false>
    - for loops → iterating over lists, tuples, dicts, sets, strings
    - while loops → condition-based iteration
    - else clause in loops (executed if loop completes normally)
    - Iterating with zip()
    - List/dict/set comprehensions one-liners, nested
    - Common pitfalls → infinite while loops, modifying collection while iterating
* **Functions** → def, arguments, return
  + **Known:**
    - Defining functions with def
    - Default parameter values
  + **Unknown:**
    - Function naming conventions (PEP8 style)
    - Function arguments → positional, keyword
    - \*args and \*\*kwargs (variable arguments)
    - Returning values → return vs return None
    - Returning multiple values (tuple packing/unpacking)
    - Scope rules → local vs global variables
    - Mutable vs immutable arguments (side effects)
    - Docstrings → documenting functions
    - Type hints (Python 3.5+)
    - Nested functions
    - First-class functions (functions as objects, passing functions as args)
    - Higher-order functions → map(), filter(), reduce()
    - Recursion (base case, stack depth)
    - Best practices → single responsibility, readability, avoiding long argument lists
* **Decorators** → @staticmethod, custom
  + **Known:**
    - (No known concepts provided)
  + **Unknown:**
    - Concept of decorators (functions modifying other functions)
    - Syntax → @decorator\_name above function definition
    - Built-in decorators → @staticmethod, @classmethod, @property
    - Creating custom decorators (functions returning functions)
    - functools.wraps → preserving metadata (name, docstring)
    - Stacking multiple decorators
    - Using decorators with arguments
    - Decorators for logging, timing, caching
    - Practical ML/AI use cases → function tracing, enforcing input/output formats
    - Common pitfalls → readability, debugging wrapped functions
* **Generators & Iterators**
  + **Known:**
    - (No known concepts provided)
  + **Unknown:**
    - Concept of iterables and iterators → **iter**(), **next**()
    - Using iter() and next()
    - StopIteration exception
    - For loops internally using iterators
    - Generator functions → using yield
    - Generator expressions (similar to list comprehensions)
    - Difference between return vs yield
    - Infinite generators
    - Memory efficiency benefits of generators
    - Chaining generators
    - Sending values into generators → send()
    - Closing generators → close()
    - Throwing exceptions into generators
    - Practical use cases → reading large files, data streaming, ML pipelines
* **Lambda functions**
  + **Known:**
    - (No known concepts provided)
  + **Unknown:**
    - Concept of anonymous functions (no def, single expression)
    - Syntax → lambda arguments: expression
    - Returning values implicitly (no return keyword)
    - Using lambda with built-ins → sorted(), map(), filter(), reduce()
    - Multiple arguments in lambda
    - Nested lambda functions
    - Lambda as key functions (e.g., sorting by second element in tuple)
    - Storing lambdas in variables
    - Limitations → single expression only, readability issues
    - Differences between def vs lambda (scope, reusability, docstrings)
    - Practical ML/AI use cases → feature transformations, inline functions in pipelines
* **OOP** → classes, inheritance, polymorphism, dunder methods
  + **Known:**
    - Creating objects (instances of classes)
  + **Unknown:**
    - Creating classes → class keyword, **init** constructor
    - Instance attributes vs class attributes
    - Methods → instance methods (self), class methods (@classmethod), static methods (@staticmethod)
    - Inheritance basics → single inheritance
    - Multiple inheritance → method resolution order (MRO)
    - Polymorphism → same method name with different implementations
    - Encapsulation → public, protected (\_), private (\_\_)
    - Abstraction → abstract classes and methods (abc module)
    - Overriding methods
    - Operator overloading (dunder methods → **add**, **sub**, **str**, **len**, etc.)
    - **repr** vs **str** difference
    - **eq**, **lt**, **gt** (comparison dunders)
    - **call** method → making objects callable
    - **enter**, **exit** → context manager protocol (with statement)
    - **iter**, **next** → iterable protocol
    - Class variables vs instance variables
    - super() function for inheritance
    - Composition vs inheritance
    - Practical ML/AI use cases → custom model classes, dataset loaders, training loops
* **Error handling** → try/except/finally
  + **Known:**
    - Basic try/except block
    - Multiple except clauses for different exceptions
    - Catching specific exceptions (ValueError, TypeError, KeyError, etc.)
    - Raising exceptions manually → raise
  + **Unknown:**
    - Using Exception hierarchy
    - finally block → cleanup actions (files, DB connections)
    - else block → runs only if no exceptions raised
    - Creating custom exceptions (subclassing Exception)
    - Using as to access exception objects
    - Best practices → catching specific exceptions, avoiding bare except
    - Logging exceptions
    - Suppressing exceptions (contextlib.suppress)
    - Practical ML/AI use cases → handling missing files, corrupt data, failed API calls
* **Async/await** → coroutines, event loops
  + **Known:**
    - (No known concepts provided)
  + **Unknown:**
    - Concept of asynchronous programming vs synchronous
    - Coroutines → functions defined with async def
    - Await keyword → pausing execution until coroutine completes
    - Event loop → managing and scheduling coroutines
    - Running coroutines → asyncio.run(), loop.run\_until\_complete()
    - Tasks and futures → asyncio.create\_task(), asyncio.Future
    - Gathering coroutines → asyncio.gather(), asyncio.wait()
    - async for and async with syntax
    - Error handling in async code
    - Blocking vs non-blocking operations
    - Mixing async with sync code
    - Practical ML/AI use cases → concurrent API calls, data streaming, serving multiple model requests
    - Common pitfalls → forgetting await, mixing blocking I/O with async, nested event loops
* **Parallelization** → multiprocessing, threading, concurrent futures
  + **Known:**
    - (No known concepts provided)
  + **Unknown:**
    - Concept of concurrency vs parallelism
    - Global Interpreter Lock (GIL) → limitation for threads
    - Threading module → Thread class, start(), join()
    - Thread safety → locks, race conditions
    - Multiprocessing module → Process class, start(), join()
    - Sharing data between processes → Queue, Pipe, Manager
    - concurrent.futures → ThreadPoolExecutor, ProcessPoolExecutor
    - Submitting tasks with submit() and map()
    - Asynchronous result handling with Future objects
    - Using multiprocessing.Pool
    - Performance trade-offs → CPU-bound vs I/O-bound tasks
    - Best practices for parallelization in Python
    - Practical ML/AI use cases → data preprocessing pipelines, parallel experiments, hyperparameter tuning
* **Modules & Packages**
  + **Known:**
    - (No known concepts provided)
  + **Unknown:**
    - Concept of a module → a Python file (.py) with functions, classes, variables
    - Importing modules → import, from ... import, import as
    - Standard library modules (os, sys, math, random, datetime, pathlib, json, re, etc.)
    - Creating a custom module (my\_module.py)
    - Reloading modules → importlib.reload()
    - Concept of a package → folder with **init**.py file
    - Importing from packages → from package import module
    - Nested packages and submodules
    - Relative vs absolute imports
    - **all** attribute in packages
    - Organizing large projects with packages
    - Using pip to install external packages
    - Virtual environments (venv, conda) for managing dependencies
    - Requirements.txt and dependency management
    - Practical ML/AI use cases → structuring ML projects into data, models, utils, training modules
* File handling (read/write CSV, JSON, text)**:**

**Known:**

**Unknown:**

* Opening and closing files: open(), close(), with statement (context managers)
* File modes: 'r', 'w', 'a', 'b', 'x', combinations ('rb', 'wb')
* Reading text files: read(), readline(), readlines()
* Writing text files: write(), writelines()
* Appending to files
* File pointer control: seek(), tell()
* Exception handling with file operations
* Working with directories: os, pathlib
* CSV files: csv.reader, csv.writer, csv.DictReader, csv.DictWriter
* CSV with pandas: read\_csv(), to\_csv()
* JSON files: json.load(), json.loads(), json.dump(), json.dumps()
* Handling encoding (UTF-8, etc.)
* Binary files basics (images, model files)
* Practical ML/AI use cases: dataset loading, saving preprocessing results, storing experiment logs

**Notes:**

* None
* Virtual Environments (venv, conda)

**Known:**

* Concept of isolated environments: avoiding dependency conflicts
* Creating virtual environments with venv: python -m venv env\_name
* Activating and deactivating venv: source/bin/activate, deactivate (Linux/Mac), .\\Scripts\\activate (Windows)
* Installing packages inside venv: pip install
* Installing from requirements.txt: pip install -r requirements.txt

**Unknown:**

* Checking installed packages: pip list, pip freeze
* Saving dependencies: requirements.txt (pip freeze > requirements.txt)
* Creating environments with conda: conda create -n env\_name python=3.x
* Activating/deactivating conda env: conda activate, conda deactivate
* Managing packages with conda: conda install, conda list, conda remove
* Conda vs pip differences
* Exporting conda environments: conda env export > environment.yml
* Recreating env from YAML: conda env create -f environment.yml
* Best practices: one environment per project, documenting dependencies
* Practical ML/AI use cases: ensuring reproducibility, GPU/CPU dependency management

**Notes:**

* None

**2. Core Foundations (🔑 High Priority, learn alongside Python)**

* **Mathematics**
  + **Linear Algebra** → vectors, matrices, dot product, transpose, eigenvalues/eigenvectors

Known:

Unknown:

* + - * - Vectors → definition, notation, vector addition, scalar multiplication
      * - Vector norms → L1, L2, Euclidean distance
      * - Matrices → definition, notation, dimensions (m × n)
      * - Matrix addition, subtraction, scalar multiplication
      * - Matrix multiplication rules (row × column, dimensionality conditions)
      * - Identity matrix and zero matrix
      * - Transpose of a matrix → A^T
      * - Symmetric matrices → A = A^T
      * - Dot product of vectors → algebraic definition, geometric interpretation (cosθ)
      * - Matrix-vector multiplication
      * - Matrix-matrix multiplication
      * - Determinant of a matrix
      * - Inverse of a matrix → A⁻¹, conditions for invertibility
      * - Rank of a matrix
      * - Linear independence, span, basis
      * - Eigenvalues and eigenvectors → definition, interpretation
      * - Characteristic polynomial
      * - Diagonalization of a matrix
      * - Orthogonality and orthonormality
      * - Singular Value Decomposition (SVD) basics
      * - Practical AI/ML applications:
      * - Representing datasets as matrices
      * - Dot product → similarity (cosine similarity, embeddings)
      * - Eigenvectors/eigenvalues → PCA for dimensionality reduction
      * - SVD → recommendation systems, data compression
      * - Matrix multiplication → forward/backpropagation in neural networks

Notes:

* + - * -
  + **Probability** → distributions (Normal, Bernoulli, Binomial), Bayes theorem

Known:

* + - -

Unknown:

* + - - Probability basics → sample space, events, outcomes
    - - Random variables → discrete vs continuous
    - - Probability rules → addition rule, multiplication rule
    - - Conditional probability → P(A|B)
    - - Independent vs dependent events
    - - Bernoulli distribution → single trial success/failure
    - - Binomial distribution → multiple Bernoulli trials
    - - Normal (Gaussian) distribution → mean, variance, standard deviation
    - - Properties of Normal distribution (68–95–99.7 rule)
    - - Standard Normal distribution (Z-scores)
    - - Uniform distribution
    - - Probability mass function (PMF) vs probability density function (PDF)
    - - Cumulative distribution function (CDF)
    - - Expectation and variance of distributions
    - - Law of large numbers
    - - Central Limit Theorem
    - - Bayes theorem → P(A|B) = P(B|A)P(A) / P(B)
    - - Prior, likelihood, posterior
    - - Applications of Bayes theorem in ML/AI:
    - - Naive Bayes classifier
    - - Bayesian inference
    - - Probabilistic graphical models
    - - Updating beliefs with new evidence

Notes:

* + - -
  + **Statistics** → mean, variance, std dev, correlation, hypothesis testing, p-value

Known:

* + - -

Unknown:

* + - - Descriptive statistics → mean, median, mode
    - - Variance and standard deviation (spread of data)
    - - Range, quartiles, interquartile range (IQR)
    - - Skewness and kurtosis
    - - Correlation → Pearson, Spearman, Kendall
    - - Covariance → relationship between variables
    - - Population vs sample statistics
    - - Inferential statistics → sampling, estimators, confidence intervals
    - - Central Limit Theorem (relation to statistics)
    - - Hypothesis testing → null hypothesis (H0), alternative hypothesis (H1)
    - - Test statistics → z-test, t-test, chi-square test, ANOVA (overview)
    - - p-value → interpretation, significance level (α)
    - - Type I vs Type II errors
    - - Confidence intervals and margins of error
    - - Effect size
    - - Practical AI/ML applications:
    - - Correlation → feature selection
    - - Variance/std dev → understanding data spread
    - - Hypothesis testing → A/B testing, model comparison
    - - p-values → statistical significance of model results
    - - Confidence intervals → uncertainty in predictions

Notes:

* + - -
  + **Calculus** → derivatives, partial derivatives, gradients

Known:

* + - -

Unknown:

* + - - Concept of a function → f(x), input → output mapping
    - - Limits → definition of derivative as a limit
    - - Derivative → rate of change, slope of tangent line
    - - Common differentiation rules → power rule, product rule, quotient rule, chain rule
    - - Higher-order derivatives → second derivative, concavity
    - - Partial derivatives → functions of multiple variables
    - - Gradient → vector of partial derivatives
    - - Directional derivatives
    - - Jacobian matrix → derivatives of vector-valued functions
    - - Hessian matrix → second-order partial derivatives
    - - Critical points → maxima, minima, saddle points
    - - Gradient descent → using gradients for optimization
    - - Role of derivatives in backpropagation
    - - Practical AI/ML applications:
    - - Training neural networks (optimization of loss functions)
    - - Convex vs non-convex optimization
    - - Feature scaling impact on gradients
    - - Understanding vanishing/exploding gradients

Notes:

* + - -
  + Pandas

Known:

* + - -

Unknown:

* + - - Basics:
    - - Importing Pandas (import pandas as pd)
    - - Data structures:
    - - Series (1D labeled array)
    - - DataFrame (2D labeled table)
    - - Creating DataFrames → from dict, CSV, Excel, NumPy
    - - Data exploration:
    - - head(), tail(), info(), describe()
    - - shape, columns, dtypes
    - - Indexing & selection:
    - - .loc[] → label-based
    - - .iloc[] → position-based
    - - Boolean indexing & filtering
    - - Data cleaning:
    - - Handling missing values → isnull(), dropna(), fillna()
    - - Duplicates → duplicated(), drop\_duplicates()
    - - Renaming columns, resetting index
    - - Data transformation:
    - - apply(), map(), applymap()
    - - String operations → .str methods
    - - Type conversions → astype()
    - - Aggregation & grouping:
    - - groupby(), agg(), pivot\_table()
    - - value\_counts()
    - - Merging & joining:
    - - merge(), join(), concat()
    - - Sorting:
    - - sort\_values(), sort\_index()
    - - Time series handling:
    - - pd.to\_datetime(), resample(), shift(), rolling()
    - - Input/Output:
    - - read\_csv(), to\_csv()
    - - read\_excel(), to\_excel()
    - - read\_sql(), read\_json()
    - - Practical AI/ML applications:
    - - Data cleaning & preprocessing
    - - Feature engineering
    - - Exploratory Data Analysis (EDA)
    - - Pipeline integration with NumPy, scikit-learn

Notes:

* + - -
  + Numpy

Known:

* + - -

Unknown:

* + - - Basics:
    - - Importing NumPy (import numpy as np)
    - - Creating arrays → np.array(), np.zeros(), np.ones(), np.arange(), np.linspace()
    - - Array attributes → shape, dtype, ndim, size
    - - Array operations:
    - - Indexing & slicing → 1D, 2D, advanced indexing
    - - Boolean indexing & masking
    - - Reshaping → reshape(), ravel(), flatten()
    - - Concatenation & splitting → hstack(), vstack(), concatenate(), split()
    - - Mathematical operations:
    - - Elementwise ops (+, -, \*, /, \*\*)
    - - Broadcasting rules
    - - Universal functions (ufuncs) → np.sqrt(), np.exp(), np.log()
    - - Linear algebra:
    - - Dot product → np.dot(), @ operator
    - - Matrix multiplication → np.matmul()
    - - Transpose → .T
    - - Determinant → np.linalg.det()
    - - Inverse → np.linalg.inv()
    - - Eigenvalues & eigenvectors → np.linalg.eig()
    - - Statistics:
    - - Mean, median, std dev, variance → np.mean(), np.median(), np.std(), np.var()
    - - Correlation → np.corrcoef()
    - - Random module:
    - - Random numbers → np.random.rand(), np.random.randn(), np.random.randint()
    - - Random choice, shuffle, seed
    - - Useful utilities:
    - - np.where(), np.unique(), np.sort(), np.argsort()
    - - np.argmax(), np.argmin()
    - - Practical AI/ML applications:
    - - Core array computations in ML pipelines
    - - Data preprocessing, feature engineering
    - - Fast matrix ops for ML algorithms

Notes:

* + - -
* **Optimization Theory**
  + Convex vs non-convex

Known:

* + - -

Unknown:

* + - - Concept of optimization → objective/cost/loss function
    - - Convex function → definition, properties (global minimum = local minimum)
    - - Convex sets → line segment property
    - - Examples of convex functions (linear, quadratic)
    - - Non-convex functions → multiple local minima, saddle points
    - - Visual intuition of convex vs non-convex
    - - Importance of convexity in optimization
    - - Role of derivatives/gradients in convex optimization
    - - Global vs local minima
    - - Challenges with non-convex optimization (neural networks)
    - - Convex optimization algorithms → gradient descent, Newton’s method
    - - Non-convex optimization heuristics → stochastic gradient descent (SGD), momentum, Adam optimizer
    - - Regularization and convexity (L1, L2 penalties)
    - - Practical AI/ML applications:
    - - Linear regression → convex optimization
    - - Logistic regression → convex optimization
    - - Neural networks → non-convex optimization
    - - Why deep learning still works despite non-convexity (many good minima)

Notes:

* + - -
  + Gradient-based methods

Known:

* + - -
    - Unknown:
    - - Gradient descent → concept, update rule (θ = θ - α∇J(θ))
    - - Learning rate (α) → effect of too small/too large values
    - - Batch gradient descent
    - - Stochastic gradient descent (SGD)
    - - Mini-batch gradient descent
    - - Convergence criteria → stopping when gradient ≈ 0
    - - Momentum → accelerating gradients in relevant directions
    - - Nesterov accelerated gradient (NAG)
    - - Adaptive learning rate methods:
    - - AdaGrad
    - - RMSProp
    - - Adam optimizer (combining momentum + RMSProp)
    - - Gradient clipping → handling exploding gradients
    - - Vanishing gradients → causes and remedies
    - - Second-order methods → Newton’s method, quasi-Newton (L-BFGS)
    - - Line search methods
    - - Practical AI/ML applications:
    - - Training linear/logistic regression models
    - - Training neural networks with backpropagation
    - - Optimizing high-dimensional loss landscapes
    - - Choosing optimizers in deep learning frameworks (PyTorch, TensorFlow)

Notes:

* + - -

* + Constrained optimization basics

Known:

* + - -

Unknown:

* + - - Concept of constraints → equality vs inequality constraints
    - - Feasible region → solutions that satisfy constraints
    - - Constrained optimization vs unconstrained optimization
    - - Lagrange multipliers → method for handling equality constraints
    - - Lagrangian function → objective + multipliers × constraints
    - - Karush-Kuhn-Tucker (KKT) conditions → necessary conditions for optimality
    - - Slack variables in inequality constraints
    - - Duality → primal vs dual problems
    - - Convex constrained optimization → guarantees of global optimum
    - - Non-convex constrained optimization → challenges
    - - Numerical methods for constrained optimization
    - - Practical AI/ML applications:
    - - Regularization → equivalent to constrained optimization (e.g., weight norm ≤ c)
    - - Support Vector Machines (SVM) → optimization with constraints
    - - Resource allocation problems in AI
    - - Hyperparameter tuning under constraints

Notes:

* + - -
* **Data Handling**
  + Data gathering (APIs, web scraping basics, SQL)

Known:

* + - -

Unknown:

* + - - Data sources → structured (databases), semi-structured (JSON, XML), unstructured (text, images)
    - - APIs → concept of REST APIs, endpoints, requests & responses
    - - HTTP methods → GET, POST, PUT, DELETE
    - - Python requests library → requests.get(), requests.post(), response.json()
    - - Authentication → API keys, tokens, OAuth
    - - Rate limiting and pagination
    - - Handling JSON and XML responses
    - - Web scraping basics → HTML structure, DOM, tags
    - - Using BeautifulSoup → parsing HTML, find(), find\_all()
    - - Using requests + BeautifulSoup together
    - - Scrapy basics for large-scale scraping
    - - Handling dynamic content → Selenium for JavaScript-heavy pages
    - - Ethics & legality of scraping → robots.txt, rate limits
    - - SQL basics → SELECT, INSERT, UPDATE, DELETE
    - - Filtering data → WHERE, ORDER BY, LIMIT
    - - Aggregations → COUNT, SUM, AVG, GROUP BY, HAVING
    - - Joins → INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN
    - - Subqueries and nested SELECT
    - - SQL in Python → sqlite3, SQLAlchemy, pandas.read\_sql()
    - - Practical AI/ML applications:
    - - Gathering datasets via APIs (Twitter, Reddit, Kaggle APIs)
    - - Scraping websites for text/image datasets
    - - Extracting data from relational databases for training
    - - Automating ETL pipelines
    - - Data collection for real-time AI systems

Notes:

* + - -
  + Data cleaning (missing values, outliers, normalization)

Known:

* + - -
    - Unknown:
    - - Identifying missing values → NaN, None, null detection
    - - Handling missing values:
    - - Removing rows/columns with missing values
    - - Imputation → mean, median, mode
    - - Forward fill, backward fill
    - - Interpolation methods
    - - Advanced imputation → KNN imputer, regression imputation
    - - Detecting outliers:
    - - Statistical methods → z-score, IQR rule
    - - Visualization → boxplots, scatterplots
    - - Domain-specific rules
    - - Handling outliers:
    - - Removal
    - - Capping/winsorization
    - - Transformation (log, sqrt)
    - - Data normalization vs standardization:
    - - Min-Max scaling (0–1 normalization)
    - - Z-score standardization (mean=0, std=1)
    - - Robust scaling (median & IQR based)
    - - Encoding categorical variables:
    - - One-hot encoding
    - - Label encoding
    - - Ordinal encoding
    - - Detecting and handling duplicates
    - - Data type conversions (numeric, categorical, datetime)
    - - Dealing with inconsistent formatting (e.g., string casing, whitespace, date formats)
    - - Practical AI/ML applications:
    - - Handling missing values in healthcare/finance datasets
    - - Outlier removal in anomaly detection
    - - Normalization before distance-based ML models (k-NN, SVM)
    - - Encoding categorical features for regression/classification

Notes:

* + - -
  + Data processing (scaling, encoding, feature engineering)

Known:

* + - -

Unknown:

* + - - Scaling numerical data:
    - - Min-Max scaling (0–1 normalization)
    - - Standardization (Z-score scaling → mean=0, std=1)
    - - Robust scaling (median & IQR based)
    - - Log/square-root transformations for skewed data
    - - Encoding categorical data:
    - - One-hot encoding
    - - Label encoding
    - - Ordinal encoding
    - - Frequency/Count encoding
    - - Target/Mean encoding (advanced)
    - - Feature engineering:
    - - Creating interaction features (feature1 × feature2)
    - - Polynomial features
    - - Binning (discretization of continuous variables)
    - - Extracting date/time components (year, month, day, weekday)
    - - Text feature extraction → bag of words, TF-IDF
    - - Image feature extraction (pixels, embeddings)
    - - Domain-specific engineered features (finance ratios, medical scores, etc.)
    - - Feature selection:
    - - Filter methods (correlation, variance threshold)
    - - Wrapper methods (recursive feature elimination)
    - - Embedded methods (Lasso, tree-based importance)
    - - Handling imbalanced data:
    - - Oversampling (SMOTE)
    - - Undersampling
    - - Class weighting
    - - Practical AI/ML applications:
    - - Scaling → ensures fair contribution in distance-based models (k-NN, SVM, clustering)
    - - Encoding → converting categorical data for regression/classification
    - - Feature engineering → boosting model performance
    - - Feature selection → reducing overfitting, improving interpretability
    - - Imbalanced handling → better classification in fraud detection, healthcare

Notes:

* + - -
  + Dimensionality reduction → PCA, t-SNE

Known:

* + - -

Unknown:

* + - - Concept of dimensionality → curse of dimensionality
    - - Benefits of dimensionality reduction (faster training, less overfitting, visualization)
    - - Principal Component Analysis (PCA):
    - - Variance and covariance matrices
    - - Eigenvalues and eigenvectors
    - - Principal components → linear combinations of features
    - - Explained variance ratio
    - - Choosing number of components
    - - Whitening transformation
    - - Applications → noise reduction, preprocessing before clustering/ML
    - - t-SNE (t-Distributed Stochastic Neighbor Embedding):
    - - Concept → nonlinear dimensionality reduction for visualization
    - - Distance preservation → local neighborhoods
    - - Perplexity parameter
    - - Learning rate tuning
    - - Limitations → computationally expensive, only for visualization
    - - Comparison: PCA vs t-SNE
    - - Alternatives → UMAP (Uniform Manifold Approximation and Projection)
    - - Practical AI/ML applications:
    - - PCA → preprocessing high-dimensional datasets (images, genomics, finance)
    - - PCA → feature reduction before regression/classification
    - - t-SNE → visualizing word embeddings, clusters in NLP/CV
    - - t-SNE → anomaly detection visualization

Notes:

* + - -
  + Kaggle practice

Known:

Unknown:

* + - - Creating a Kaggle account and setting up profile
    - - Navigating competitions → types (getting started, research, featured)
    - - Downloading datasets → Kaggle datasets API
    - - Using Kaggle Kernels/Notebooks → Python, R, GPU/TPU usage
    - - Submitting predictions → CSV submission format
    - - Understanding competition metrics (accuracy, RMSE, log-loss, AUC, etc.)
    - - Using Kaggle discussion forums and notebooks (learning from others)
    - - Common starter competitions → Titanic, House Prices, Digit Recognizer
    - - Building baseline models quickly
    - - Data exploration (EDA) in Kaggle notebooks
    - - Feature engineering for Kaggle competitions
    - - Model ensembling → stacking, blending, voting
    - - Using Kaggle datasets outside competitions (for practice projects)
    - - Participating in code competitions vs data competitions
    - - Kaggle ranking system → points, medals, tiers (Novice → Grandmaster)
    - - Practical AI/ML applications:
    - - End-to-end ML pipelines on real-world data
    - - Improving model performance with experimentation
    - - Exposure to diverse problem domains (NLP, CV, tabular, time series)
    - - Collaboration with Kaggle notebooks/teams

Notes:

* + - -
* **Deprioritize for now(Information retrieval):**
  + Information Retrieval (search/indexing) — niche, not core ML foundation

Known:

* + - -

Unknown:

* + - - Concept of information retrieval (IR) → retrieving relevant documents from a large collection
    - - Document representation → Bag of Words, TF-IDF
    - - Tokenization and preprocessing → stopword removal, stemming, lemmatization
    - - Inverted index → mapping terms to documents
    - - Ranking models → BM25, vector space model
    - - Similarity measures → cosine similarity, Jaccard similarity
    - - Relevance feedback → Rocchio algorithm
    - - Precision, Recall, F1 score in IR context
    - - Query expansion techniques
    - - Web search basics → crawling, indexing, ranking
    - - Modern IR → embeddings, semantic search
    - - Vector databases → FAISS, Pinecone, Weaviate
    - - Applications in ML/AI:
    - - Search engines (Google, ElasticSearch, Solr)
    - - Recommendation systems
    - - RAG (Retrieval-Augmented Generation) for LLMs
    - - Knowledge graph search
    - - Enterprise document retrieval

Notes:

* + - - Deprioritize for now → not essential for ML/DL basics
    - - Useful later for working with LLMs, RAGs, or applied NLP search systems

**3. Core ML/DL Concepts (🔑 Medium → crucial after Python + Math)**

* Gradient Descent → batch vs stochastic vs mini-batch

Known:

* + - -

Unknown:

* + - - Concept of gradient descent → iterative optimization of loss function
    - - Update rule → θ = θ - α∇J(θ) (parameters, learning rate, gradient)
    - - Batch gradient descent → uses entire dataset per update
    - - Stochastic gradient descent (SGD) → updates per single sample
    - - Mini-batch gradient descent → updates using small random batches
    - - Trade-offs:
    - - Batch → stable but slow, requires full dataset in memory
    - - SGD → noisy updates, faster convergence, risk of instability
    - - Mini-batch → balance between efficiency & stability
    - - Learning rate selection (α) → small vs large step size
    - - Convergence criteria → when to stop training
    - - Loss surface interpretation → local minima, saddle points
    - - Challenges → vanishing/exploding gradients
    - - Improvements over vanilla GD:
    - - Momentum
    - - Nesterov Accelerated Gradient (NAG)
    - - Adaptive methods (AdaGrad, RMSProp, Adam)
    - - Practical AI/ML applications:
    - - Training linear/logistic regression
    - - Backpropagation in neural networks
    - - Large-scale deep learning (always uses mini-batches)
    - - Online learning with streaming data (SGD)

Notes:

* + - -
* Forward propagation (basic neural nets)

Known:

* + - -

Unknown:

* + - - Concept of forward propagation → passing input data through the network to compute output
    - - Structure of a simple neural network:
    - - Input layer → features
    - - Hidden layers → weights, biases, activations
    - - Output layer → predictions
    - - Weighted sum → z = w·x + b
    - - Activation functions:
    - - Sigmoid
    - - Tanh
    - - ReLU
    - - Softmax
    - - Flow of data → input → linear transformation → activation → next layer
    - - Output interpretation:
    - - Regression → raw output
    - - Classification → softmax/sigmoid probabilities
    - - Loss functions:
    - - Mean Squared Error (MSE) for regression
    - - Cross-Entropy Loss for classification
    - - Vectorized implementation (matrix multiplications instead of loops)
    - - Role of dimensions/shapes in forward pass
    - - Practical AI/ML applications:
    - - Prediction in trained neural networks
    - - Basis for backpropagation
    - - Used in inference pipelines (production models)

Notes:

* + - -
* Backpropagation (chain rule, weight updates)

Known:

* + - -

Unknown:

* + - - Concept of backpropagation → computing gradients by propagating error backward
    - - Loss function gradient → ∂L/∂ŷ (loss wrt prediction)
    - - Chain rule → breaking down gradients layer by layer
    - - Weight update rule → w := w - α \* ∂L/∂w
    - - Bias update rule → b := b - α \* ∂L/∂b
    - - Gradients of activation functions:
    - - Sigmoid derivative
    - - Tanh derivative
    - - ReLU derivative
    - - Gradient flow → output layer → hidden layers → input
    - - Vanishing gradients → why it happens (sigmoid/tanh saturation)
    - - Exploding gradients → why it happens (deep networks, large weights)
    - - Techniques to mitigate:
    - - Weight initialization strategies (Xavier, He)
    - - Gradient clipping
    - - Normalization layers (BatchNorm, LayerNorm)
    - - Vectorized implementation (matrix calculus)
    - - Practical AI/ML applications:
    - - Training deep neural networks
    - - Basis of all optimizers (SGD, Adam, RMSProp)
    - - Used in PyTorch/TensorFlow autograd
    - Notes:
    - -
* PyTorch & TensorFlow

Known:

* + - -

Unknown:

* + - - Concept of DL frameworks → automate tensor operations, autograd, GPU acceleration
    - - PyTorch basics:
    - - Tensors → torch.tensor(), operations
    - - Autograd → automatic differentiation
    - - Defining models with nn.Module
    - - Optimizers → torch.optim (SGD, Adam, etc.)
    - - Loss functions → nn.CrossEntropyLoss, nn.MSELoss
    - - Data handling → DataLoader, Dataset
    - - Training loop (forward, loss, backward, optimizer.step())
    - - TorchScript & ONNX export
    - - TensorFlow basics:
    - - Tensors → tf.constant, tf.Variable
    - - Graph execution (TensorFlow 1.x) vs eager execution (TF 2.x)
    - - Keras API → Sequential & Functional API
    - - Optimizers → SGD, Adam
    - - Loss functions → categorical\_crossentropy, mse
    - - Dataset API → tf.data pipelines
    - - Model training → model.fit(), evaluate(), predict()
    - - TensorFlow Serving, TensorFlow Lite
    - - PyTorch vs TensorFlow:
    - - PyTorch → dynamic graphs (eager execution), research-friendly
    - - TensorFlow → production-ready, better deployment tools
    - - Advanced topics:
    - - GPU/TPU usage
    - - Mixed precision training
    - - Transfer learning (pretrained models)
    - - Distributed training
    - - Practical AI/ML applications:
    - - PyTorch → widely used in academia, Hugging Face models
    - - TensorFlow → industry deployment, mobile/edge AI
    - - Both → computer vision, NLP, reinforcement learning

Notes:-

* Parameters vs Hyperparameters

Known:

* + - -

Unknown:

* + - - Concept of model parameters:
    - - Learned from data during training
    - - Examples: weights, biases in neural networks
    - - Coefficients in linear regression
    - - Support vectors in SVMs
    - - Concept of hyperparameters:
    - - Set before training, not learned from data
    - - Examples: learning rate, batch size, number of epochs
    - - Number of hidden layers/neurons
    - - Regularization strength (L1/L2)
    - - Dropout rate
    - - Parameters vs hyperparameters → key differences
    - - Hyperparameter tuning approaches:
    - - Grid search
    - - Random search
    - - Bayesian optimization
    - - Hyperband
    - - AutoML frameworks for hyperparameter tuning
    - - Practical AI/ML applications:
    - - Parameters → updated during forward/backpropagation
    - - Hyperparameters → chosen to optimize training performance
    - - Hyperparameter tuning can dramatically change model accuracy

Notes:

* + - -
* Hyperparameter tuning → grid, random, Bayesian

Known:

* + - -

Unknown:

* + - - Concept of hyperparameter tuning → finding best set of hyperparameters for model performance
    - - Grid Search:
    - - Exhaustive search across all parameter combinations
    - - Pros → simple, systematic
    - - Cons → computationally expensive, scales poorly
    - - Random Search:
    - - Samples random combinations of hyperparameters
    - - Pros → more efficient in high-dimensional spaces
    - - Cons → may miss optimal regions
    - - Bayesian Optimization:
    - - Builds a probabilistic model of the objective function
    - - Uses acquisition functions (e.g., Expected Improvement) to choose next hyperparameters
    - - Pros → more sample-efficient than grid/random
    - - Cons → more complex, requires tuning itself
    - - Practical hyperparameters to tune:
    - - Learning rate
    - - Batch size
    - - Number of epochs
    - - Number of hidden layers/neurons
    - - Regularization strength
    - - Dropout rate
    - - Tools for hyperparameter tuning:
    - - Scikit-learn → GridSearchCV, RandomizedSearchCV
    - - Optuna, Hyperopt, Ray Tune
    - - Keras/TensorFlow Tuner
    - - Practical AI/ML applications:
    - - Grid → small search space (e.g., SVM parameters)
    - - Random → large search space, quick improvement
    - - Bayesian → deep learning, expensive models
    - - AutoML frameworks combine all three approaches

Notes:

* + - -
* Regularization → L1, L2, dropout

=

Known:

* + - -

Unknown:

* + - - Concept of regularization → reducing overfitting by penalizing model complexity
    - - L1 regularization (Lasso):
    - - Penalty term = λ \* Σ |wi|
    - - Encourages sparsity (some weights → 0)
    - - Feature selection effect
    - - L2 regularization (Ridge):
    - - Penalty term = λ \* Σ wi²
    - - Shrinks weights but doesn’t set them to 0
    - - Helps with multicollinearity
    - - Elastic Net → combination of L1 + L2
    - - Choosing λ (regularization strength) → tuning required
    - - Dropout (neural networks):
    - - Randomly “drop” neurons during training
    - - Prevents co-adaptation of neurons
    - - Dropout rate as hyperparameter
    - - Other regularization techniques:
    - - Early stopping
    - - Data augmentation
    - - Batch normalization (acts like regularization)
    - - Practical AI/ML applications:
    - - L1 → sparse models, feature selection
    - - L2 → smoother, stable models
    - - Dropout → deep neural networks generalization
    - - Early stopping → avoids overfitting during long training runs

Notes:

* + - -
* Ensemble methods → bagging, boosting, stacking

Known:

* + - -

Unknown:

* + - - Concept of ensemble learning → combining multiple models to improve performance
    - - Bagging (Bootstrap Aggregating):
    - - Training models on random subsets (with replacement)
    - - Aggregating results via averaging/voting
    - - Example → Random Forest
    - - Reduces variance, helps with overfitting
    - - Boosting:
    - - Sequential training of weak learners (focus on errors of previous models)
    - - Weighted combination of learners
    - - Examples → AdaBoost, Gradient Boosting, XGBoost, LightGBM, CatBoost
    - - Reduces bias, improves accuracy
    - - Stacking:
    - - Combining predictions from multiple base models
    - - Meta-learner (stacker) learns how to best combine them
    - - Example → logistic regression stacking outputs from decision trees & SVM
    - - Differences:
    - - Bagging → parallel, reduces variance
    - - Boosting → sequential, reduces bias
    - - Stacking → meta-learning approach
    - - Practical AI/ML applications:
    - - Bagging → image classification (Random Forests on extracted features)
    - - Boosting → Kaggle competitions, tabular data (XGBoost/LightGBM dominate)
    - - Stacking → combining different ML models for best results

Notes:

* + - -
* Transfer Learning basics

Known:

* + - -

Unknown:

* + - - Concept of transfer learning → using knowledge from a pre-trained model on a new task
    - - Feature extraction → freezing base layers, training only classifier layers
    - - Fine-tuning → unfreezing some/all layers and retraining
    - - Pre-trained models in CV → VGG, ResNet, Inception, EfficientNet
    - - Pre-trained models in NLP → BERT, GPT, RoBERTa, DistilBERT
    - - Domain adaptation → transferring between related but different domains
    - - Benefits:
    - - Faster training
    - - Requires less data
    - - Better generalization
    - - When transfer learning works best → similar domains, same input modalities
    - - When it may fail → very different domains or small model capacity
    - - Practical AI/ML applications:
    - - Computer Vision → image classification with pre-trained CNNs
    - - NLP → sentiment analysis, text classification with BERT embeddings
    - - Speech → ASR with pre-trained acoustic models
    - - Multimodal → leveraging large LLMs for vision + text tasks

Notes:

* + - -
* XAI/Interpretability → SHAP, LIME, feature importance

Known:

* + - -

Unknown:

* + - - Concept of interpretability vs explainability
    - - Global vs local explanations
    - - Feature importance:
    - - Tree-based feature importance
    - - Permutation importance
    - - Partial Dependence Plots (PDP)
    - - SHAP (SHapley Additive exPlanations):
    - - Based on Shapley values from cooperative game theory
    - - Consistent, local + global explanations
    - - Pros: theoretically sound, model-agnostic
    - - Cons: computationally expensive
    - - LIME (Local Interpretable Model-agnostic Explanations):
    - - Perturbs input data to approximate local decision boundaries
    - - Pros: simple, intuitive
    - - Cons: unstable, may vary with random perturbations
    - - Counterfactual explanations → “what if” scenarios
    - - Surrogate models → approximating complex models with simpler interpretable ones
    - - Model-specific methods:
    - - Attention visualization in NLP
    - - Grad-CAM for CNNs
    - - Practical AI/ML applications:
    - - Understanding model predictions in healthcare/finance
    - - Debugging biased models
    - - Regulatory compliance (GDPR, AI Act)
    - - Improving trust in AI systems

Notes:

* + - -

**4. Traditional ML (🔑 Medium Priority)**

* Linear regression (simple, multiple)

Known:

* + - -

Unknown:

* + - - Concept of regression → predicting continuous values
    - - Simple linear regression:
    - - Equation: y = β0 + β1x + ε
    - - Interpretation of slope (β1) and intercept (β0)
    - - Assumptions → linearity, independence, homoscedasticity, normality
    - - Multiple linear regression:
    - - Equation: y = β0 + β1x1 + β2x2 + ... + βnxn + ε
    - - Interpreting coefficients with multiple predictors
    - - Multicollinearity issues
    - - Cost function → Mean Squared Error (MSE)
    - - Optimization → Gradient Descent, Normal Equation
    - - R² (coefficient of determination) → goodness of fit
    - - Adjusted R² → penalizing extra features
    - - Residual analysis → checking assumptions
    - - Regularization in regression:
    - - Lasso (L1)
    - - Ridge (L2)
    - - Elastic Net
    - - Overfitting vs underfitting in regression
    - - Practical AI/ML applications:
    - - Predicting prices (housing, stock returns)
    - - Forecasting demand
    - - Baseline model before complex ML
    - - Feature importance via coefficients

Notes:

* + - -
* Classification → logistic regression, k-NN, SVM

Known:

* + - -

Unknown:

* + - - Concept of classification → predicting discrete labels
    - - Logistic Regression:
    - - Equation → logit(p) = β0 + β1x1 + ... + βnxn
    - - Sigmoid function → mapping to probability (0–1)
    - - Decision boundary
    - - Cost function → log loss (cross-entropy)
    - - Assumptions and limitations
    - - k-Nearest Neighbors (k-NN):
    - - Concept → majority vote of nearest neighbors
    - - Distance metrics → Euclidean, Manhattan, cosine similarity
    - - Choosing k (bias-variance tradeoff)
    - - Weighted k-NN
    - - Computational complexity
    - - Support Vector Machines (SVM):
    - - Concept of maximum margin hyperplane
    - - Support vectors
    - - Hard vs soft margin
    - - Kernel trick → polynomial, RBF kernels
    - - Pros/cons → effective in high-dimensional spaces, slower on large datasets
    - - Evaluation metrics for classification:
    - - Accuracy, Precision, Recall, F1 score
    - - ROC curve, AUC
    - - Practical AI/ML applications:
    - - Logistic regression → credit scoring, binary classification
    - - k-NN → recommendation, pattern recognition
    - - SVM → text classification, image recognition, small/medium datasets

Notes:

* + - -
* Decision trees, Random Forests

Known:

* + - -

Unknown:

* + - - Decision Trees:
    - - Concept → splitting data based on feature thresholds
    - - Nodes, branches, leaves
    - - Splitting criteria:
    - - Gini impurity
    - - Entropy / Information Gain
    - - Variance reduction (for regression trees)
    - - Stopping criteria → max depth, min samples per leaf
    - - Overfitting in deep trees
    - - Pruning (pre-pruning, post-pruning)
    - - Interpreting decision paths
    - - Random Forests:
    - - Concept → ensemble of multiple decision trees (bagging)
    - - Bootstrap sampling
    - - Feature randomness → selecting subset of features for splits
    - - Aggregation → majority vote (classification), averaging (regression)
    - - Feature importance (mean decrease in impurity, permutation importance)
    - - Hyperparameters → n\_estimators, max\_depth, max\_features
    - - Advantages → robust, reduces variance, less overfitting
    - - Limitations → less interpretable than single trees
    - - Practical AI/ML applications:
    - - Decision Trees → interpretable models for healthcare, finance
    - - Random Forests → Kaggle/tabular data competitions, feature importance analysis
    - - Baseline models before gradient boosting (XGBoost, LightGBM)

Notes:

* + - -
* Clustering (k-means, DBSCAN)

Known:

* + - -

Unknown:

* + - - Concept of clustering → grouping similar data points without labels
    - - Distance metrics → Euclidean, Manhattan, cosine similarity
    - - k-Means:
    - - Objective → minimize within-cluster sum of squares
    - - Initialization → random vs k-means++
    - - Iterative process → assign clusters → update centroids → repeat
    - - Choosing k → Elbow method, Silhouette score
    - - Strengths → simple, fast
    - - Limitations → assumes spherical clusters, sensitive to outliers
    - - DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
    - - Concept → clusters based on density (core points, border points, noise)
    - - Parameters → eps (neighborhood radius), minPts (minimum neighbors)
    - - Advantages → finds arbitrarily shaped clusters, handles noise
    - - Limitations → performance drops in varying density datasets
    - - Comparison → k-Means vs DBSCAN
    - - Cluster evaluation:
    - - Internal → Silhouette score, Davies–Bouldin index
    - - External (when labels available) → Adjusted Rand Index (ARI), Mutual Information
    - - Practical AI/ML applications:
    - - Market/customer segmentation
    - - Image compression
    - - Anomaly detection (fraud, outliers)
    - - Preprocessing for other ML algorithms

Notes:

* + - -
* Recommendation systems (basic matrix factorization)

Known:

* + - -

Unknown:

* + - - Concept of recommendation systems → predicting user preferences
    - - Types of recommendation systems:
    - - Content-based filtering (similar items by features)
    - - Collaborative filtering (similar users/items by behavior)
    - - User-Item interaction matrix (ratings, clicks, purchases)
    - - Cold start problem → new users or new items
    - - Content-based filtering:
    - - TF-IDF, cosine similarity for item similarity
    - - Pros/cons (no cold start for items, limited diversity)
    - - Collaborative filtering:
    - - User-based collaborative filtering
    - - Item-based collaborative filtering
    - - k-NN for similarity
    - - Matrix Factorization:
    - - Decomposing user-item matrix into latent factors
    - - Techniques → Singular Value Decomposition (SVD), Alternating Least Squares (ALS)
    - - Pros → captures hidden relationships
    - - Cons → sparse data, scalability
    - - Evaluation metrics:
    - - RMSE, MAE for rating prediction
    - - Precision@K, Recall@K, NDCG for ranking
    - - Practical AI/ML applications:
    - - E-commerce → product recommendations (Amazon, Flipkart)
    - - Streaming → movie/music recommendations (Netflix, Spotify)
    - - Social media → friend/content suggestions
    - - Personalized learning → adaptive learning platforms

Notes:

* + - -
* Time series → ARIMA, Prophet basics

Known:

* + - -

Unknown:

* + - - Concept of time series → ordered data points over time
    - - Components of time series:
    - - Trend
    - - Seasonality
    - - Cyclic behavior
    - - Noise
    - - Stationarity → constant mean/variance over time
    - - Autocorrelation & Partial Autocorrelation (ACF, PACF)
    - - ARIMA model:
    - - AR (Auto-Regressive) component
    - - I (Integrated/differencing) for stationarity
    - - MA (Moving Average) component
    - - Parameters (p, d, q)
    - - Seasonal ARIMA (SARIMA) extension
    - - Prophet model (by Facebook/Meta):
    - - Intuitive additive model (trend + seasonality + holidays)
    - - Handling missing data and outliers
    - - Automatic changepoint detection
    - - Easy parameter tuning
    - - Model evaluation metrics:
    - - MAE, RMSE, MAPE
    - - Train-test split for time series (rolling/expanding window)
    - - Practical AI/ML applications:
    - - Stock price forecasting
    - - Demand forecasting (retail, supply chain)
    - - Energy consumption prediction
    - - Web traffic/engagement forecasting

Notes:

* + - -
* Reinforcement learning (only basics now)

Known:

* + - -

Unknown:

* + - - Concept of reinforcement learning → agent learns by interacting with environment
    - - Key components:
    - - Agent
    - - Environment
    - - State (S)
    - - Action (A)
    - - Reward (R)
    - - Policy → mapping from states to actions
    - - Value function → expected long-term reward
    - - Q-function (action-value function)
    - - Exploration vs exploitation trade-off
    - - Reward signal → immediate vs delayed rewards
    - - Markov Decision Process (MDP) basics
    - - Discount factor (γ) → future reward weighting
    - - Episodic vs continuous tasks
    - - Simple RL algorithms:
    - - Monte Carlo methods
    - - Temporal Difference (TD) learning
    - - Practical AI/ML applications:
    - - Game playing (Atari, Chess, Go)
    - - Robotics → navigation, control
    - - Recommendation systems → adaptive policies
    - - Finance → portfolio optimization

Notes:

* + - -

**5. Neural Networks & Advanced AI (🔑 Later, after core ML is solid)**

* **NN Concepts**
  + Perceptron, MLPs

Known:

* + - -

Unknown:

* + - - Perceptron:
    - - Concept → single-layer linear classifier
    - - Equation → y = step(w·x + b)
    - - Limitations → only linearly separable problems
    - - Historical context → Rosenblatt’s perceptron, XOR problem
    - - Multi-Layer Perceptrons (MLPs):
    - - Extension of perceptron with hidden layers
    - - Each neuron → weighted sum + bias + activation
    - - Universal Approximation Theorem
    - - Forward propagation in MLPs
    - - Backpropagation for training
    - - Activation functions in MLPs:
    - - Sigmoid
    - - Tanh
    - - ReLU
    - - Softmax (for classification output)
    - - Key hyperparameters:
    - - Number of hidden layers
    - - Number of neurons per layer
    - - Learning rate
    - - Batch size
    - - Regularization in MLPs → dropout, L2 weight decay
    - - Limitations of MLPs:
    - - Struggle with image and sequence data
    - - Require feature engineering
    - - Practical AI/ML applications:
    - - Tabular data classification/regression
    - - Early neural network models
    - - Basis for deeper architectures (CNNs, RNNs, Transformers)

Notes:

* + - -
  + Activation functions (Sigmoid, ReLU, Tanh, Softmax)

Known:

* + - -

Unknown:

* + - - Role of activation functions → introduce non-linearity into neural networks
    - - Sigmoid:
    - - Formula: σ(x) = 1 / (1 + e^(-x))
    - - Range: (0, 1)
    - - Used for probabilities
    - - Problems → vanishing gradients
    - - Tanh:
    - - Formula: tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x))
    - - Range: (-1, 1)
    - - Zero-centered
    - - Problems → vanishing gradients (less severe than sigmoid)
    - - ReLU (Rectified Linear Unit):
    - - Formula: f(x) = max(0, x)
    - - Pros → simple, efficient, reduces vanishing gradient problem
    - - Cons → dying ReLU (neurons stuck at 0)
    - - Variants of ReLU:
    - - Leaky ReLU
    - - Parametric ReLU (PReLU)
    - - ELU
    - - Softmax:
    - - Formula: exp(xi) / Σ exp(xj)
    - - Converts logits into probability distribution
    - - Used in classification output layer
    - - Choosing activation functions:
    - - Hidden layers → ReLU or variants
    - - Output layer:
    - - Sigmoid → binary classification
    - - Softmax → multiclass classification
    - - Linear → regression
    - - Practical AI/ML applications:
    - - Sigmoid/Tanh → early neural networks, RNNs
    - - ReLU → modern deep learning (CNNs, MLPs, Transformers)
    - - Softmax → classification models

Notes:

* + - -
  + CNNs (image tasks)

Known:

* + - -

Unknown:

* + - - Concept of CNNs → specialized neural networks for spatial data (images)
    - - Image representation → pixels as matrices (height × width × channels)
    - - Convolution operation:
    - - Filters/kernels
    - - Stride
    - - Padding (valid vs same)
    - - Feature maps
    - - Pooling layers:
    - - Max pooling
    - - Average pooling
    - - Purpose → downsampling, translation invariance
    - - Activation functions in CNNs → ReLU, variants
    - - CNN architecture basics:
    - - Convolution → Activation → Pooling → Fully Connected
    - - Flattening → converting 2D feature maps to 1D vector
    - - Regularization in CNNs:
    - - Dropout
    - - Batch Normalization
    - - Data augmentation
    - - Transfer learning in CNNs:
    - - Using pretrained models (VGG, ResNet, Inception, EfficientNet)
    - - Fine-tuning vs feature extraction
    - - Practical image tasks:
    - - Image classification (cats vs dogs, CIFAR-10, ImageNet)
    - - Object detection (YOLO, Faster R-CNN)
    - - Image segmentation (U-Net, Mask R-CNN)
    - - Face recognition
    - - Medical image analysis
    - Notes:
    - -
  + RNNs (sequence tasks)

Known:

* + - -

Unknown:

* + - - Concept of sequential data → order matters (text, time, audio)
    - - Limitation of MLPs/CNNs → can’t handle temporal dependencies directly
    - - RNN basics:
    - - Hidden state (ht)
    - - Recurrence relation → ht = f(xt, ht-1)
    - - Shared weights across time steps
    - - Types of RNNs:
    - - Vanilla RNN
    - - Bidirectional RNN
    - - Training RNNs:
    - - Backpropagation Through Time (BPTT)
    - - Vanishing/exploding gradient problem
    - - Improvements over vanilla RNN:
    - - LSTM (Long Short-Term Memory)
    - - GRU (Gated Recurrent Unit)
    - - Common activation functions in RNNs:
    - - Tanh
    - - Sigmoid
    - - Sequence-to-sequence (seq2seq) models
    - - Attention mechanism (intro, leads to Transformers)
    - - Practical applications:
    - - Natural Language Processing → sentiment analysis, machine translation
    - - Time-series forecasting
    - - Speech recognition
    - - Music generation

Notes:

* + - -
  + Graph ML (GNNs) Low priority

Known:

* + - -

Unknown:

* + - - Concept of graph data → nodes, edges, adjacency matrix
    - - Types of graphs:
    - - Directed vs undirected
    - - Weighted vs unweighted
    - - Homogeneous vs heterogeneous
    - - Graph representation learning → node embeddings
    - - Message passing framework → nodes update based on neighbors
    - - Graph Convolutional Networks (GCN):
    - - Aggregation of neighbor features
    - - Normalization with adjacency matrix
    - - Graph Attention Networks (GAT):
    - - Attention weights on neighbor nodes
    - - GraphSAGE → sampling neighbors for scalability
    - - Applications of GNNs:
    - - Node classification
    - - Link prediction
    - - Graph classification
    - - Training challenges:
    - - Over-smoothing
    - - Scalability for large graphs
    - - Practical AI/ML use cases:
    - - Social network analysis (friend recommendations, community detection)
    - - Recommendation systems
    - - Molecular property prediction (drug discovery)
    - - Knowledge graph completion
    - - Traffic and sensor networks
    - Notes:
    - -
  + LSTMs for time series

Known:

* + - -

Unknown:

* + - - Limitation of vanilla RNNs → vanishing/exploding gradients
    - - Concept of LSTM → special type of RNN with memory cells
    - - LSTM architecture:
    - - Cell state (long-term memory)
    - - Hidden state (short-term memory)
    - - Input gate
    - - Forget gate
    - - Output gate
    - - How gates regulate information flow
    - - Sequence-to-sequence modeling with LSTMs
    - - Many-to-one vs many-to-many architectures
    - - Training with Backpropagation Through Time (BPTT)
    - - Practical considerations:
    - - Handling long sequences
    - - Computational cost
    - - Variants:
    - - GRU (simpler alternative to LSTM)
    - - Bidirectional LSTMs
    - - Stacked LSTMs
    - - Practical AI/ML applications:
    - - Time series forecasting (stock prices, energy demand, weather)
    - - Anomaly detection in temporal data
    - - Speech recognition
    - - Language modeling (before Transformers)

Notes:

* + - -
  + Attention mechanisms

Known:

* + - -

Unknown:

* + - - Concept of attention → focusing on the most relevant parts of input
    - - Motivation → limitations of RNNs/LSTMs in handling long sequences
    - - Key, Query, Value (KQV) framework:
    - - Query → what we’re looking for
    - - Keys → possible matches
    - - Values → information retrieved
    - - Attention score calculation → similarity(Query, Key)
    - - Types of attention:
    - - Soft attention (differentiable, used in Transformers)
    - - Hard attention (non-differentiable, requires sampling)
    - - Self-attention (attention within the same sequence)
    - - Scaled Dot-Product Attention → (QKᵀ / √d) × V
    - - Multi-Head Attention → parallel attention over multiple subspaces
    - - Attention visualization → interpretability
    - - Comparison:
    - - RNN/LSTM → sequential processing
    - - Attention → parallelizable, handles long-range dependencies
    - - Practical AI/ML applications:
    - - Machine Translation (seq2seq with attention)
    - - Document summarization
    - - Image captioning
    - - Core of Transformer models (BERT, GPT, T5)

Notes:

* + - -
  + Transformers → encoder-decoder, self-attention
    - === Topic: Transformers (Encoder-Decoder, Self-Attention) ===

Known:

* + - -

Unknown:

* + - - Motivation → limitations of RNNs/LSTMs (sequential processing, vanishing gradients)
    - - Core architecture:
    - - Encoder → processes input sequence, outputs representations
    - - Decoder → generates output sequence step by step
    - - Self-Attention in Transformer:
    - - Each token attends to all other tokens
    - - Scaled dot-product attention → (QKᵀ / √d) × V
    - - Multi-Head Attention:
    - - Multiple attention heads → learn different relationships in parallel
    - - Positional Encoding:
    - - Adds sequence order information (since attention has no inherent order)
    - - Encoder block:
    - - Self-attention layer
    - - Feed-forward network
    - - Residual connections + LayerNorm
    - - Decoder block:
    - - Masked self-attention (prevents looking ahead)
    - - Encoder-decoder attention
    - - Feed-forward network
    - - Residual connections + LayerNorm
    - - Training objective → teacher forcing with cross-entropy loss
    - - Practical AI/ML applications:
    - - NLP → translation (original Transformer paper), text summarization, question answering
    - - LLMs → GPT, BERT, T5, LLaMA
    - - Computer Vision → Vision Transformers (ViTs)
    - - Multimodal → CLIP, Flamingo, GPT-4V

Notes:

* + - -
  + Hugging Face basics (pipelines, models)

Known:

* + - -

Unknown:

* + - - Hugging Face ecosystem → Transformers library, Datasets, Tokenizers, Hub
    - - Concept of pretrained models → downloading from Hugging Face Hub
    - - Pipelines:
    - - High-level API for common tasks
    - - Examples: sentiment-analysis, text-classification, translation, summarization, question-answering, text-generation
    - - Using a pipeline with default model
    - - Using a specific model in a pipeline
    - - Tokenization basics:
    - - Converting text → tokens → IDs
    - - WordPiece, BPE (Byte Pair Encoding), SentencePiece
    - - Models:
    - - Encoder-only (BERT, RoBERTa) → classification, embedding tasks
    - - Decoder-only (GPT, LLaMA) → text generation
    - - Encoder-decoder (T5, BART, mBART) → translation, summarization
    - - Model loading:
    - - from\_pretrained() for models & tokenizers
    - - Using GPU vs CPU for inference
    - - Hugging Face Hub:
    - - Searching models, datasets, spaces
    - - Community contributions
    - - Practical AI/ML applications:
    - - Plug-and-play NLP tasks via pipelines
    - - Fine-tuning pretrained models for custom datasets
    - - Sharing models/datasets via Hugging Face Hub
    - - Experimentation in research and production

Notes:

* + - -
* **Advanced AI**
  + LLMs (GPT, Gemini, Claude)

Known:

* + - -

Unknown:

* + - - Concept of LLMs → trained on massive text corpora with billions of parameters
    - - Transformer-based architecture
    - - Pretraining objectives:
    - - Causal language modeling (GPT-style, next word prediction)
    - - Masked language modeling (BERT-style, fill in the blank)
    - - Sequence-to-sequence modeling (T5-style, text-to-text)
    - - GPT (OpenAI):
    - - Decoder-only Transformer
    - - Autoregressive generation
    - - Variants → GPT-2, GPT-3, GPT-4
    - - Applications → text generation, coding, agents
    - - Gemini (Google DeepMind):
    - - Multimodal (text, image, audio, video understanding)
    - - Integrated with Google search and tools
    - - Focus on reasoning and factual accuracy
    - - Claude (Anthropic):
    - - Alignment-focused LLM
    - - Constitutional AI → safety via rules & principles
    - - Strengths → long context handling, safe conversations
    - - Fine-tuning methods:
    - - Instruction tuning
    - - Reinforcement Learning with Human Feedback (RLHF)
    - - Practical AI/ML applications:
    - - Chatbots, coding assistants, copilots
    - - Summarization, translation, content generation
    - - Agents → tool use, reasoning, planning
    - - Research, tutoring, knowledge work
    - - Limitations:
    - - Hallucinations (confidently wrong answers)
    - - Biases from training data
    - - Cost of training/inference
    - - Alignment/safety challenges

Notes:

* + - -
  + RAGs (retrieval-augmented generation)

Known:

* + - -

Unknown:

* + - - Concept of RAG → combining information retrieval with LLM generation
    - - Motivation → LLMs have limited knowledge cutoff and may hallucinate
    - - Workflow:
    - - User query
    - - Retriever → searches external knowledge base (documents, vector DB)
    - - Retrieved context fed into LLM
    - - Generator → LLM produces grounded answer
    - - Key components:
    - - Retriever → dense embeddings (FAISS, Pinecone, Weaviate) or sparse search (BM25)
    - - Generator → LLM (GPT, Claude, Gemini, LLaMA, etc.)
    - - Vector databases:
    - - FAISS, Pinecone, Weaviate, Milvus
    - - Embedding models → text to vector representations (sentence-transformers, OpenAI embeddings)
    - - Chunking & indexing → breaking documents into smaller retrievable pieces
    - - Prompt engineering in RAG → context + query formatting
    - - Evaluation of RAG systems:
    - - Faithfulness (grounded in source docs)
    - - Relevance (retrieved docs actually useful)
    - - Applications of RAG:
    - - Enterprise chatbots
    - - Document Q&A (legal, financial, healthcare)
    - - Customer support automation
    - - Research assistants
    - - Challenges:
    - - Retrieval quality bottleneck
    - - Latency (retriever + generator combo)
    - - Keeping knowledge base updated

Notes:

* + - -
  + Agentic AI (LangChain, LangGraph, MCP servers)

Known:

* + - -

Unknown:

* + - - Concept of Agentic AI → LLMs acting as agents that can plan, reason, and use tools
    - - Core abilities of AI agents:
    - - Tool use (APIs, databases, external apps)
    - - Multi-step reasoning
    - - Memory (short-term & long-term context)
    - - Planning & goal-oriented execution
    - - LangChain:
    - - Framework for building LLM-powered agents
    - - Chains → sequences of calls (LLMs, tools, retrievers)
    - - Agents → LLMs deciding dynamically which tools to use
    - - Memory → conversation history, vector stores
    - - Integrations → OpenAI, Hugging Face, Pinecone, FAISS, APIs
    - - LangGraph:
    - - Graph-based orchestration of LLM agents
    - - Visualizing and controlling workflows
    - - Useful for multi-agent systems
    - - MCP (Model Context Protocol) servers:
    - - Standardized protocol for LLMs to interact with tools and data
    - - Tool abstraction → consistent interface across agents
    - - Enables ecosystem interoperability
    - - Comparison:
    - - LangChain → developer framework
    - - LangGraph → orchestration + visualization
    - - MCP → interoperability layer
    - - Practical AI/ML applications:
    - - Autonomous research assistants
    - - Customer support copilots
    - - Workflow automation (emails, tickets, CRM updates)
    - - Multi-step data analysis
    - - AI agents in software testing, devops, and education

Notes:

* + - -
  + Multimodal learning (text+image+audio)
    - === Topic: Multimodal Learning (Text + Image + Audio) ===

Known:

* + - -

Unknown:

* + - - Concept of multimodal learning → combining multiple data modalities (text, image, audio, video)
    - - Why multimodal? → richer representation, closer to human perception
    - - Early fusion vs late fusion:
    - - Early → combine raw features before modeling
    - - Late → combine outputs of separate models
    - - Cross-modal learning → aligning representations between modalities
    - - Text + Image models:
    - - CLIP (Contrastive Language-Image Pretraining)
    - - BLIP, BLIP-2
    - - Vision-Language Models (VLMs)
    - - Text + Audio models:
    - - AudioLM, Speech2Text, Whisper
    - - Applications in transcription, speech translation
    - - Text + Image + Audio (true multimodal):
    - - Flamingo (DeepMind)
    - - GPT-4V (Vision-enabled GPT-4)
    - - Gemini (Google DeepMind) → text, image, audio, video
    - - Embedding alignment → mapping text, image, audio into shared latent space
    - - Challenges:
    - - Data alignment across modalities
    - - Computational cost
    - - Missing/noisy modality handling
    - - Practical AI/ML applications:
    - - Visual question answering (VQA)
    - - Image captioning
    - - Multimodal sentiment analysis
    - - Video understanding
    - - Healthcare → combining images + text reports + signals
    - - Assistive AI (speech + visual context)

Notes:

* + - -
  + Model compression → pruning, quantization, distillation

Known:

* + - -

Unknown:

* + - - Motivation → reduce model size, memory, latency while keeping accuracy
    - - Pruning:
    - - Removing unnecessary weights or neurons
    - - Structured pruning (entire neurons/layers)
    - - Unstructured pruning (individual weights)
    - - Trade-off: smaller, faster model vs possible accuracy loss
    - - Quantization:
    - - Reducing precision of weights/activations (e.g., FP32 → INT8, FP16, BF16)
    - - Post-training quantization
    - - Quantization-aware training
    - - Hardware benefits (GPUs/TPUs with INT8 support)
    - - Knowledge Distillation:
    - - Teacher-student paradigm
    - - Small “student” model learns from large “teacher” model
    - - Soft labels vs hard labels
    - - Reduces size while preserving performance
    - - Other optimization techniques:
    - - Low-rank factorization
    - - Weight sharing
    - - Neural Architecture Search (NAS) for efficient models
    - - Practical AI/ML applications:
    - - Deploying LLMs on edge devices
    - - Mobile apps (speech recognition, vision models)
    - - Real-time inference (chatbots, recommendation systems)
    - - Energy-efficient AI (green AI)

Notes:

* + - -
  + Federated & Edge AI

Known:

* + - -

Unknown:

* + - - Federated Learning (FL):
    - - Concept → training models across decentralized devices without sharing raw data
    - - Clients (devices) train locally, send model updates
    - - Aggregation on central server (e.g., FedAvg)
    - - Benefits → privacy-preserving, less data transfer
    - - Challenges → non-IID data, communication overhead, device heterogeneity
    - - Edge AI:
    - - Running AI models on edge devices (phones, IoT, embedded systems)
    - - Motivation → low latency, offline inference, privacy
    - - Deployment tools:
    - - TensorFlow Lite
    - - PyTorch Mobile
    - - ONNX Runtime
    - - Hardware accelerators → NPUs, GPUs on devices
    - - Model optimization for Edge:
    - - Quantization
    - - Pruning
    - - Knowledge distillation
    - - Combined concepts → Federated Edge AI:
    - - Local training on edge devices
    - - Central aggregation with privacy guarantees
    - - Practical AI/ML applications:
    - - Smartphones → predictive keyboards, voice assistants
    - - Healthcare → training on distributed hospital data without centralizing
    - - Finance → fraud detection on user devices
    - - IoT → smart cameras, wearables, autonomous vehicles

Notes:

* + - -
  + AI Alignment & Trustworthiness

Known:

* + - -
    - Unknown:
    - - Concept of AI alignment → making AI systems follow human values, goals, and safety constraints
    - - Alignment challenges:
    - - Goal mis-specification
    - - Distributional shift (training vs deployment environment)
    - - Unintended emergent behaviors
    - - Trustworthy AI principles:
    - - Fairness
    - - Transparency
    - - Accountability
    - - Privacy
    - - Robustness
    - - Bias in AI:
    - - Sources → training data, labeling, algorithms
    - - Types → gender, racial, cultural, confirmation bias
    - - Mitigation strategies → debiasing data, fairness constraints
    - - Interpretability for trust:
    - - Feature importance
    - - SHAP, LIME
    - - Attention visualization
    - - Robustness & safety:
    - - Adversarial robustness (defending against adversarial examples)
    - - Data poisoning detection
    - - Alignment techniques:
    - - Instruction tuning
    - - Reinforcement Learning with Human Feedback (RLHF)
    - - Constitutional AI (Anthropic → AI guided by principles)
    - - Safety layers (e.g., refusal, content moderation filters)
    - - Regulatory & ethical frameworks:
    - - EU AI Act
    - - U.S. AI Bill of Rights
    - - OECD AI principles
    - - Practical AI/ML applications:
    - - LLM safety → preventing hallucinations, toxic outputs
    - - Healthcare AI → safe decision support
    - - Finance AI → fair lending models
    - - Autonomous systems → fail-safe design

Notes:

* + - -

**6. MLOps & Professional Skills (🔑 Later / Continuous)**

* Model training, deployment, monitoring

Known:

* + - -

Unknown:

* + - - Model Training:
    - - Training loops (epochs, batches)
    - - Training/validation/test splits
    - - Early stopping
    - - Checkpointing
    - - Experiment tracking → MLflow, Weights & Biases
    - - Hyperparameter tuning integration
    - - Model Deployment:
    - - Batch inference vs real-time inference
    - - Deployment environments → cloud (AWS, GCP, Azure), edge, on-prem
    - - Containerization → Docker
    - - Model serving frameworks:
    - - TensorFlow Serving
    - - TorchServe
    - - FastAPI / Flask for custom APIs
    - - Using ONNX for cross-framework compatibility
    - - Model Monitoring:
    - - Performance drift → model accuracy degrades over time
    - - Data drift → input distribution changes
    - - Concept drift → target relationships change
    - - Monitoring tools → EvidentlyAI, Prometheus, Grafana
    - - Retraining triggers
    - - CI/CD for ML (Continuous Training/Continuous Deployment)
    - - Version control for data, code, and models
    - - Pipelines with Airflow, Kubeflow, MLflow
    - - Practical AI/ML applications:
    - - Training → fine-tuning BERT for sentiment analysis
    - - Deployment → recommendation API on AWS
    - - Monitoring → fraud detection model tracking data drift

Notes:

* + - -
* Experiment tracking (MLflow, Weights & Biases)
  + - === Topic: Experiment Tracking (MLflow, Weights & Biases) ===

Known:

* + - -

Unknown:

* + - - Motivation → ML experiments involve many runs with changing hyperparameters, models, datasets
    - - Core elements to track:
    - - Hyperparameters
    - - Metrics (accuracy, loss, F1, etc.)
    - - Artifacts (models, plots, datasets)
    - - Code version
    - - MLflow:
    - - Open-source ML lifecycle platform
    - - Components:
    - - Tracking → log params, metrics, artifacts
    - - Models → packaging ML models
    - - Registry → versioning & managing models
    - - Projects → reproducible ML code
    - - Integration with TensorFlow, PyTorch, Scikit-learn
    - - Weights & Biases (W&B):
    - - Cloud-first experiment tracking
    - - Features → live dashboards, collaboration
    - - Hyperparameter sweeps
    - - Dataset & model versioning
    - - Integration with Hugging Face, Keras, PyTorch Lightning
    - - Comparison:
    - - MLflow → flexible, self-hosted or managed
    - - W&B → team collaboration, visualization-rich, SaaS
    - - Best practices:
    - - Always log parameters & metrics automatically
    - - Version datasets
    - - Save model checkpoints
    - - Automate logging inside training scripts
    - - Practical AI/ML applications:
    - - Comparing models during Kaggle competitions
    - - Tracking experiments in research projects
    - - Production ML → monitoring experiments before deployment

Notes:

* + - -
* Pipelines (Airflow, Prefect)

Known:

* + - -

Unknown:

* + - - Concept of ML pipelines → orchestrating end-to-end workflow (data ingestion → preprocessing → training → deployment → monitoring)
    - - Why pipelines? → reproducibility, automation, scalability
    - - Core pipeline steps:
    - - Data gathering & cleaning
    - - Feature engineering
    - - Model training
    - - Evaluation
    - - Deployment
    - - Monitoring & retraining
    - - Apache Airflow:
    - - DAGs (Directed Acyclic Graphs) → define workflows
    - - Scheduling & orchestration of tasks
    - - Strong for batch workflows
    - - Integrates with cloud (AWS, GCP, Azure)
    - - Limitations → heavier setup, less Python-native
    - - Prefect:
    - - Python-native workflow orchestration
    - - Simpler, developer-friendly API
    - - Handles retries, scheduling, monitoring
    - - Cloud & local execution options
    - - Easier setup than Airflow
    - - Comparison:
    - - Airflow → mature, enterprise-grade, good for complex ETL
    - - Prefect → lightweight, modern, easier for ML workflows
    - - Best practices:
    - - Modularize pipeline tasks
    - - Log artifacts & metrics at each stage
    - - Use version control for data, models, code
    - - Practical AI/ML applications:
    - - Data pipelines (ETL) for ML
    - - Automated retraining pipelines
    - - Scheduling daily batch predictions
    - - Experiment orchestration

Notes:

* + - -
* Feature store basics

Known:

* + - -

Unknown:

* + - - Concept of a feature store → centralized system for storing, managing, and serving ML features
    - - Why feature stores?:
    - - Ensure consistency between training and inference (no feature leakage)
    - - Reuse features across models
    - - Enable real-time feature serving
    - - Core components:
    - - Feature definitions → metadata, schema
    - - Offline store → historical data for training
    - - Online store → low-latency feature serving for inference
    - - Feature transformations → standardization, scaling, encoding
    - - Popular feature stores:
    - - Feast (open-source)
    - - Tecton (enterprise, built on Feast)
    - - AWS SageMaker Feature Store
    - - Databricks Feature Store
    - - Challenges solved by feature stores:
    - - Avoiding training-serving skew
    - - Faster experimentation by reusing existing features
    - - Governance & versioning of features
    - - Integration into pipelines:
    - - ETL → populate offline store
    - - Streaming ingestion → populate online store
    - - Model pipelines → fetch features for inference
    - - Practical AI/ML applications:
    - - Real-time fraud detection (latency-sensitive inference)
    - - Recommendation systems
    - - Predictive maintenance (sensor data)
    - - Customer personalization at scale

Notes:

* + - -
* AutoML frameworks → Auto-sklearn, [H2O.ai](http://h2o.ai)

Known:

* + - -

Unknown:

* + - - Concept of AutoML → automating end-to-end ML pipeline (feature engineering → model selection → tuning)
    - - Why AutoML? → saves time, reduces manual effort, good for baselines & non-experts
    - - Auto-sklearn:
    - - Built on scikit-learn
    - - Uses Bayesian optimization + meta-learning
    - - Automated preprocessing, feature selection, algorithm selection
    - - Good for small/medium tabular datasets
    - - Integrates easily into Python ML workflows
    - - H2O.ai:
    - - Enterprise-grade AutoML platform
    - - Supports multiple algorithms (GLM, GBM, XGBoost, Deep Learning)
    - - Automated feature engineering
    - - Leaderboard with best models
    - - Distributed & scalable
    - - GUI + Python/R/Java APIs
    - - Benefits of AutoML:
    - - Fast prototyping
    - - Baseline comparisons
    - - Non-experts can build decent models
    - - Limitations of AutoML:
    - - Less control over model internals
    - - May not handle very large-scale or highly custom problems
    - - Risk of overfitting if not monitored
    - - Practical AI/ML applications:
    - - Tabular ML tasks → classification, regression
    - - Business intelligence → churn prediction, credit scoring
    - - Kaggle/competitions → baseline models
    - - Enterprise use cases → predictive maintenance, fraud detection

Notes:

* + - -
* Security (adversarial robustness, poisoning defenses)
* Governance & compliance
* **Always ongoing:**
  + Research reading (Arxiv, PapersWithCode)
  + Communication & storytelling with data