Create Environment:

* conda create -p venv python==3.10
* conda activate venv/

Create .gitignore file:

* Files like .venv don’t get uploaded in github

Create folder .github/workflows/main.yaml

* Deployement github actions

Github Setup

* echo "# Credit-Risk-Default-Prediction" >> README.md
* git init
* git add README.md ( git add .)
* git commit -m "first commit"
* git branch -M main
* git remote add origin https://github.com/reevubabbai2003/Credit-Risk-Default-Prediction.git
* git push -u origin main

Setup.py

'''

The setup.py file is an essential part of packaging and

distributing Python projects. It is used by setuptools

(or distutils in older Python versions) to define the configuration

of your project, such as its metadata, dependencies, and more

'''

from setuptools import find\_packages

* find\_packages: scan through every folder and whenever there is \_\_init\_\_.py file consider that folder as a package.

**Why We Create a setup.py File 📦**

Think of a setup.py file as the official instruction manual for your project. It tells packaging tools like pip everything they need to know to handle your code correctly.

Its main purposes are:

1. **To Provide Metadata:** It contains key information about your project, such as its **name**, **version**, **author**, and a brief description. This is what you see when you browse packages on PyPI (the Python Package Index).
2. **To List Dependencies:** The **install\_requires** argument inside setup.py lists all the other Python packages that your project needs to run. When someone installs your project, pip reads this list and automatically installs all of them.
3. **To Find Your Code:** It tells pip which parts of your project are the actual Python packages to be included. The **find\_packages()** function is a common way to do this automatically.
4. **To Make it Installable:** Most importantly, having a setup.py file allows you to bundle your project into standard formats (like a "wheel" file) that can be easily shared and installed by others using a simple pip install command.

Without it, your project is just a collection of scripts that someone would have to manually download and manage.

* **-e** stands for **"editable."**
* **.** is a shortcut for the **current directory** (where your setup.py is).

So, the command means: "Install the project in the current directory in editable mode."

**How It Works: The Shortcut Analogy**

To understand what "editable" means, think of it like creating a shortcut versus making a copy of a file.

* **Normal Install (pip install .):** This is like **making a copy** of your project's files and moving that copy into Python's site-packages directory. If you change your original source code, the installed copy remains unchanged, and you won't see your updates until you reinstall.
* **Editable Install (pip install -e .):** This is like **creating a shortcut**. Instead of copying the files, pip places a link in the site-packages directory that points directly back to your original source code.

**The Connection**

When you run pip install -e ., pip **reads your setup.py file** to understand how to create this "shortcut." It uses the metadata and dependency information from setup.py to link your project into your Python environment.

**Logging -> logger.py**

**What Code Goes Inside logger.py?**

The file typically contains code to configure a logger using Python's built-in logging module. It defines the format of the log messages, the level of detail to capture (e.g., INFO, ERROR), and where the logs should be sent.

**Why We Make This File**

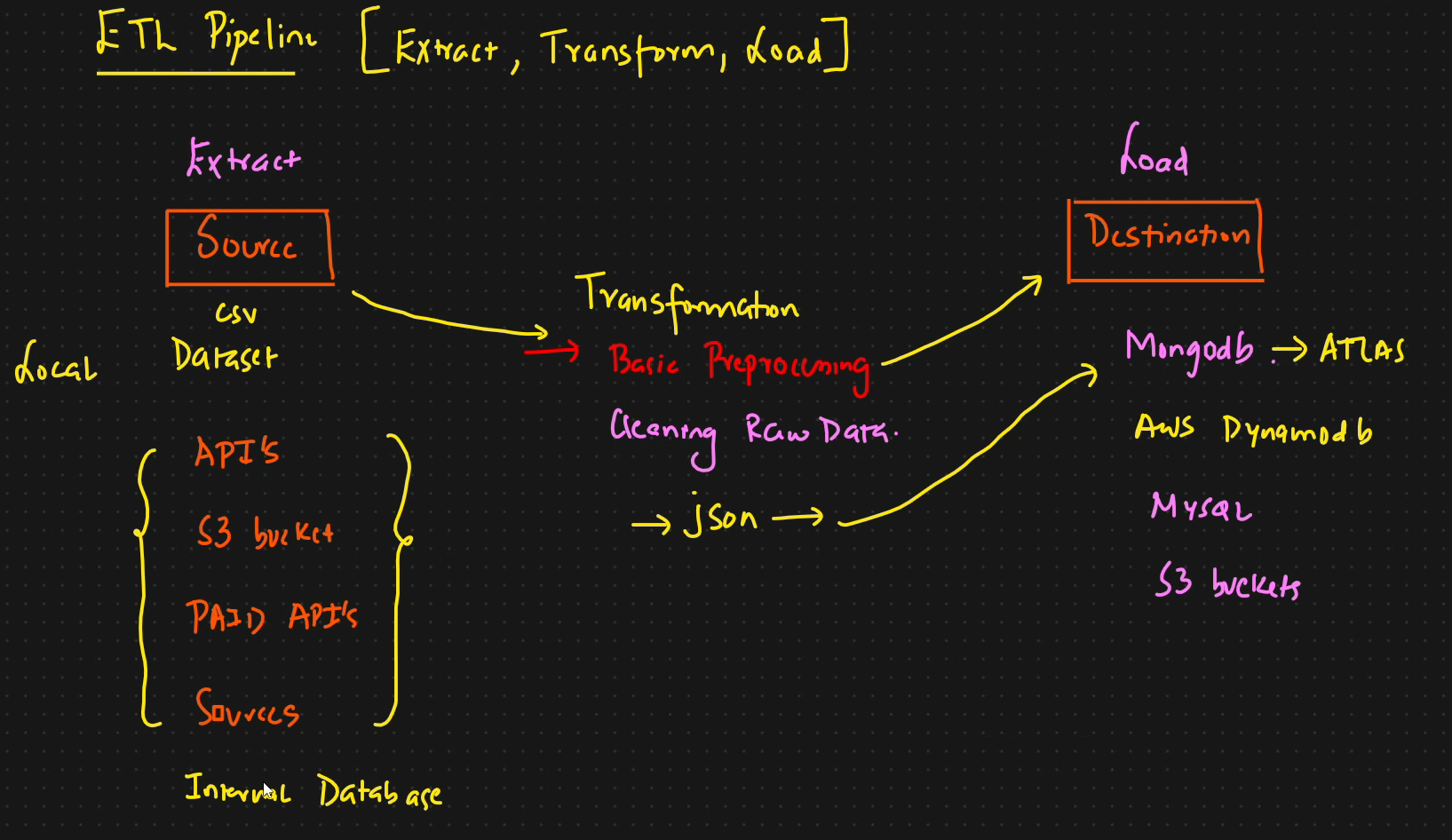
Creating a dedicated logger.py is about moving from messy, temporary debugging to a professional, maintainable system. Think of it like installing a centralized security system in a building instead of placing random webcams in different rooms.

Here are the key reasons:

1. **Centralization and Consistency**: All parts of your application can import the same logger instance. This ensures that every log message across your entire project has the **exact same format** and goes to the **same place**.
2. **Control and Flexibility**: You can change your entire project's logging behavior by editing just this one file. Want to stop logging to a file and send logs to a web service instead? Or change the log level from INFO to DEBUG to get more detail? You only have to make the change in logger.py.
3. **Separates Concerns**: It keeps the logic for *how* to log separate from the business logic of your application. Your main code just needs to say "log this message" (logging.info(...)) without worrying about timestamps, file paths, or formatting.
4. **Different Log Levels**: Logging provides severity levels (DEBUG, INFO, WARNING, ERROR, CRITICAL). This is far more powerful than print(), as you can configure your logger to show only errors in a production environment while showing detailed debug information during development.
5. **Disables print() Statements**: It's a best practice to remove all print() statements used for debugging before deploying code. A logging system replaces them, allowing you to simply change the log level to silence unimportant messages without having to find and remove every print statement.

def \_\_init\_\_(self,error\_message,error\_details:sys):

* error\_details:sys, is a **type hint** in Python.
* **sys**: This is the **expected type**. It tells anyone reading the code that the error\_details parameter is expected to be the Python sys module itself. The sys module provides access to system-specific parameters and functions, and it's commonly used in custom exception handling to get detailed information about an error that just occurred.



* Pip install –upgrade pymongo

This error means your MongoDB username or password contains a special character (like @, :, #, /, ?) that is interfering with the structure of the connection URL.

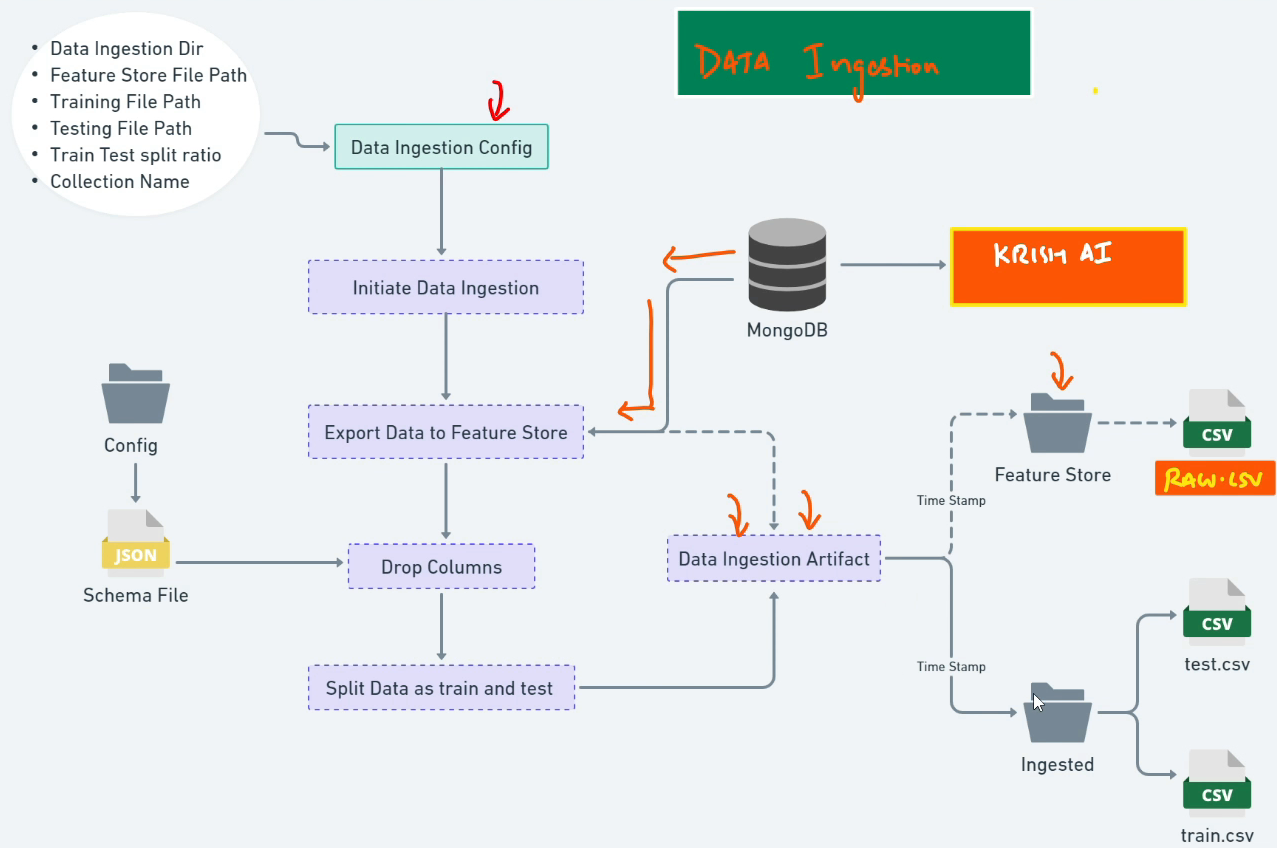
A connection URL has a specific format, and certain characters are reserved to define that format.

For example, the @ symbol is used to separate the username and password from the cluster's address. If your password itself contains an @ (e.g., my\_p@ssword123), the parser gets confused.

**The Solution ✅**

The error message itself points to the best solution: use Python's built-in **urllib.parse.quote\_plus** function to safely encode your username and password before building the connection string.

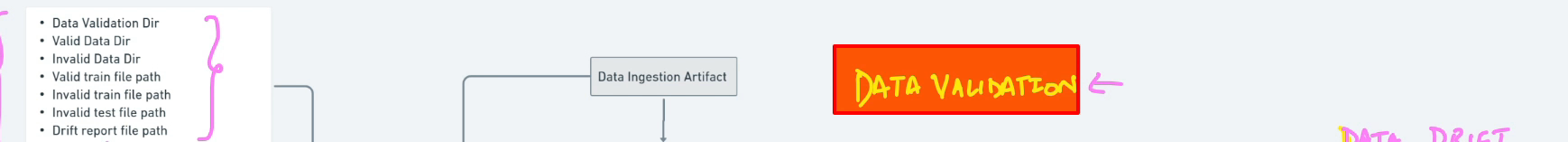
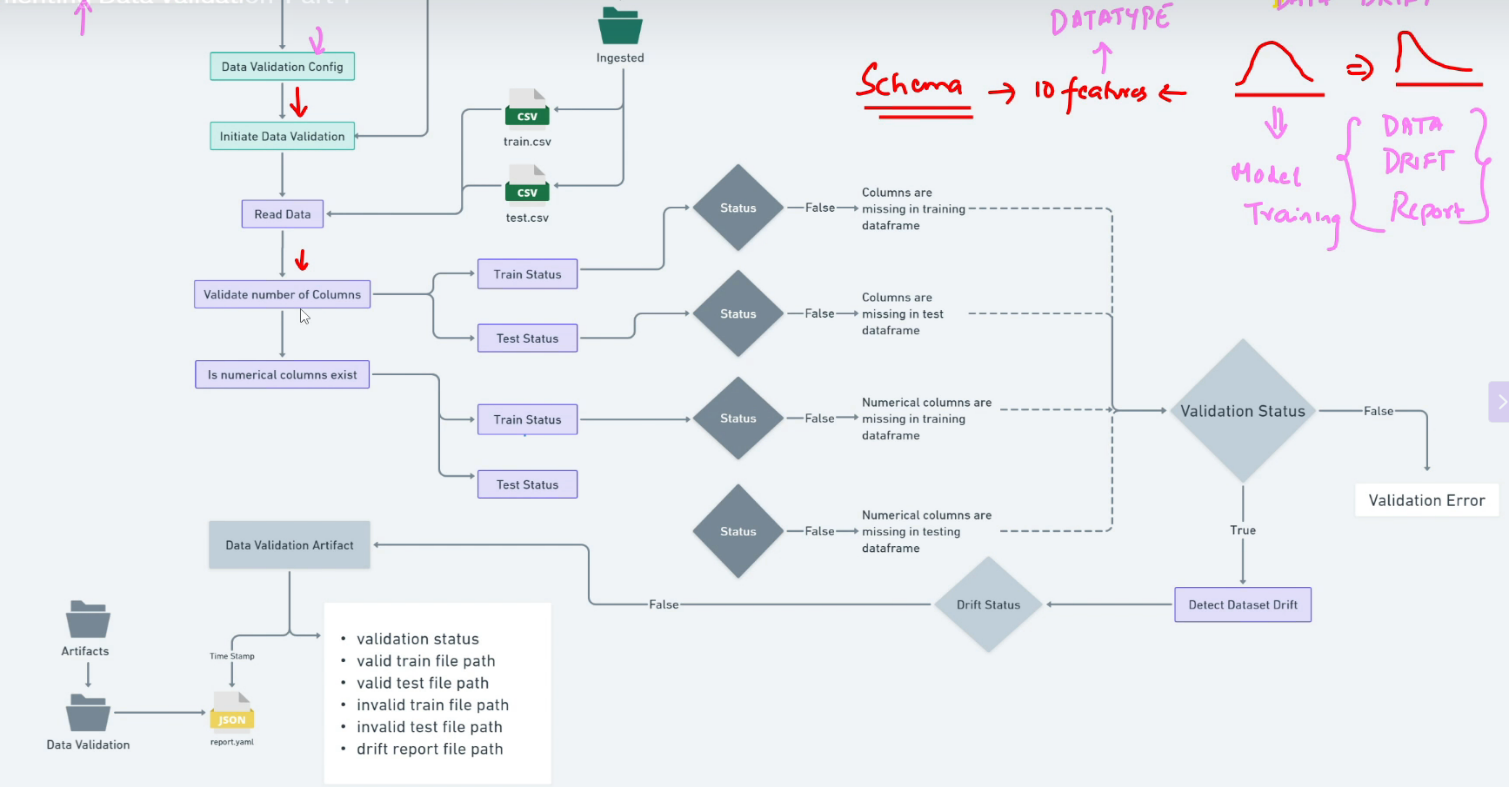
**Data Ingestion**



In the line def \_\_init\_\_(self, training\_pipeline\_config: TrainingPipelineConfig):, the role of TrainingPipelineConfig is to act as a **type hint** that specifies the required data type for the training\_pipeline\_config parameter.

In simpler terms, it's a rule that says: "To create a DataIngestionConfig object, you **must provide** an existing object that was created from the TrainingPipelineConfig class."

**DATA VALIDATION**



**What About Dropped Columns?**

You should **include all original columns** in this initial schema.yaml, even the ones you plan to drop.

The process looks like this:

1. **Ingest Data**: Load the raw dataset (e.g., from a CSV file).
2. **Validate Data**: Use your schema.yaml to confirm the raw data is structured as expected.
3. **Feature Engineering**: Perform your transformations. This is the step where you would drop the columns. Your code for this step knows which columns to drop, but the initial validation ensures they were present to begin with.