# SYNTAXES:

#### timestamp = timestamp.strftime("%m\_%d\_%Y\_%M\_H\_%S")

**Explanation:**

* **timestamp**: This variable is assumed to be a datetime object, which represents a specific date and time.
* **.strftime(format)**: This method formats the datetime object into a string based on the provided format. The format is specified using format codes.

**Format Codes Used:**

* **%m**: Month as a zero-padded decimal number (01 to 12).
* **%d**: Day of the month as a zero-padded decimal number (01 to 31).
* **%Y**: Year with century as a decimal number (e.g., 2023).
* **%M**: Minute as a zero-padded decimal number (00 to 59).
* **%H**: Hour (24-hour clock) as a zero-padded decimal number (00 to 23).
* **%S**: Second as a zero-padded decimal number (00 to 59).

**Result:**

After executing this line, the timestamp variable will contain a string formatted like "01\_19\_2025\_00\_H\_00" (if the timestamp represents January 19, 2025, at midnight

#### self.data\_ingestion\_dir: str = os.path.join(

#### training\_pipeline\_config.artifacts\_dir, training\_pipeline.DATA\_INGESTION\_DIR\_NAME

)

**Explanation:**

* **self.data\_ingestion\_dir: str**: This indicates that data\_ingestion\_dir is an instance variable of a class and is annotated to be of type str (string).
* **os.path.join(path1, path2)**: This function is used to join one or more path components intelligently. It concatenates the given paths using the appropriate separator for the operating system (e.g., \ for Windows and / for Unix/Linux).

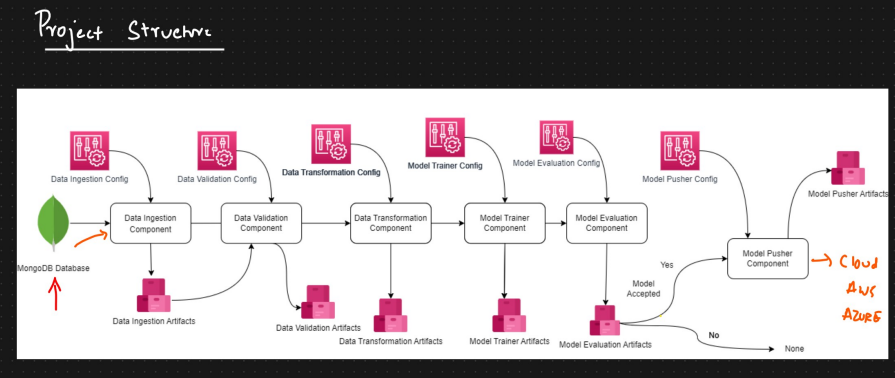
**Components:**

* **training\_pipeline\_config.artifacts\_dir**: This represents the directory path where artifacts are stored, which is presumably an attribute of the training\_pipeline\_config object.
* **training\_pipeline.DATA\_INGESTION\_DIR\_NAME**: This likely refers to a constant or attribute that contains the name of the directory where data ingestion takes place.

**Result:**

After executing this line, self.data\_ingestion\_dir will contain the full path to the data ingestion directory, constructed by joining the artifacts\_dir and DATA\_INGESTION\_DIR\_NAME. This is useful for organizing and accessing files in a structured way.

# PROJECT STRUCTURE:



This diagram represents a **machine learning project pipeline** with modular components, each performing a specific task. Below is an explanation of each part of the pipeline, its role, and how it connects to real-life scenarios.

**Components Overview**

1. **Data Ingestion**
   * **Purpose**: Collect raw data and prepare it for analysis.
   * **Input**: Raw data from a **MongoDB database** (can also be CSV files, APIs, etc.).
   * **Output**: Stored data in a standardized format (artifacts for further processing).
   * **Real-Life Example**:  
     Think of a food delivery app where orders, customer details, and delivery times are stored in a database. This component extracts that data for analysis, like predicting delivery times.
2. **Data Validation**
   * **Purpose**: Ensure the data is clean and usable for training.
   * **Tasks**:
     + Identify missing, incorrect, or inconsistent data.
     + Separate valid and invalid data into respective directories.
     + Perform drift detection to check if new data differs significantly from training data.
   * **Output**: Validated data artifacts.
   * **Real-Life Example**:  
     Imagine analyzing customer reviews. If the dataset includes duplicate reviews or reviews in unsupported languages, this component will flag or remove such data.
3. **Data Transformation**
   * **Purpose**: Convert data into a format suitable for model training.
   * **Tasks**:
     + Encode categorical features (e.g., turning "Yes/No" into 1/0).
     + Normalize or scale numerical values.
     + Handle missing data.
   * **Output**: Transformed data stored in artifacts.
   * **Real-Life Example**:  
     In an online retail store, customer purchase data may include categories like "Product Type." These are converted into numerical values for machine learning models to process.
4. **Model Trainer**
   * **Purpose**: Train the machine learning model using the transformed data.
   * **Tasks**:
     + Split data into training and validation sets.
     + Train the model (e.g., Random Forest, Neural Network).
     + Tune hyperparameters to improve performance.
   * **Output**: A trained model artifact.
   * **Real-Life Example**:  
     A fraud detection system trains a model using historical transaction data to predict whether a new transaction is fraudulent.
5. **Model Evaluation**
   * **Purpose**: Evaluate the trained model's performance.
   * **Tasks**:
     + Test the model using unseen (test) data.
     + Calculate metrics like accuracy, precision, recall, or F1 score.
     + Check if the model meets the expected performance threshold.
   * **Output**: A decision whether to accept or reject the model.
   * **Real-Life Example**:  
     In a spam detection system, the model is tested on new emails to see how accurately it identifies spam.
6. **Model Pusher**
   * **Purpose**: Deploy the model to a production environment.
   * **Tasks**:
     + Save the model in cloud storage (AWS, Azure, etc.).
     + Make it accessible to an application (e.g., via APIs).
   * **Output**: A deployed model artifact ready for real-world use.
   * **Real-Life Example**:  
     A voice assistant (e.g., Alexa or Siri) deploys its updated speech recognition model to handle user queries better.

**Workflow Explanation**

1. **Input (Raw Data)**:  
   Data starts in the **MongoDB database**. For example, you might have user data (age, location, purchases) stored in MongoDB.
   * **Data Ingestion**: Extracts and organizes the data for processing.
   * **Data Validation**: Ensures the extracted data is complete, correct, and relevant.
2. **Processing (Preparing Data)**:
   * **Data Transformation**: Converts data into a machine-readable format. Example: Convert text labels ("cat", "dog") into numbers (0, 1).
3. **Training and Testing**:
   * **Model Trainer**: Uses the prepared data to build a model (e.g., predicting whether a user will click an ad).
   * **Model Evaluation**: Tests the model with unseen data to ensure it works well.
4. **Deployment**:
   * **Model Pusher**: Deploys the final model for use in production, such as a recommendation system on an e-commerce website.

**Key Benefits**

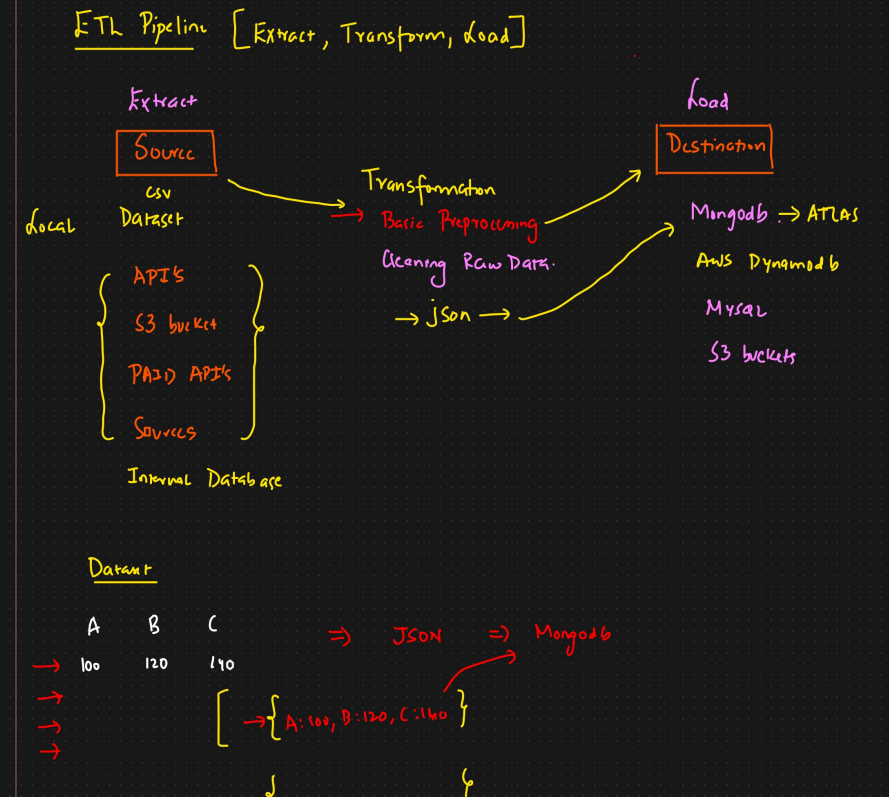
1. **Modularity**: Each stage is independent, making it easier to debug and improve specific components.
2. **Reusability**: Components like data ingestion or transformation can be reused for similar projects.
3. **Scalability**: With cloud integration (AWS, Azure), the system can handle large datasets and complex models.
4. **Automation**: Reduces manual work by automating data preparation, training, and deployment.

**Relatable Example**

**Problem**: A food delivery company wants to predict delivery times.

* **Data Ingestion**: Extract data like order time, restaurant location, and customer location from MongoDB.
* **Data Validation**: Remove incomplete orders (e.g., missing restaurant details).
* **Data Transformation**: Convert categorical features like restaurant types ("Fast Food", "Fine Dining") into numbers.
* **Model Trainer**: Train a model to predict delivery times using features like distance, order size, and time of day.
* **Model Evaluation**: Test the model to ensure it predicts delivery times accurately.
* **Model Pusher**: Deploy the model to a cloud server so the app can use it to show delivery time estimates to users.

This pipeline structure ensures a systematic, efficient, and reliable process for developing machine learning solutions, from raw data to deployment.



**1. Extract (Data Collection)**

This step focuses on gathering data from various sources.

* **Sources**:
  + **Local Files**: Data stored as CSV files on your system.
  + **APIs**: Data accessed from services like weather APIs, finance APIs, etc.
  + **S3 Buckets**: Data stored in AWS cloud storage.
  + **Internal Databases**: Company-maintained databases (e.g., MongoDB, MySQL).
  + **Paid APIs**: Third-party APIs that provide specific datasets for a fee.

**Real-Life Example:**

Imagine you're managing a **food delivery service**:

* Extract order data from an internal MySQL database.
* Fetch weather data from a weather API.
* Download customer feedback from CSV files exported by customer service tools.

**2. Transformation (Data Cleaning & Formatting)**

Once the data is extracted, it must be transformed into a usable format.

* **Steps Involved**:
  + **Basic Preprocessing**: Remove duplicates, fill missing values, and standardize formats (e.g., dates).
  + **Cleaning Raw Data**: Address issues like typos, invalid values, or outliers.
  + **Convert to JSON**: Transform the data into a JSON structure for compatibility with other systems.

**Real-Life Example:**

In the food delivery service:

* Transform numerical order IDs into strings for consistency.
* Convert raw data (e.g., customer age) into standardized bins like "18-25" or "26-40".
* Format order details like {"Customer": "John", "OrderID": 101} to JSON for storage.

**3. Load (Data Storage)**

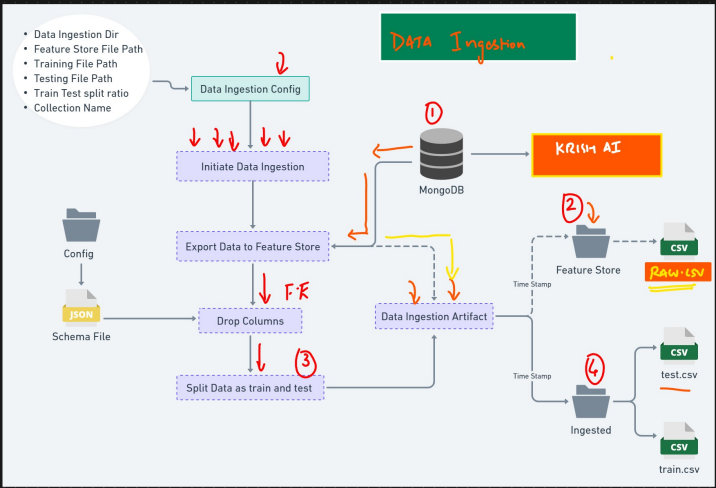
The cleaned and transformed data is stored in a **destination system** for analysis, modeling, or reporting.

* **Destination Options**:
  + **MongoDB (Atlas)**: A NoSQL database to store data in JSON-like format.
  + **AWS DynamoDB**: A scalable database for high-performance applications.
  + **MySQL**: A relational database for structured data storage.
  + **S3 Buckets**: Cloud storage for raw or processed data files.

**Real-Life Example:**

* Save customer order details into **MongoDB** for quick search and retrieval.
* Store historical weather data into **AWS DynamoDB** for predictive analysis.
* Keep backup CSVs or processed data in **S3 Buckets** for disaster recovery.

# DATA INGESTION:

****

In the context of **Machine Learning (ML)**, data ingestion involves collecting, importing, and preparing data for building, training, and evaluating ML models. Here’s a tailored breakdown of the process:

**1. Define ML Objectives**

* **Problem Definition**: Clearly define the ML problem (e.g., regression, classification, clustering).
* **Data Requirements**: Identify the type, volume, and format of the data needed for the ML task.
* **Target Variable**: Specify the dependent variable for supervised learning.

**2. Identify Data Sources**

* **Data Sources for ML**:
  + **Structured Data**: Databases, spreadsheets (e.g., sales records, financial data).
  + **Unstructured Data**: Text, images, videos (e.g., social media posts, medical images).
  + **Streaming Data**: Sensor feeds, logs, real-time events.
  + **External APIs**: Public datasets or external APIs (e.g., weather data, stock prices).

**3. Choose Ingestion Method Based on ML Use Case**

| **Mode** | **Description** | **Example ML Use Case** | **Tools** |
| --- | --- | --- | --- |
| **Batch Ingestion** | Load data periodically. | Offline training for demand forecasting. | Apache Nifi, Talend, AWS Glue |
| **Streaming Ingestion** | Ingest data in real-time as it arrives. | Real-time fraud detection or anomaly detection. | Apache Kafka, Spark Streaming |

**4. Data Extraction**

* **Connect to Data Sources**: Extract data using database queries, API calls, or file uploads.
* **Authentication**: Securely connect using credentials or API tokens.
* **Incremental Data Loading**: Avoid reloading entire datasets by fetching only new or updated records.

**5. Data Preprocessing (Critical for ML)**

Once data is ingested, preprocessing is essential to prepare it for ML models:

* **Data Cleaning**:
  + Handle missing values (imputation or removal).
  + Remove duplicates and outliers.
* **Data Transformation**:
  + Normalize/scale numerical features (e.g., StandardScaler, MinMaxScaler).
  + Encode categorical variables (e.g., One-Hot Encoding, Label Encoding).
  + Extract meaningful features from complex data (e.g., text tokenization, image resizing).
* **Time-Series Data**:
  + Handle datetime fields (extract day, month, hour, etc.).
  + Resample for consistent intervals.

**6. Data Splitting**

* Split the ingested data into:
  + **Training Set**: For training the model (e.g., 70-80%).
  + **Validation Set**: For hyperparameter tuning (e.g., 10-20%).
  + **Test Set**: For final model evaluation (e.g., 10-20%).

**7. Data Quality Checks**

* **Completeness**: Ensure all required features are present.
* **Relevance**: Confirm the data aligns with the ML objective.
* **Consistency**: Check for uniform units, formats, and values.
* **Distribution Check**: Analyze feature distributions to detect imbalances (e.g., class imbalance in classification).

**8. Data Storage**

* Use appropriate storage solutions for ML workflows:
  + **Local Storage**: For small projects or prototypes.
  + **Cloud Storage**: For scalable and collaborative ML projects (e.g., AWS S3, Azure Blob, GCP Storage).
  + **Data Lake**: For raw, unprocessed data.
  + **Feature Store**: Specialized storage for ML features (e.g., Tecton, Feast).

**9. Automating Data Ingestion Pipelines**

Automate the ingestion process for consistency and scalability:

* **ETL Pipelines**: Use tools to extract, transform, and load data.
  + Tools: Apache Airflow, AWS Glue, Databricks.
* **Monitoring**: Track the pipeline for failures and latency.
  + Tools: Prometheus, Grafana.

**10. Challenges in ML Data Ingestion**

| **Challenge** | **Solution** |
| --- | --- |
| Handling Missing Data | Imputation techniques or feature engineering. |
| Dealing with Large Data Volumes | Use distributed systems like Spark or Hadoop. |
| Real-Time Data Processing | Use streaming tools like Apache Kafka or Spark Streaming. |
| Data Imbalance in ML | Use techniques like SMOTE, undersampling, or oversampling. |

**Example Workflow for ML Data Ingestion**

1. **Extract Data**: Load sales data from a relational database using SQL.
2. **Transform Data**:
   * Clean missing values.
   * Scale features like "Price" and "Quantity."
   * One-hot encode "Category."
3. **Store Transformed Data**: Save the processed dataset in CSV format for training.
4. **Split Data**: Divide into training, validation, and test sets.
5. **Automate Pipeline**: Schedule daily updates to include new data.

**Key Tools for ML Data Ingestion**

| **Tool** | **Purpose** | **ML Use Case** |
| --- | --- | --- |
| Apache Kafka | Real-time data streaming | Real-time predictions. |
| Apache Airflow | Workflow orchestration | ETL for training ML models. |
| Google BigQuery | Data warehousing | Large-scale data analysis. |
| AWS Glue | Serverless ETL | Preprocessing for ML pipelines. |

By ingesting clean, well-prepared data, the foundation is set for building effective ML models.

# Creating training pipeline file:

import os

import sys

import numpy as np

import pandas as pd

"""

defining common constant variable for training pipeline

"""

TARGET\_COLUMN = "Result"

PIPELINE\_NAME: str= "NetworkSecurity"

ARTIFACT\_DIR: str = "Artifacts"

FILE\_NAME  : str = "phisingData.csv"

TRAIN\_FILE\_NAME : str = "train.csv"

TEST\_FILE\_NAME: str = "test.csv"

"""

data ingestion related constants starts with DATA\_INGESTION VAR NAME

"""

DATA\_INGESTION\_COLLECTION\_NAME : str = "NetworkData"

DATA\_INGESTION\_DATABASE\_NAME : str = "VAISHU"

DATA\_INGESTION\_DIR\_NAME : str = "data\_ingestion"

DATA\_INGESTION\_FEATURE\_STORE\_DIR : str = "feature\_store"

DATA\_INGESTION\_INGESTED\_DIR : str = "ingested"

DATA\_INGESTION\_TRAIN\_TEST\_SPLIT\_RATION: float = 0.2

**1. Purpose of Defining Constants**

Constants make the code more readable, maintainable, and scalable. In this case, they are specific to the data ingestion process and serve as centralized values that control key parameters and directories.

**General Constants for the Training Pipeline**

**a. TARGET\_COLUMN**

* **Purpose**: Specifies the target variable (dependent variable) that the model is trained to predict, e.g., "Result."
* **Need**:
  + Essential for supervised learning tasks.
  + Ensures consistent reference to the column throughout the pipeline.
* **Benefit**:
  + Avoids hardcoding the target column name multiple times, reducing errors.

**b. PIPELINE\_NAME**

* **Purpose**: Provides a name for the overall ML pipeline, e.g., "NetworkSecurity."
* **Need**:
  + Useful for logging, organizing files, and monitoring purposes.
* **Benefit**:
  + Adds clarity to logs and artifacts when working on multiple pipelines.

**c. ARTIFACT\_DIR**

* **Purpose**: Central directory where all outputs (artifacts) from the pipeline (models, logs, processed data) are stored.
* **Need**:
  + Necessary for organizing outputs, especially for large projects.
* **Benefit**:
  + Simplifies debugging and result interpretation.
  + Keeps all pipeline-related outputs centralized.

**d. FILE\_NAME**

* **Purpose**: The name of the primary dataset used for training, e.g., phisingData.csv.
* **Need**:
  + Allows clear identification of the raw dataset.
* **Benefit**:
  + Makes it easy to update or replace datasets without modifying the code.

**e. TRAIN\_FILE\_NAME & TEST\_FILE\_NAME**

* **Purpose**: Define the filenames for training and testing datasets, e.g., train.csv and test.csv.
* **Need**:
  + Required for saving and accessing datasets after the train-test split.
* **Benefit**:
  + Ensures consistent naming conventions.
  + Facilitates debugging and reproducibility.

**2. Data Ingestion Constants**

**a. DATA\_INGESTION\_COLLECTION\_NAME & DATA\_INGESTION\_DATABASE\_NAME**

* **Purpose**: Define the database (VAISHU) and collection (NetworkData) for fetching data.
* **Need**:
  + Centralizes database connection details for consistent data access.
* **Benefit**:
  + Makes it easier to switch between different databases or collections without modifying multiple code sections.

**b. DATA\_INGESTION\_DIR\_NAME**

* **Purpose**: Directory name for data ingestion outputs.
* **Need**:
  + Necessary for organizing intermediate outputs during the data ingestion process.
* **Benefit**:
  + Keeps raw and processed data organized for easy debugging and tracking.

**c. DATA\_INGESTION\_FEATURE\_STORE\_DIR**

* **Purpose**: Directory for saving feature-engineered data.
* **Need**:
  + Required for storing reusable and preprocessed features.
* **Benefit**:
  + Saves computation time by reusing preprocessed features across experiments or pipelines.

**d. DATA\_INGESTION\_INGESTED\_DIR**

* **Purpose**: Directory for saving ingested (processed) data.
* **Need**:
  + Separates raw data from processed data, ensuring a clear data flow.
* **Benefit**:
  + Simplifies debugging and ensures clarity in data lineage.

**e. DATA\_INGESTION\_TRAIN\_TEST\_SPLIT\_RATIO**

* **Purpose**: Specifies the ratio for splitting data into training and testing sets (e.g., 80-20 split).
* **Need**:
  + Required to ensure consistent and reproducible train-test splits.
* **Benefit**:
  + Centralized control for adjusting the split ratio.
  + Facilitates experimentation by easily changing split proportions.

**Benefits of Using Constants in ML Pipelines**

1. **Code Reusability**:
   * Constants can be reused across the pipeline, avoiding duplication.
2. **Ease of Maintenance**:
   * Centralized values make updates seamless (e.g., changing file names or directories).
3. **Readability**:
   * Well-named constants improve the readability of the code.
4. **Error Prevention**:
   * Reduces hardcoding and potential errors from inconsistent references.
5. **Scalability**:
   * Simplifies adapting the pipeline for new datasets or projects.

**Example Workflow**

1. Use DATA\_INGESTION\_DATABASE\_NAME and DATA\_INGESTION\_COLLECTION\_NAME to fetch data.
2. Save raw data to ARTIFACT\_DIR under the directory defined by DATA\_INGESTION\_DIR\_NAME.
3. Process and save feature-engineered data in DATA\_INGESTION\_FEATURE\_STORE\_DIR.
4. Perform train-test split based on DATA\_INGESTION\_TRAIN\_TEST\_SPLIT\_RATIO, saving files as TRAIN\_FILE\_NAME and TEST\_FILE\_NAME.
5. Train models using the TARGET\_COLUMN as the dependent variable.

This modular structure ensures your ML pipeline remains flexible, organized, and maintainable.

# Creating entity\_config file:

**Why Create config\_entity.py?**

**1. Centralized Configuration Management**

* Stores all settings like file paths, database names, and hyperparameters in one place.
* Avoids hardcoding values across multiple scripts.

**2. Scalability**

* Simplifies handling more parameters as the project grows.
* Supports dynamic configurations for different environments (development, testing, production).

**3. Experimentation Made Easy**

* Allows quick changes (e.g., train-test split ratio, model hyperparameters) without modifying core code.
* Facilitates running multiple experiments efficiently.

**4. Reproducibility**

* Ensures all configurations are consistent, making experiments and results easier to replicate.

**5. Improved Maintainability**

* Modifications to parameters can be made in one place without affecting the rest of the codebase.

from datetime import datetime

import os

from networksecurity.constants import training\_pipeline

print(training\_pipeline.PIPELINE\_NAME)

print(training\_pipeline.ARTIFACT\_DIR)

class TrainingPipelineConfig:

    def \_\_init\_\_(self,timestamp=datetime.now()):

        timestamp=timestamp.strftime("%m\_%d\_%Y\_%H\_%M\_%S")

        self.pipeline\_name=training\_pipeline.PIPELINE\_NAME

        self.artifact\_name=training\_pipeline.ARTIFACT\_DIR

        self.artifact\_dir=os.path.join(self.artifact\_name,timestamp)

        self.model\_dir=os.path.join("final\_model")

        self.timestamp: str=timestamp

class DataIngestionConfig:

    def \_\_init\_\_(self,training\_pipeline\_config:TrainingPipelineConfig):

        self.data\_ingestion\_dir:str=os.path.join(

            training\_pipeline\_config.artifact\_dir,training\_pipeline.DATA\_INGESTION\_DIR\_NAME

        )

        self.feature\_store\_file\_path: str = os.path.join(

                self.data\_ingestion\_dir, training\_pipeline.DATA\_INGESTION\_FEATURE\_STORE\_DIR, training\_pipeline.FILE\_NAME

            )

        self.training\_file\_path: str = os.path.join(

                self.data\_ingestion\_dir, training\_pipeline.DATA\_INGESTION\_INGESTED\_DIR, training\_pipeline.TRAIN\_FILE\_NAME

            )

        self.testing\_file\_path: str = os.path.join(

                self.data\_ingestion\_dir, training\_pipeline.DATA\_INGESTION\_INGESTED\_DIR, training\_pipeline.TEST\_FILE\_NAME

            )

        self.train\_test\_split\_ratio: float = training\_pipeline.DATA\_INGESTION\_TRAIN\_TEST\_SPLIT\_RATION

        self.collection\_name: str = training\_pipeline.DATA\_INGESTION\_COLLECTION\_NAME

        self.database\_name: str = training\_pipeline.DATA\_INGESTION\_DATABASE\_NAME

class DataValidationConfig:

    def \_\_init\_\_(self,training\_pipeline\_config:TrainingPipelineConfig):

        self.data\_validation\_dir: str = os.path.join( training\_pipeline\_config.artifact\_dir, training\_pipeline.DATA\_VALIDATION\_DIR\_NAME)

        self.valid\_data\_dir: str = os.path.join(self.data\_validation\_dir, training\_pipeline.DATA\_VALIDATION\_VALID\_DIR)

        self.invalid\_data\_dir: str = os.path.join(self.data\_validation\_dir, training\_pipeline.DATA\_VALIDATION\_INVALID\_DIR)

        self.valid\_train\_file\_path: str = os.path.join(self.valid\_data\_dir, training\_pipeline.TRAIN\_FILE\_NAME)

        self.valid\_test\_file\_path: str = os.path.join(self.valid\_data\_dir, training\_pipeline.TEST\_FILE\_NAME)

        self.invalid\_train\_file\_path: str = os.path.join(self.invalid\_data\_dir, training\_pipeline.TRAIN\_FILE\_NAME)

        self.invalid\_test\_file\_path: str = os.path.join(self.invalid\_data\_dir, training\_pipeline.TEST\_FILE\_NAME)

        self.drift\_report\_file\_path: str = os.path.join(

            self.data\_validation\_dir,

            training\_pipeline.DATA\_VALIDATION\_DRIFT\_REPORT\_DIR,

            training\_pipeline.DATA\_VALIDATION\_DRIFT\_REPORT\_FILE\_NAME,

        )

class DataTransformationConfig:

     def \_\_init\_\_(self,training\_pipeline\_config:TrainingPipelineConfig):

        self.data\_transformation\_dir: str = os.path.join( training\_pipeline\_config.artifact\_dir,training\_pipeline.DATA\_TRANSFORMATION\_DIR\_NAME )

        self.transformed\_train\_file\_path: str = os.path.join( self.data\_transformation\_dir,training\_pipeline.DATA\_TRANSFORMATION\_TRANSFORMED\_DATA\_DIR,

            training\_pipeline.TRAIN\_FILE\_NAME.replace("csv", "npy"),)

        self.transformed\_test\_file\_path: str = os.path.join(self.data\_transformation\_dir,  training\_pipeline.DATA\_TRANSFORMATION\_TRANSFORMED\_DATA\_DIR,

            training\_pipeline.TEST\_FILE\_NAME.replace("csv", "npy"), )

        self.transformed\_object\_file\_path: str = os.path.join( self.data\_transformation\_dir, training\_pipeline.DATA\_TRANSFORMATION\_TRANSFORMED\_OBJECT\_DIR,

            training\_pipeline.PREPROCESSING\_OBJECT\_FILE\_NAME,)

class ModelTrainerConfig:

    def \_\_init\_\_(self,training\_pipeline\_config:TrainingPipelineConfig):

        self.model\_trainer\_dir: str = os.path.join(

            training\_pipeline\_config.artifact\_dir, training\_pipeline.MODEL\_TRAINER\_DIR\_NAME

        )

        self.trained\_model\_file\_path: str = os.path.join(

            self.model\_trainer\_dir, training\_pipeline.MODEL\_TRAINER\_TRAINED\_MODEL\_DIR,

            training\_pipeline.MODEL\_FILE\_NAME

        )

        self.expected\_accuracy: float = training\_pipeline.MODEL\_TRAINER\_EXPECTED\_SCORE

        self.overfitting\_underfitting\_threshold = training\_pipeline.MODEL\_TRAINER\_OVER\_FIITING\_UNDER\_FITTING\_THRESHOLD

**Purpose of Each Class**

1. **TrainingPipelineConfig**:
   * Handles configurations for the entire training pipeline.
   * Creates a unique directory for each pipeline run using a timestamp.
   * Defines paths for final model storage.
2. **DataIngestionConfig**:
   * Manages file paths and parameters for the data ingestion stage.
   * Includes paths for raw data, training/testing datasets, and feature store.
   * Includes database and collection names for retrieving data.
3. **DataValidationConfig**:
   * Manages configurations for validating the ingested data.
   * Defines paths for valid/invalid datasets and drift report files.
   * Helps ensure the data quality is suitable for the pipeline.
4. **DataTransformationConfig**:
   * Specifies paths for storing transformed datasets and preprocessing objects.
   * Converts raw CSV files into .npy format for optimized storage and processing.
5. **ModelTrainerConfig**:
   * Handles configurations for the model training phase.
   * Includes paths for saving trained models.
   * Stores thresholds for expected accuracy and overfitting/underfitting detection.

**Why Create These Configuration Classes?**

**1. Modularity**

* Each class focuses on a specific pipeline stage, isolating its configurations from others.
* Enhances code readability and maintainability.

**2. Dynamic Path Management**

* Generates unique directories for every run, avoiding file overwrites.
* Automates file and directory creation, saving manual effort.

**3. Centralized Configuration**

* Avoids scattering constants and paths across multiple scripts.
* Updates to constants in the training\_pipeline module automatically reflect in all stages.

**4. Scalability**

* Simplifies adding new stages or modifying existing ones without disrupting the entire pipeline.

**5. Error Reduction**

* Clearly defined paths reduce the chances of misconfigurations.
* Automating directory and file creation prevents manual errors.

**6. Reproducibility**

* Tracks all paths and parameters in a structured way, facilitating experiment replication.