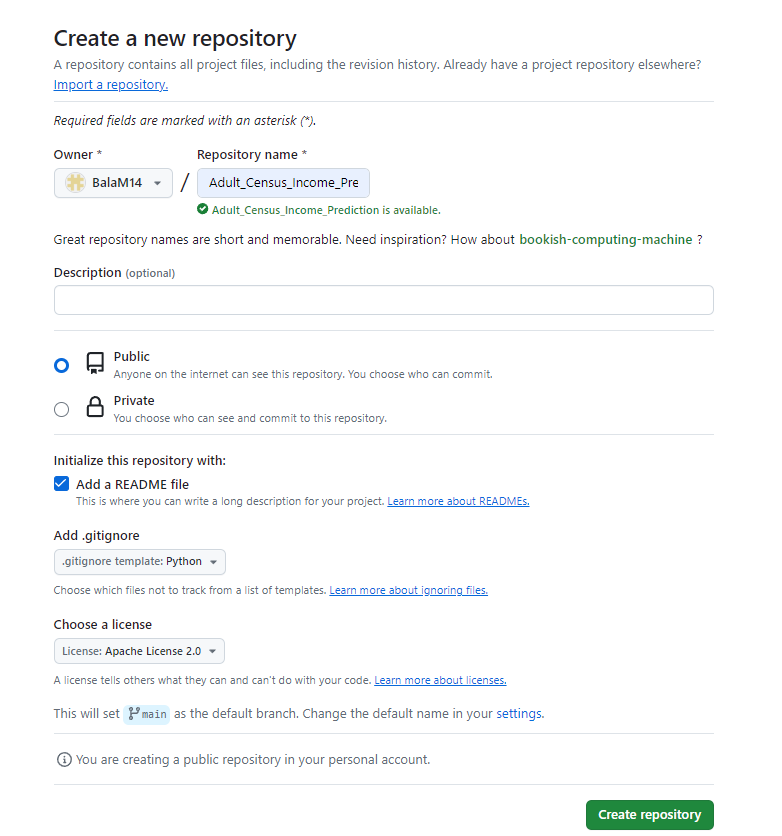
END TO END PROJECT SOP

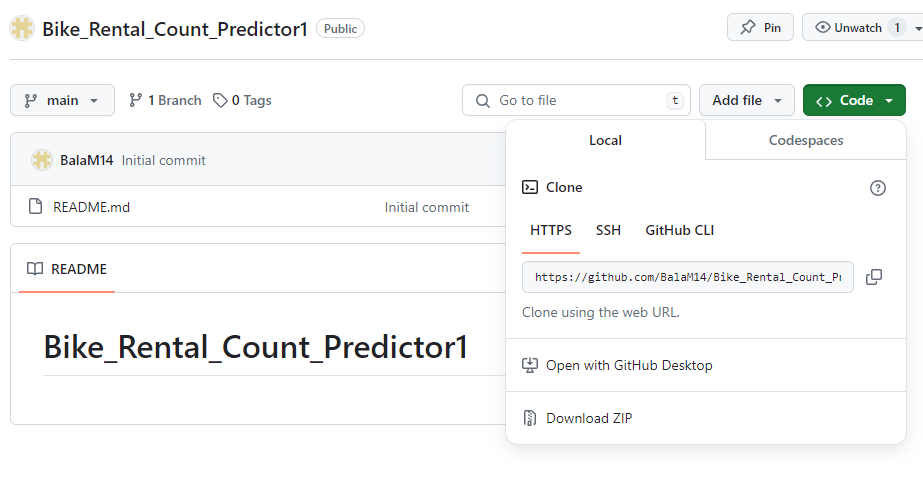
Contents:

1. Git Repository
2. Virtual Environment
3. Requirement
4. Test app.py file
5. Docker File & .dockerignore
6. .github folder
7. Project Parent Folder
8. Setup file
9. Project Child Folder
10. Architecture
11. Logger
12. Exception
13. Component
14. Entity
15. Config
16. Configuration
17. Utils
18. Constants
19. Configuration of components wrt Entity
20. Artifact Entity
21. Components – Data Ingestion
22. Pipeline – Data Ingestion
23. Components – Data Validation
    1. Data Drift
24. Pipeline – Data validation
25. Components – Data transformation
26. Pipeline – Data Transformation
27. Git Repository:

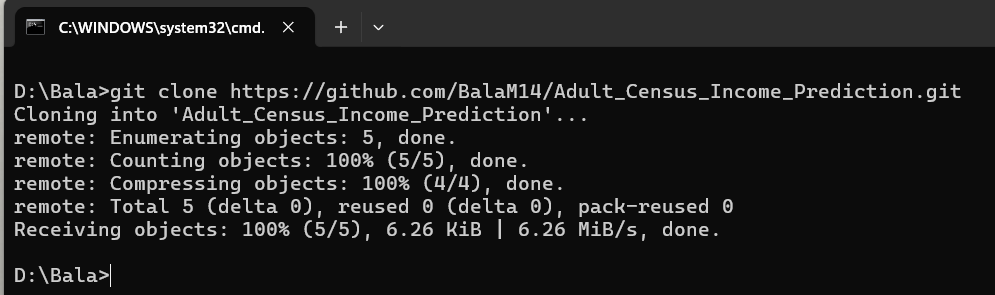
* Create a new repository in the github as below



* Get the github clone url from the github repository page.



* Open the cmd prompt & go to the path where you want to clone this repository.

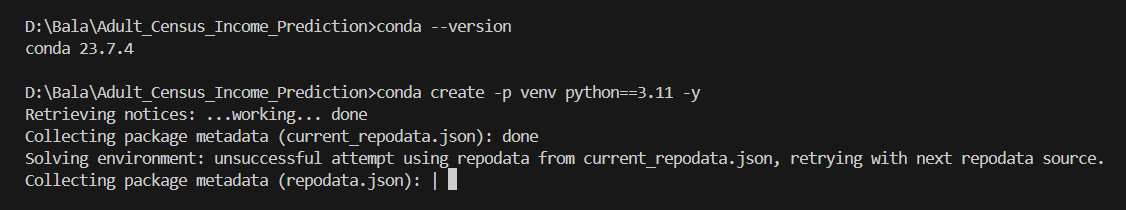


* Repository has been created & cloned to our local systems.

1. Virtual Environment:

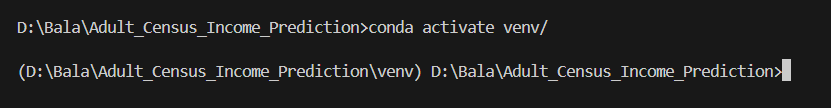
* Open the repository in the VS code
* Create the Virtual Environment for the project by using the below command in cmd.

Cmd: conda create –p venv python==3.11 -y



* Activate the newly created environment.

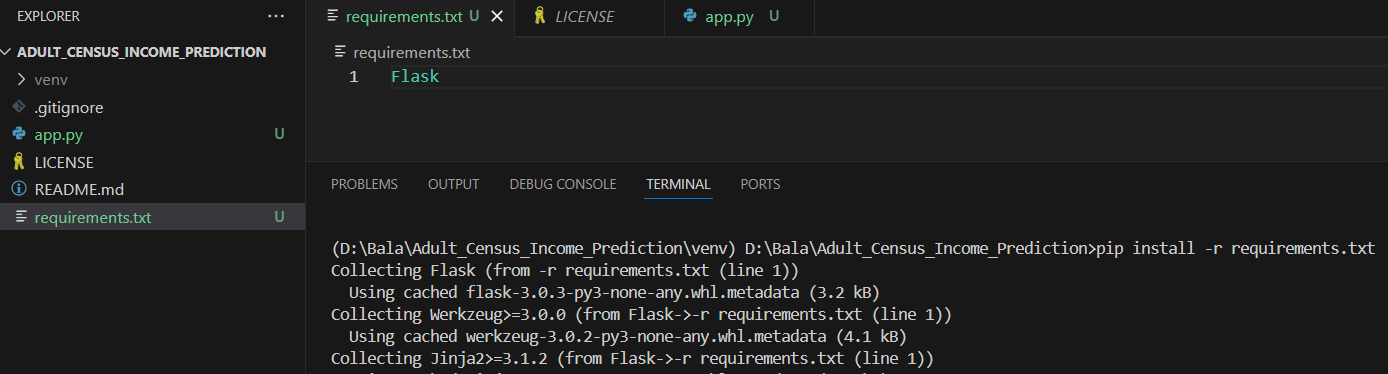
Cmd: conda activate venv



1. Requirement File:

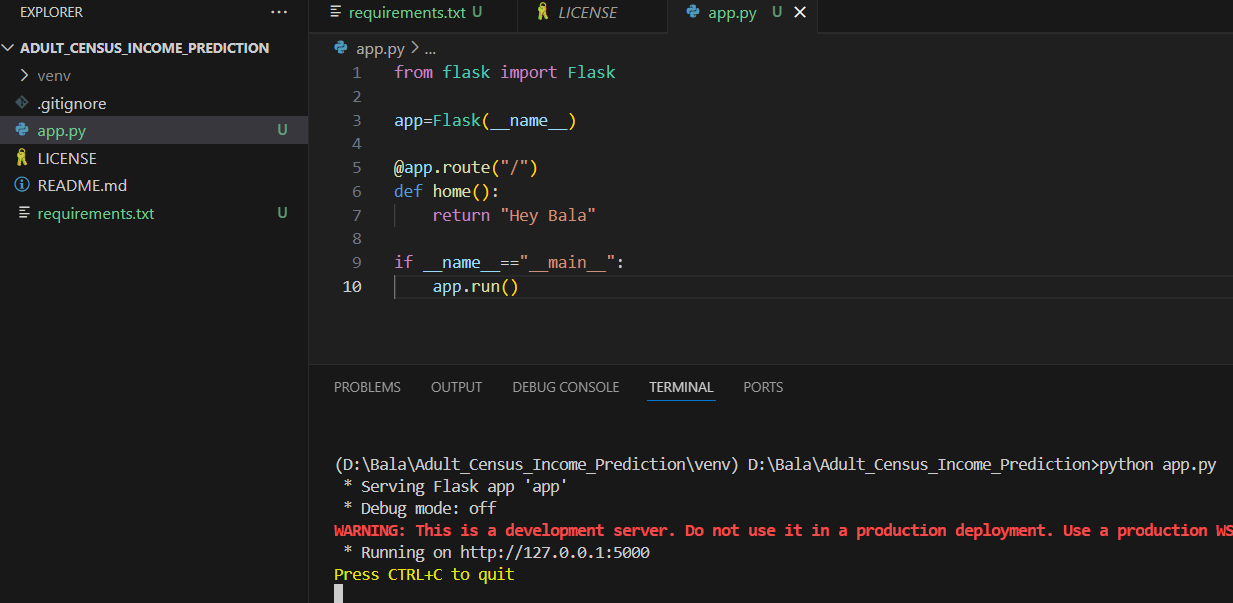
* Create the requirements.txt file to provide the package names to get install by default.

Cmd: pip install -r requirements.txt



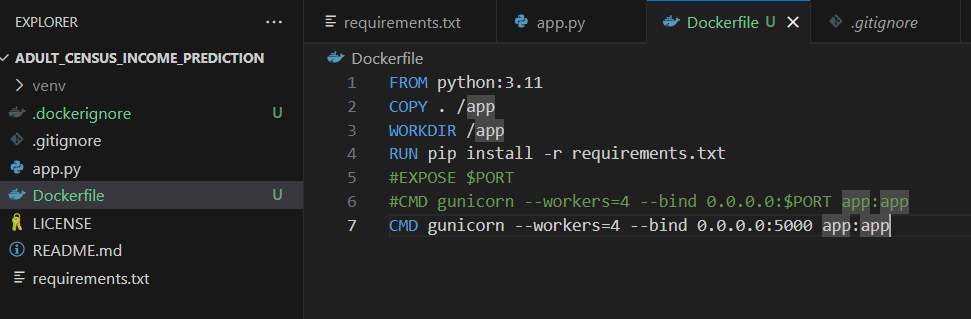
1. Test app.py file:

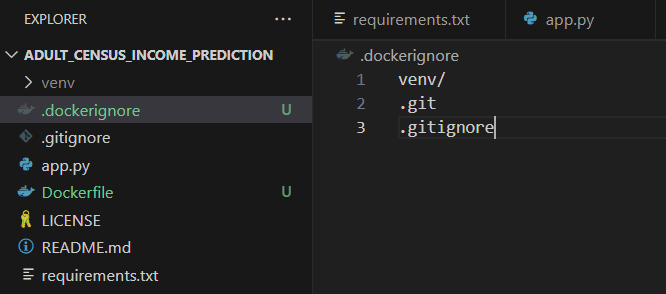
* Create the app.py file to write the demo code to check the project working.



1. Docker File & .dockerignore:

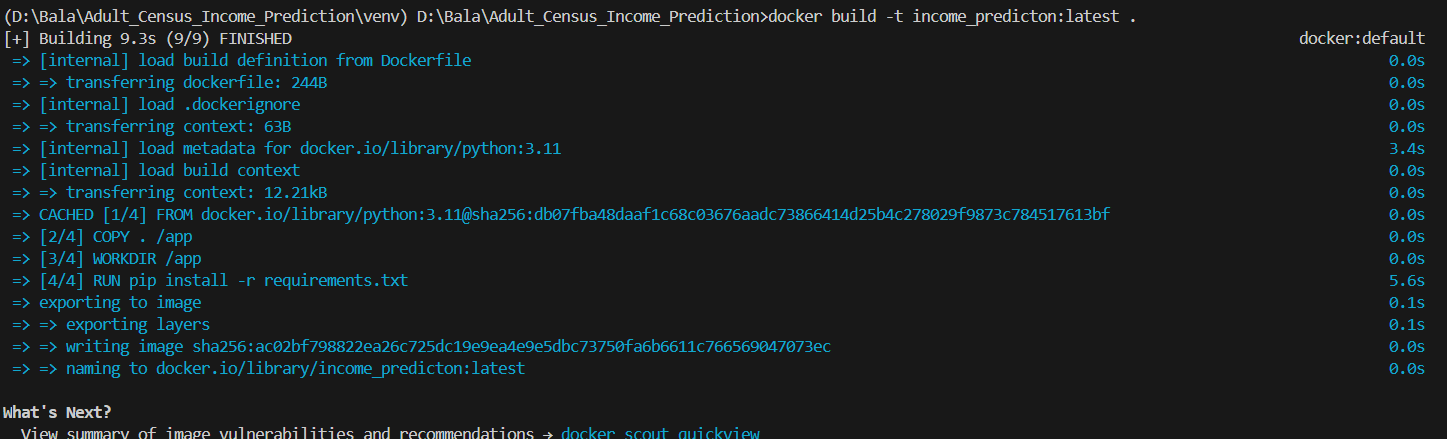
* Create a Dockerfile & .dockerignore file as mentioned below.





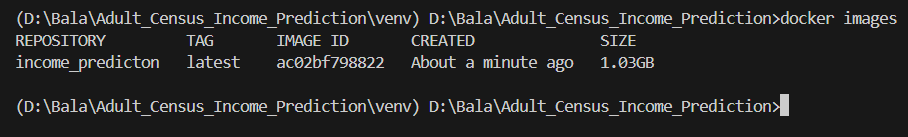
* Create a docker image by using the below command. Please make sure the docker desktop is running before creating the image.

Cmd : docker build -t income\_predicton:latest .



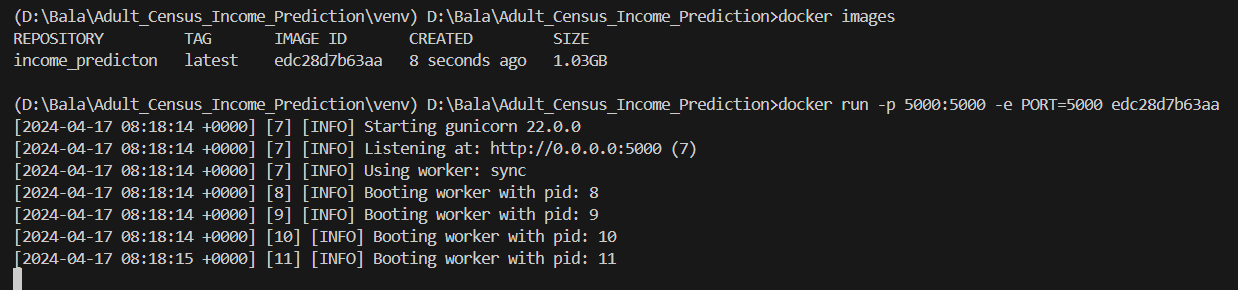
* To list the docker images use the below command.

Cmd : docker images



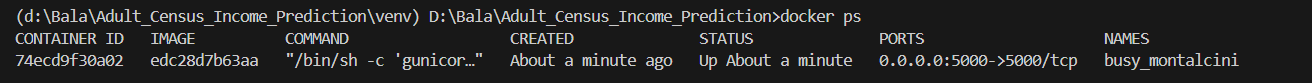
* To run the docker image use the below command.

Cmd: docker run -p 5000:5000 -e PORT=5000 <Image\_ID>



* To check the running image use the below command.

Cmd: docker ps



* To stop the cotainer use the below command.

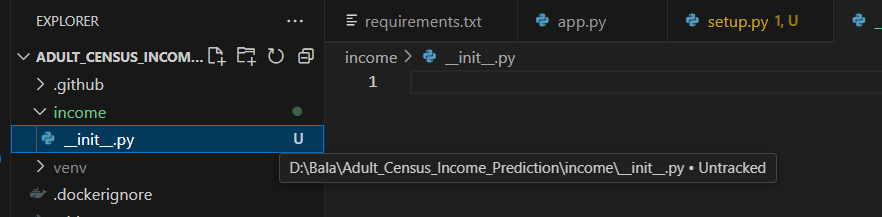
Cmd: docker stop <container\_id\_1st\_4\_char>



1. .github folder:
   * Create a .github folder & a subfolder name as workflows.
   * Inside the worflow folder create a main.yaml file as attached below.



1. Project Parent Folder:
   * Create a parent folder “income” which in future consists of all the components, entities & pipeline features.
   * Inside the housing folder, create the \_\_init\_\_.py file to use this folder as a package in later state.



1. Setup.py file:

* Create a setup.py file with the below details.

from setuptools import setup,find\_packages

from typing import List

PROJECT\_NAME="income\_predictor"

VERSION="0.0.3"

AUTHOR="Bala Murugan"

DESCRIPTION="This is a Incomee Predictor ML project"

REQUIREMENT\_FILE\_NAME="requirements.txt"

HYPHEN\_E\_DOT = "-e ."

def get\_requirements\_list() -> List[str]:

    with open(REQUIREMENT\_FILE\_NAME) as requirement\_file:

        requirement\_list = requirement\_file.readlines()

        requirement\_list = [requirement\_name.replace("\n", "") for requirement\_name in requirement\_list]

        if HYPHEN\_E\_DOT in requirement\_list:

            requirement\_list.remove(HYPHEN\_E\_DOT)

        return requirement\_list

setup(

    name=PROJECT\_NAME,

    version=VERSION,

    author=AUTHOR,

    description=DESCRIPTION,

    packages=find\_packages(),                   #Here we are using this find\_packages instead of housing is to check all the folders and wherever the \_\_init\_\_.py is there it will create a package

    install\_requires=get\_requirements\_list()

)

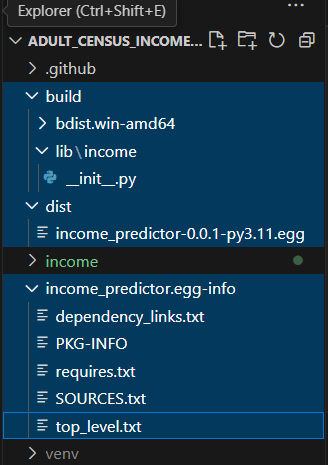
* To install the packges via setup.py file use the below command:

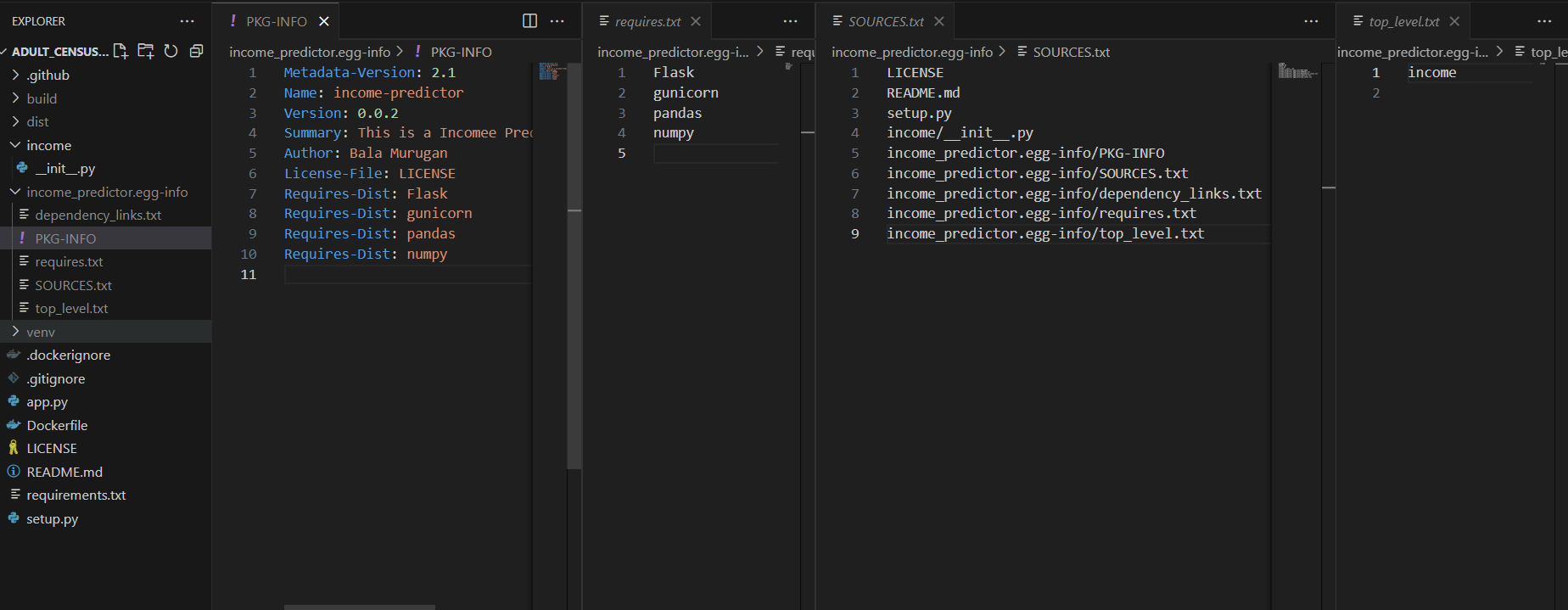
Cmd : pip install .

(or)

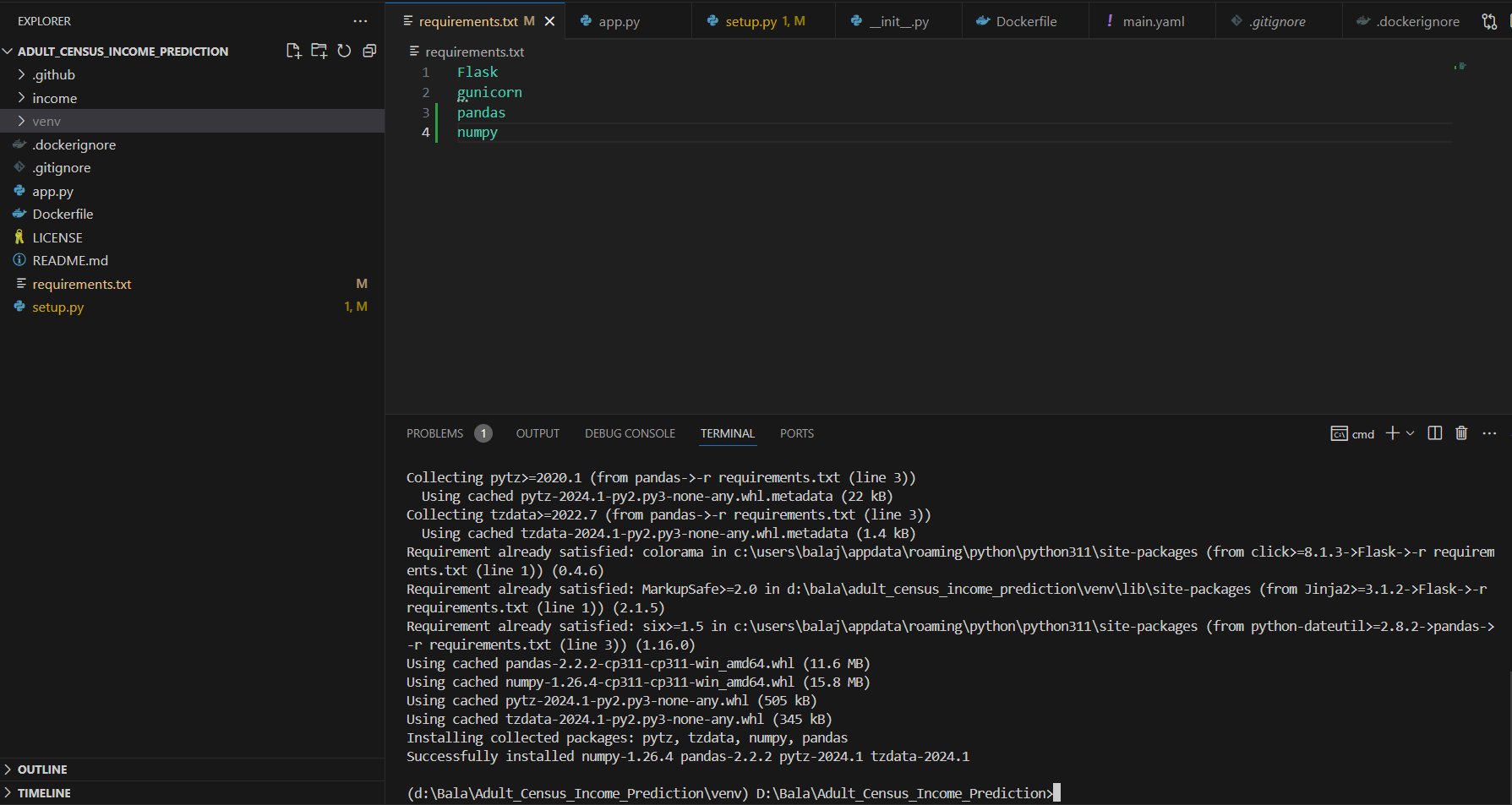
Cmd : python setup.py install

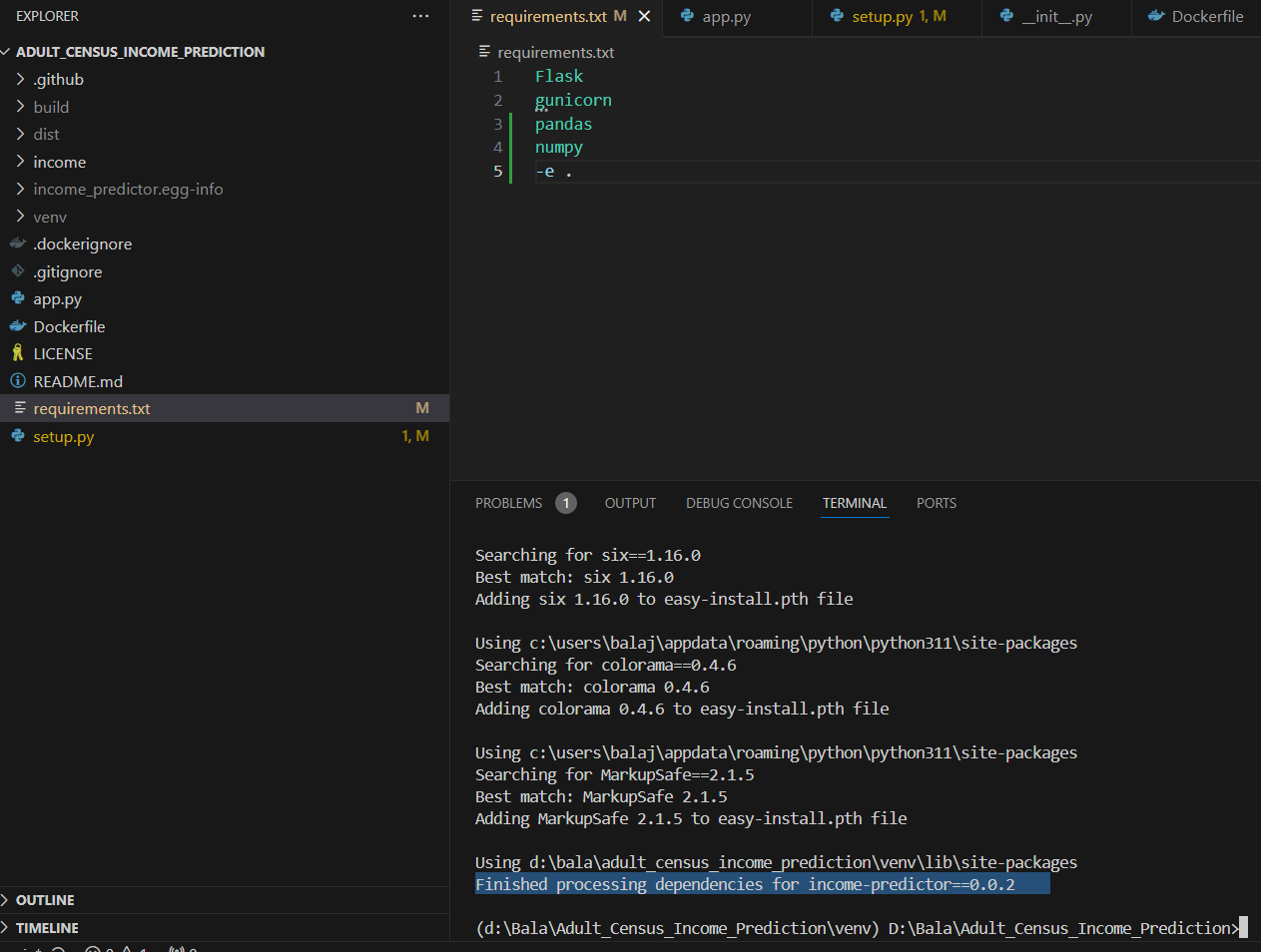
* Once you execute the above command, you will be able to see the newly genrated folders as mentioned below.





* In requirements.txt file we have specified “-e .” at the EOD file. This is bcoz we need to install all the packages in our project. If we execute the file without “-e .” then it will only install the specified packages in the requirements.txt file alone.





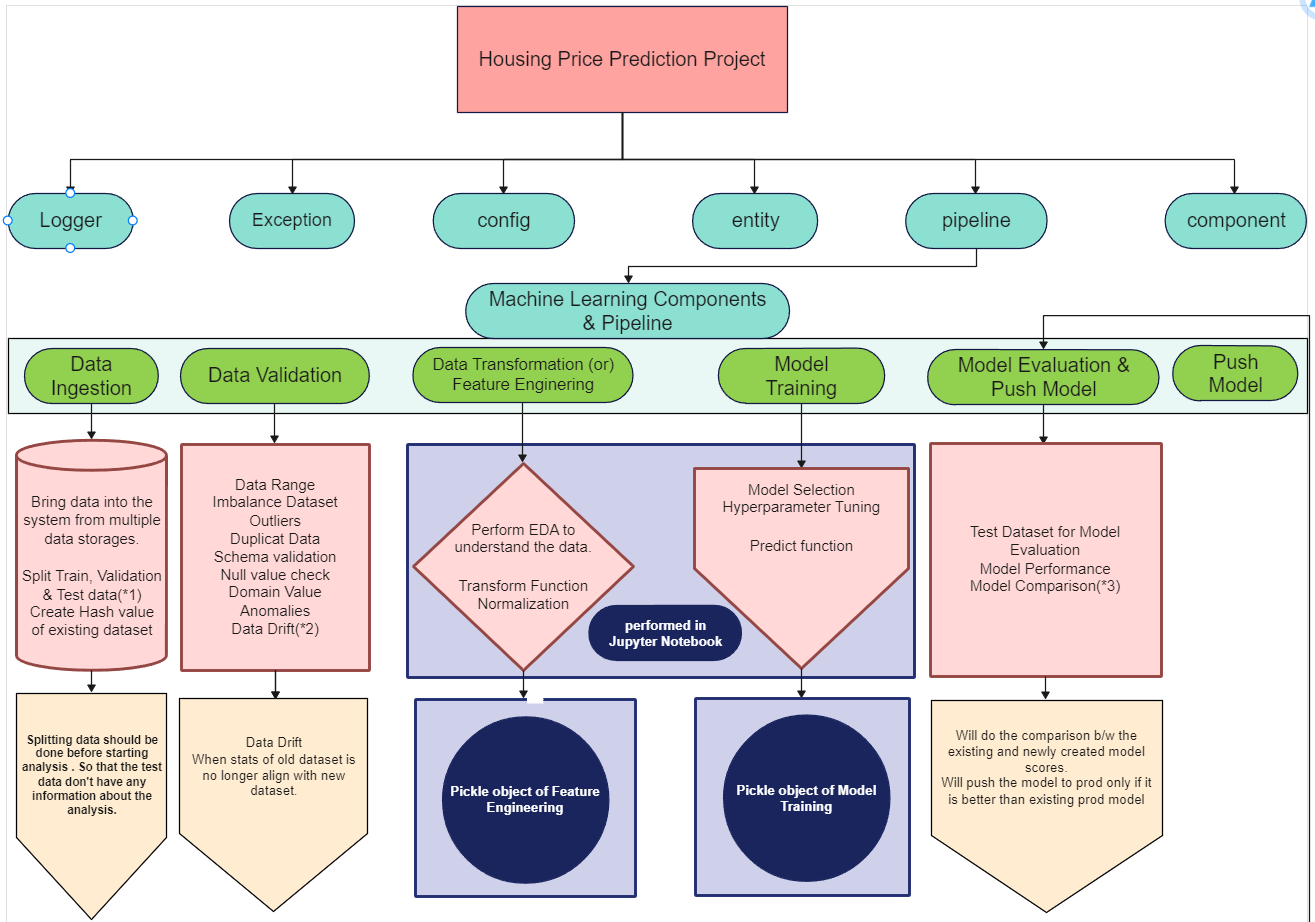
* If we are getting an file permission error then we need to change the value of version in the setup.py file.

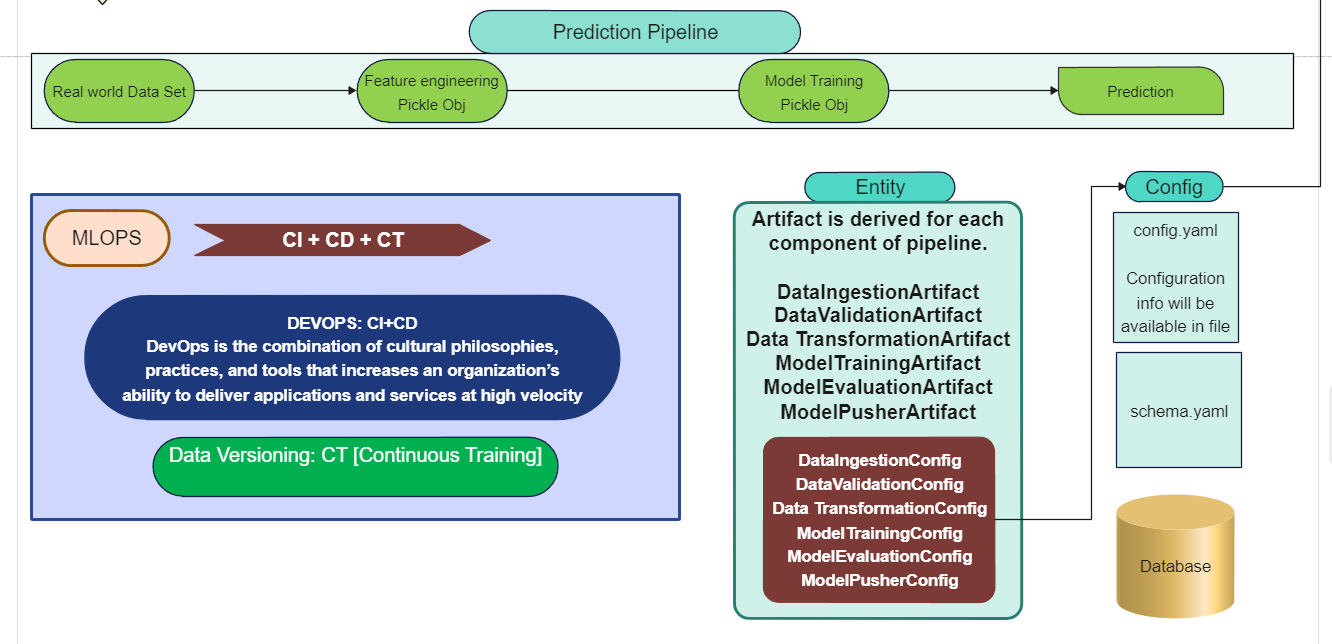
1. Project Child Folder:

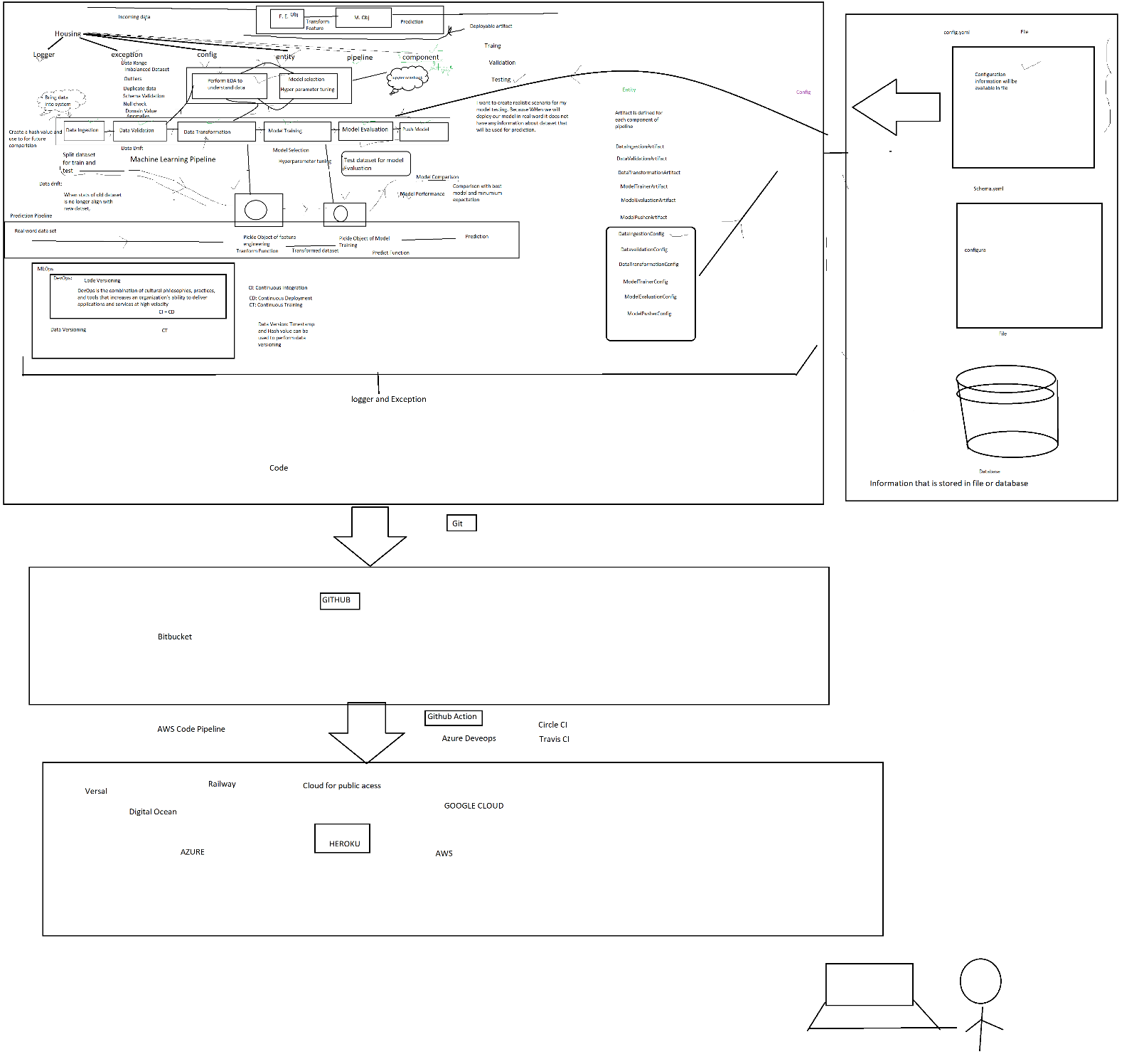
* Create the Project folder structure as below.

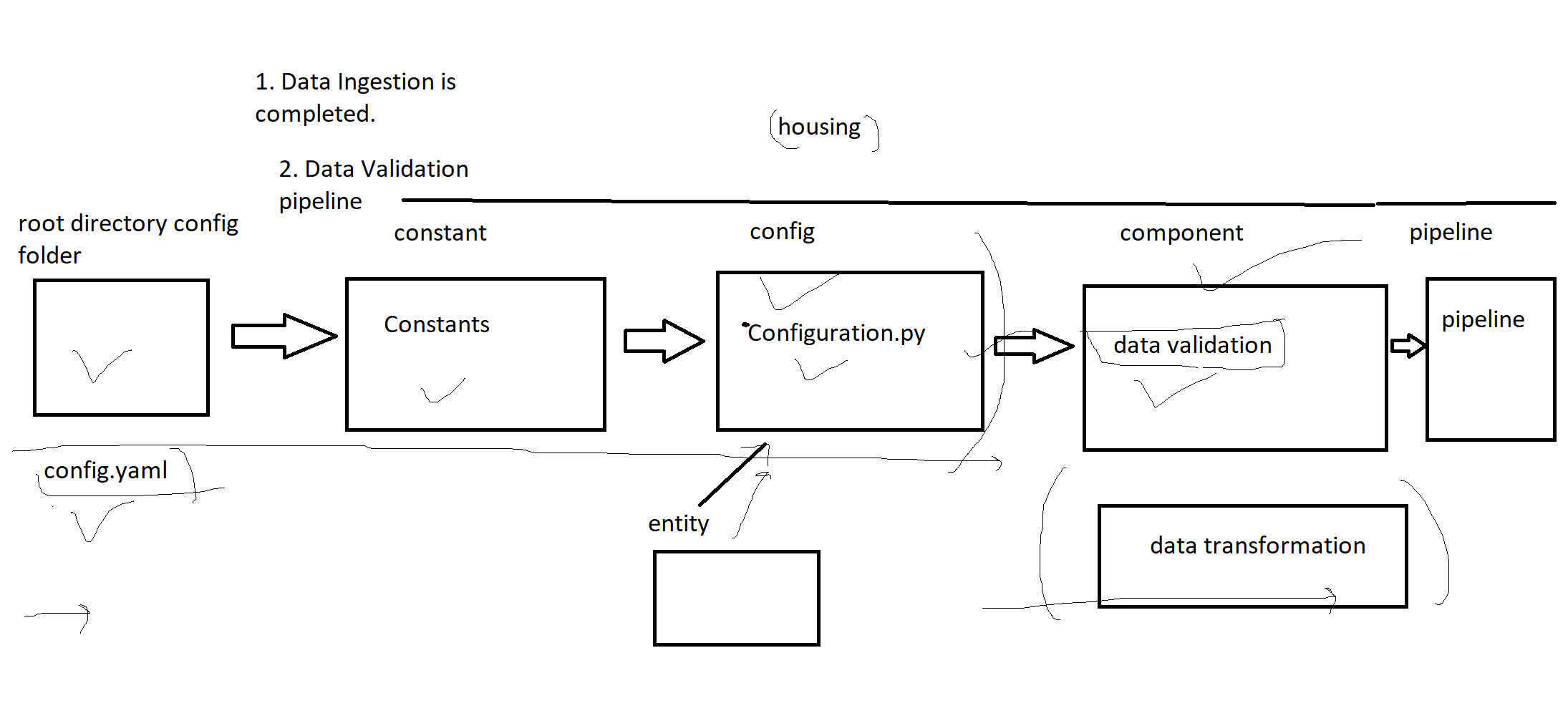


1. Architecture:









1. Logger:

* Start coding from the logger “\_\_init\_\_.py” file.

import logging

from datetime import datetime

import os

LOG\_DIR=f"income\_logs/{datetime.now().strftime('%Y\_%m\_%d')}"

CURRENT\_TIME\_STAMP = f"{datetime.now().strftime('%H\_%M\_%S')}"

LOG\_FILE\_NAME = f"log\_{CURRENT\_TIME\_STAMP}.log"

os.makedirs(LOG\_DIR,exist\_ok=True)

LOG\_FILE\_PATH = os.path.join(LOG\_DIR,LOG\_FILE\_NAME)

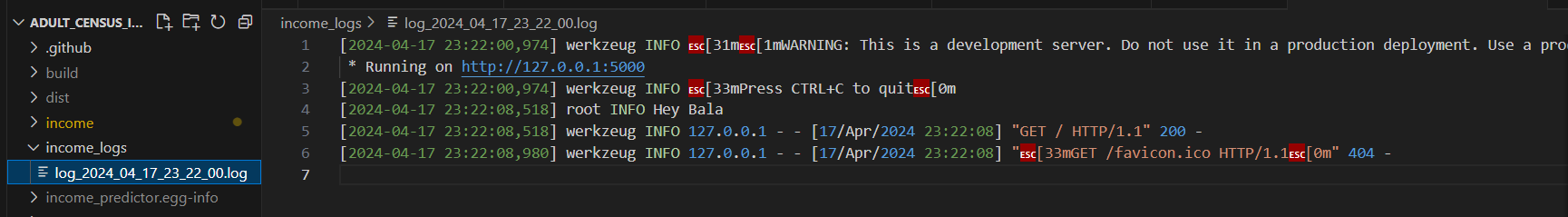
logging.basicConfig(filename=LOG\_FILE\_PATH,

                    filemode='w',

                    format='[%(asctime)s] %(name)s %(levelname)s %(message)s',

                    level=logging.INFO)

* Test the logger by importing the logger in app.py, so that it will generate the log file as below.



1. Exception:

* Start the coding for the exception.py file

import os

import sys

class IncomeException(Exception):

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

        super().\_\_init\_\_(error\_message)

        self.error\_message = IncomeException.get\_detailed\_error\_message(error\_message=error\_message,

                                                                        error\_detail=error\_detail)

    @staticmethod

    def get\_detailed\_error\_message(error\_message:Exception,error\_detail:sys) -> str:

        """

        error\_message : Exception object

        error\_detail: object of sys module

        """

        \_,\_,exec\_tb = error\_detail.exc\_info()               #ignore the 1st 2 values -> type & value and fetching only the traceback

        line\_number = exec\_tb.tb\_frame.f\_lineno

        file\_name = exec\_tb.tb\_frame.f\_code.co\_filename

        error\_message = f"Error occured in scrip : [{file\_name}] at line number : [{line\_number}] error message : [{error\_message}]"

        return error\_message

    def \_\_str\_\_(self) -> str:

        return self.error\_message

    def \_\_repr\_\_(self) -> str:

        return IncomeException.\_\_name\_\_.str()

* Test the exception by importing the IncomeException in app.py, so that it will generate the log file with the error as exception with custom error values as shown below.

**app.py**

from flask import Flask

from income.logger import logging

from income.exception import IncomeException

import sys

app=Flask(\_\_name\_\_)

@app.route("/",methods=['GET','POST'])

def index():

    try:

        raise Exception("we are testing custom exception")

    except Exception as e:

        income = IncomeException(e,sys)

        logging.info(income.error\_message)

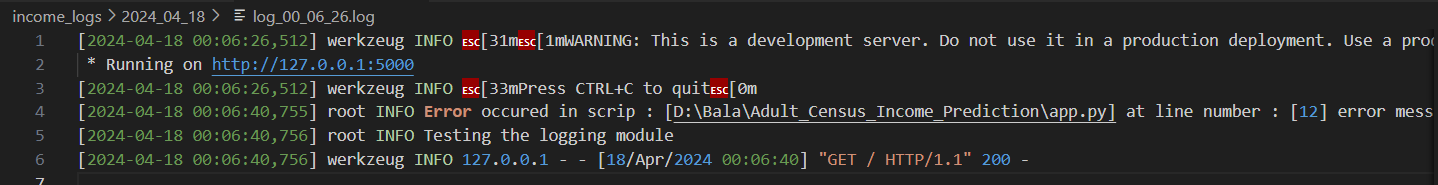
        logging.info("Testing the logging module")

    return "Hey Bala"

if \_\_name\_\_=="\_\_main\_\_":

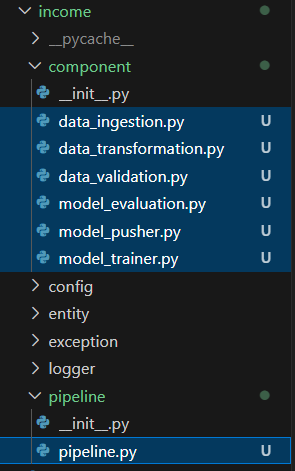
    app.run()

Log File:



1. Component:

* Create the component python files for each stages as below & same for pipeline folder also.



1. Entity

* Create the “config\_entity.py” file under the entity folder.

from collections import namedtuple

DataIngestionConfig = namedtuple("DataIngestionConfig",["dataset\_download\_url","tgz\_download\_dir",

                                                        "raw\_data\_dir","ingested\_train\_dir",

                                                        "ingested\_test\_dir"])

DataValidationConfig = namedtuple("DataValidationConfig",["schema\_file\_path","report\_file\_path","report\_page\_file\_path"])

DataTransformationConfig = namedtuple("DataTransformationConfig",["transformed\_train\_dir",

                                                                  "tramsformed\_test\_dir",

                                                                  "preprocessed\_object\_file\_path"])

ModelTrainerConfig = namedtuple("ModelTrainerConfig",["trained\_model\_file\_path","base\_accuracy"])

ModelEvaluationConfig = namedtuple("ModelEvaluationConfig",["model\_evaluation\_file\_path","time\_stamp"])

ModelPusherConfig = namedtuple("ModelPusherConfig",["export\_dir\_path"])

TrainingPipelineConfig = namedtuple("TrainingPipelineConfig",["artifact\_dir"])

1. Config:

Create a config file in the main path to maintain the generic configurations like schema’s & yaml files.

Inside the config folder create the config.yaml file as show below.



1. Configuration:

Create a configuration.py file under the income/config folder

from income.entity.config\_entity import DataIngestionConfig,DataValidationConfig,DataTransformationConfig, \

                                        ModelTrainerConfig,ModelEvaluationConfig,ModelPusherConfig,TrainingPipelineConfig

from income.util.util import read\_yaml\_file

import os,sys

from income.constant import \*

from income.exception import IncomeException

from income.logger import logging

class Configuration:

    def \_\_init\_\_(self) -> None:

        pass

    def get\_data\_ingestion\_config(self) -> DataIngestionConfig:

        pass

    def get\_data\_validation\_config(self) -> DataValidationConfig:

        pass

    def get\_data\_transformation\_config(self) -> DataTransformationConfig:

        pass

    def get\_model\_trainer\_config(self) -> ModelTrainerConfig:

        pass

    def get\_model\_evaluation\_config(self) -> ModelEvaluationConfig:

        pass

    def get\_model\_pusher\_config(self) -> ModelPusherConfig:

        pass

    def get\_training\_pipeline\_config(self) -> TrainingPipelineConfig:

        pass

1. Utils:

Create a util.py file under the util folder as shown below.

import yaml

from income.exception import IncomeException

import os,sys

def read\_yaml\_file(file\_path:str) -> dict:

    """

    Reads a Yaml file and returns the contents as a dictionary.

    file\_path:str

    """

    try:

        with open(file\_path,"rb") as yaml\_file:

            return yaml.safe\_load(yaml\_file)

    except Exception as e:

        raise IncomeException(e,sys) from e

1. Constants:

Create a constant folder with the python file as constant.py

import os,sys

from datetime import datetime

ROOT\_DIR = os.getcwd()   #To get the current working directory

CONFIG\_DIR = "config"

CONFIG\_FILE\_NAME = "config.yaml"

CONFG\_FILE\_PATH = os.path.join(ROOT\_DIR,CONFIG\_DIR,CONFIG\_FILE\_NAME)

CURRENT\_TIME\_STAMP = f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}"

Update: Configuration.py

from income.entity.config\_entity import DataIngestionConfig,DataValidationConfig,DataTransformationConfig, \

                                        ModelTrainerConfig,ModelEvaluationConfig,ModelPusherConfig,TrainingPipelineConfig

from income.util.util import read\_yaml\_file

import os,sys

from income.constant import \*

from income.exception import IncomeException

from income.logger import logging

class Configuration:

    def \_\_init\_\_(self,config\_file\_path:str=CONFG\_FILE\_PATH, current\_time\_stamp:str=CURRENT\_TIME\_STAMP) -> None:

        try:

            self.config\_info = read\_yaml\_file(file\_path=config\_file\_path)

            self.training\_pipeline\_config = self.get\_training\_pipeline\_config()

            self.time\_stamp = current\_time\_stamp

        except Exception as e:

            raise IncomeException(e,sys) from e

    def get\_data\_ingestion\_config(self) -> DataIngestionConfig:

        pass

    def get\_data\_validation\_config(self) -> DataValidationConfig:

        pass

    def get\_data\_transformation\_config(self) -> DataTransformationConfig:

        pass

    def get\_model\_trainer\_config(self) -> ModelTrainerConfig:

        pass

    def get\_model\_evaluation\_config(self) -> ModelEvaluationConfig:

        pass

    def get\_model\_pusher\_config(self) -> ModelPusherConfig:

        pass

    def get\_training\_pipeline\_config(self) -> TrainingPipelineConfig:

        pass

Update: constant.py

import os,sys

from datetime import datetime

ROOT\_DIR = os.getcwd()   #To get the current working directory

CONFIG\_DIR = "config"

CONFIG\_FILE\_NAME = "config.yaml"

CONFG\_FILE\_PATH = os.path.join(ROOT\_DIR,CONFIG\_DIR,CONFIG\_FILE\_NAME)

CURRENT\_TIME\_STAMP = f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}"

#Training Pipeline related variablee

TRAINING\_PIPELINE\_CONFIG\_KEY = "training\_pipeline\_config"

TRAINING\_PIPELINE\_ARTIFACT\_DIR\_KEY = "artifact\_dir"

TRAINING\_PIPELINE\_NAME\_KEY = "pipeline\_name"

Update: configuration.py

from income.entity.config\_entity import DataIngestionConfig,DataValidationConfig,DataTransformationConfig, \

                                        ModelTrainerConfig,ModelEvaluationConfig,ModelPusherConfig,TrainingPipelineConfig

from income.util.util import read\_yaml\_file

import os,sys

from income.constant import \*

from income.exception import IncomeException

from income.logger import logging

class Configuration:

    def \_\_init\_\_(self,config\_file\_path:str=CONFG\_FILE\_PATH, current\_time\_stamp:str=CURRENT\_TIME\_STAMP) -> None:

        try:

            self.config\_info = read\_yaml\_file(file\_path=config\_file\_path)

            self.training\_pipeline\_config = self.get\_training\_pipeline\_config()

            self.time\_stamp = current\_time\_stamp

        except Exception as e:

            raise IncomeException(e,sys) from e

    def get\_data\_ingestion\_config(self) -> DataIngestionConfig:

        pass

    def get\_data\_validation\_config(self) -> DataValidationConfig:

        pass

    def get\_data\_transformation\_config(self) -> DataTransformationConfig:

        pass

    def get\_model\_trainer\_config(self) -> ModelTrainerConfig:

        pass

    def get\_model\_evaluation\_config(self) -> ModelEvaluationConfig:

        pass

    def get\_model\_pusher\_config(self) -> ModelPusherConfig:

        pass

    def get\_training\_pipeline\_config(self) -> TrainingPipelineConfig:

        try:

            training\_pipeline\_config = self.config\_info[TRAINING\_PIPELINE\_CONFIG\_KEY]

            artifact\_dir = os.path.join(ROOT\_DIR,training\_pipeline\_config[TRAINING\_PIPELINE\_NAME\_KEY],

                                        training\_pipeline\_config[TRAINING\_PIPELINE\_ARTIFACT\_DIR\_KEY])

            training\_pipeline\_config=TrainingPipelineConfig(artifact\_dir=artifact\_dir)

            logging.info(f"Training Pipeline config: {training\_pipeline\_config}")

            return training\_pipeline\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

1. Configuration of Component wrt Entity:

Constant

import os,sys

from datetime import datetime

ROOT\_DIR = os.getcwd()   #To get the current working directory

CONFIG\_DIR = "config"

CONFIG\_FILE\_NAME = "config.yaml"

CONFG\_FILE\_PATH = os.path.join(ROOT\_DIR,CONFIG\_DIR,CONFIG\_FILE\_NAME)

CURRENT\_TIME\_STAMP = f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}"

#Training Pipeline related variablee

TRAINING\_PIPELINE\_CONFIG\_KEY = "training\_pipeline\_config"

TRAINING\_PIPELINE\_ARTIFACT\_DIR\_KEY = "artifact\_dir"

TRAINING\_PIPELINE\_NAME\_KEY = "pipeline\_name"

#Data Ingestion related variable

DATA\_INGESTION\_CONFIG\_KEY="data\_ingestion\_config"

DATA\_INGESTION\_ARTIFACT\_DIR="data\_ingestion"                #This is not a KEY variable its a direct constant

DATA\_INGESTION\_DOWNLOAD\_URL\_KEY="dataset\_download\_url"

DATA\_INGESTION\_RAW\_DATA\_DIR\_KEY="raw\_data\_dir"

DATA\_INGESTION\_TGZ\_DOWNLOAD\_DIR\_KEY="tgz\_download\_dir"

DATA\_INGESTION\_INGESTED\_DIR\_KEY="ingested\_dir"

DATA\_INGESTION\_INGESTED\_TRAIN\_DIR\_KEY="ingested\_train\_dir"

DATA\_INGESTION\_INGESTED\_TEST\_DIR\_KEY="ingested\_test\_dir"

#Data Validation related variable

DATA\_VALIDATION\_CONFIG\_KEY="data\_validation\_config"

DATA\_VALIDATION\_ARTIFACT\_DIR="data\_validation"

DATA\_VALIDATION\_SCHEMA\_DIR\_KEY="schema\_dir"

DATA\_VALIDATION\_SCHEMA\_FILE\_NAME\_KEY="schema\_file\_name"

DATA\_VALIDATION\_REPORT\_FILE\_NAME\_KEY="report\_file\_name"

DATA\_VALIDATION\_REPORT\_PAGE\_FILE\_NAME\_KEY="report\_page\_file\_name"

#Data Transformation related variable

DATA\_TRANSFORMATION\_CONFIG\_KEY="data\_transformation\_config"

DATA\_TRANSFORMATION\_ARTIFACT\_DIR="data\_transformation"

DATA\_TRANSFORMATION\_TRANSFORMED\_DIR\_KEY="transformed\_dir"

DATA\_TRANSFORMATION\_TRANSFORMED\_TRAIN\_DIR\_KEY="transformed\_train\_dir"

DATA\_TRANSFORMATION\_TRANSFORMED\_TEST\_DIR\_KEY="transformed\_test\_dir"

DATA\_TRANSFORMATION\_PREPROCESSING\_DIR\_KEY="preprocessing\_dir"

DATA\_TRANSFORMATION\_PREPROCESSED\_OBJECT\_FILE\_NAME="preprocessed\_object\_file\_name"

#Model Trainer related variable

MODEL\_TRAINER\_CONFIG\_KEY="model\_trainer\_config"

MODEL\_TRAINER\_ARTIFACT\_DIR="model\_trainer"

MODEL\_TRAINER\_TRAINED\_MODEL\_DIR\_KEY="trained\_model\_dir"

MODEL\_TRAINER\_MODEL\_FILE\_NAME\_KEY="model\_file\_name"

MODEL\_TRAINER\_BASE\_ACCURACY\_KEY="base\_accuracy"

MODEL\_TRAINER\_MODEL\_CONFIG\_DIR\_KEY="model\_config\_dir"

MODEL\_TRAINER\_MODEL\_CONFIG\_FILE\_NAME\_KEY="model\_config\_file\_name"

#Model Evaluation related variable

MODEL\_EVALUATION\_CONFIG\_KEY="model\_evaluation\_config"

MODEL\_EVALUATION\_ARTIFACT\_DIR="model\_evaluation"

MODEL\_EVALUATION\_FILE\_NAME="model\_evaluation\_file\_name"

#Model Pusher related variable

MODEL\_PUSHER\_CONFIG\_KEY="model\_pusher\_config"

MODEL\_PUSHER\_ARTIFACT\_DIR="model\_pusher"

MODEL\_PUSHER\_MODEL\_EXPORT\_DIR="model\_export\_dir"

Configuration:

* Data Ingestion

    def get\_data\_ingestion\_config(self) -> DataIngestionConfig:

        try:

            artifact\_dir = self.training\_pipeline\_config.artifact\_dir

            data\_ingestion\_artifact\_dir = os.path.join(artifact\_dir,DATA\_INGESTION\_ARTIFACT\_DIR,self.time\_stamp)

            data\_ingestion\_config = self.config\_info[DATA\_INGESTION\_CONFIG\_KEY]

            dataset\_download\_url = data\_ingestion\_config[DATA\_INGESTION\_DOWNLOAD\_URL\_KEY]

            tgz\_download\_dir = os.path.join(data\_ingestion\_artifact\_dir,data\_ingestion\_config[DATA\_INGESTION\_TGZ\_DOWNLOAD\_DIR\_KEY])

            raw\_data\_dir = os.path.join(data\_ingestion\_artifact\_dir,data\_ingestion\_config[DATA\_INGESTION\_RAW\_DATA\_DIR\_KEY])

            ingested\_dir = os.path.join(data\_ingestion\_artifact\_dir,data\_ingestion\_config[DATA\_INGESTION\_INGESTED\_DIR\_KEY])

            ingested\_train\_dir = os.path.join(ingested\_dir,data\_ingestion\_config[DATA\_INGESTION\_INGESTED\_TRAIN\_DIR\_KEY])

            ingested\_test\_dir = os.path.join(ingested\_dir,data\_ingestion\_config[DATA\_INGESTION\_INGESTED\_TEST\_DIR\_KEY])

            data\_ingestion\_config = DataIngestionConfig(dataset\_download\_url=dataset\_download\_url,

                                                        tgz\_download\_dir=tgz\_download\_dir,

                                                        raw\_data\_dir=raw\_data\_dir,

                                                        ingested\_train\_dir=ingested\_train\_dir,

                                                        ingested\_test\_dir=ingested\_test\_dir)

            logging.info(f"Data Ingestion config: {data\_ingestion\_config}")

            return data\_ingestion\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

* Data Validation:

    def get\_data\_validation\_config(self) -> DataValidationConfig:

        try:

            artifact\_dir = self.training\_pipeline\_config.artifact\_dir

            data\_validation\_artifact\_dir = os.path.join(artifact\_dir,DATA\_VALIDATION\_ARTIFACT\_DIR,self.time\_stamp)

            data\_validation\_config = self.config\_info[DATA\_VALIDATION\_CONFIG\_KEY]

            schema\_dir = os.path.join(ROOT\_DIR,data\_validation\_config[DATA\_VALIDATION\_SCHEMA\_DIR\_KEY])

            schema\_file\_path = os.path.join(schema\_dir,data\_validation\_config[DATA\_VALIDATION\_SCHEMA\_FILE\_NAME\_KEY])

            report\_file\_path = os.path.join(data\_validation\_artifact\_dir,data\_validation\_config[DATA\_VALIDATION\_REPORT\_FILE\_NAME\_KEY])

            report\_page\_file\_path = os.path.join(data\_validation\_artifact\_dir,data\_validation\_config[DATA\_VALIDATION\_REPORT\_PAGE\_FILE\_NAME\_KEY])

            data\_validation\_config = DataValidationConfig(schema\_file\_path=schema\_file\_path,

                                                          report\_file\_path=report\_file\_path,

                                                          report\_page\_file\_path=report\_page\_file\_path)

            logging.info(f"Data Validation Config: {data\_validation\_config}")

            return data\_validation\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

* Data Transformation

    def get\_data\_transformation\_config(self) -> DataTransformationConfig:

        try:

            artifact\_dir = self.training\_pipeline\_config.artifact\_dir

            data\_transformation\_artifact\_dir = os.path.join(artifact\_dir,DATA\_TRANSFORMATION\_ARTIFACT\_DIR,self.time\_stamp)

            data\_transformation\_config = self.config\_info[DATA\_TRANSFORMATION\_CONFIG\_KEY]

            transformed\_dir = os.path.join(data\_transformation\_artifact\_dir,data\_transformation\_config[DATA\_TRANSFORMATION\_TRANSFORMED\_DIR\_KEY])

            transformed\_train\_dir = os.path.join(transformed\_dir,data\_transformation\_config[DATA\_TRANSFORMATION\_TRANSFORMED\_TRAIN\_DIR\_KEY])

            transformed\_test\_dir = os.path.join(transformed\_dir,data\_transformation\_config[DATA\_TRANSFORMATION\_TRANSFORMED\_TEST\_DIR\_KEY])

            preprocessing\_dir = os.path.join(data\_transformation\_artifact\_dir,data\_transformation\_config[DATA\_TRANSFORMATION\_PREPROCESSING\_DIR\_KEY])

            preprocessed\_object\_filed\_path = os.path.join(preprocessing\_dir,data\_transformation\_config[DATA\_TRANSFORMATION\_PREPROCESSED\_OBJECT\_FILE\_NAME])

            data\_transformation\_config = DataTransformationConfig(transformed\_train\_dir=transformed\_train\_dir,

                                                                  tramsformed\_test\_dir=transformed\_test\_dir,

                                                                  preprocessed\_object\_file\_path=preprocessed\_object\_filed\_path)

            logging.info(f"Data Transformation Config: {data\_transformation\_config}")

            return data\_transformation\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

* Model Trainer:

    def get\_model\_trainer\_config(self) -> ModelTrainerConfig:

        try:

            artifact\_dir = self.training\_pipeline\_config.artifact\_dir

            model\_trainer\_artifact\_dir = os.path.join(artifact\_dir,MODEL\_TRAINER\_ARTIFACT\_DIR,self.time\_stamp)

            model\_trainer\_config = self.config\_info[MODEL\_TRAINER\_CONFIG\_KEY]

            trained\_model\_dir = os.path.join(model\_trainer\_artifact\_dir,model\_trainer\_config[MODEL\_TRAINER\_TRAINED\_MODEL\_DIR\_KEY])

            trained\_model\_file\_path = os.path.join(trained\_model\_dir,model\_trainer\_config[MODEL\_TRAINER\_MODEL\_FILE\_NAME\_KEY])

            base\_accuracy = model\_trainer\_config[MODEL\_TRAINER\_BASE\_ACCURACY\_KEY]

            model\_config\_dir = os.path.join(ROOT\_DIR,model\_trainer\_config[MODEL\_TRAINER\_MODEL\_CONFIG\_DIR\_KEY])

            model\_config\_file\_name = os.path.join(model\_config\_dir,model\_trainer\_config[MODEL\_TRAINER\_MODEL\_CONFIG\_FILE\_NAME\_KEY])

            model\_trainer\_config = ModelTrainerConfig(trained\_model\_file\_path=trained\_model\_file\_path,

                                                      base\_accuracy=base\_accuracy)

            logging.info(f"Model Trainer Config: {model\_trainer\_config}")

            return model\_trainer\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

* Model Evaluation:

    def get\_model\_evaluation\_config(self) -> ModelEvaluationConfig:

        try:

            artifact\_dir = self.training\_pipeline\_config.artifact\_dir

            model\_evaluation\_artifact\_dir = os.path.join(artifact\_dir,MODEL\_EVALUATION\_ARTIFACT\_DIR, self.time\_stamp)

            model\_evaluation\_config = self.config\_info[MODEL\_EVALUATION\_CONFIG\_KEY]

            model\_evauation\_file\_path = os.path.join(model\_evaluation\_artifact\_dir,model\_evaluation\_config[MODEL\_EVALUATION\_FILE\_NAME])

            model\_evaluation\_config = ModelEvaluationConfig(model\_evaluation\_file\_path=model\_evauation\_file\_path,

                                                            time\_stamp=self.time\_stamp)

            logging.info(f"Model Evaluation Config: {model\_evaluation\_config}")

            return model\_evaluation\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

* Model Pusher:

    def get\_model\_pusher\_config(self) -> ModelPusherConfig:

        try:

            artifact\_dir = self.training\_pipeline\_config.artifact\_dir

            model\_pusher\_artifact\_dir = os.path.join(artifact\_dir,MODEL\_PUSHER\_ARTIFACT\_DIR,self.time\_stamp)

            model\_pusher\_config = self.config\_info[MODEL\_PUSHER\_CONFIG\_KEY]

            model\_export\_dir = os.path.join(artifact\_dir,model\_pusher\_config[MODEL\_PUSHER\_MODEL\_EXPORT\_DIR])

            model\_pusher\_config = ModelPusherConfig(export\_dir\_path=model\_export\_dir)

            logging.info(f"Model Pusher Config: {model\_pusher\_config}")

            return model\_pusher\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

1. Artifact Entity:

Create a artifact\_entity.py file under the entity folder as shown below.

from collections import namedtuple

DataIngestionArtifact = namedtuple("DataIngestionArtifact",["train\_file\_path", "test\_file\_path", "is\_ingested", "message"])

We need to update the artifact entity for each components as we did for Data Ingestion.

1. Components – Data Ingestion:

Started to write thee code for data\_ingestion.py component as shown below.

Here we have to specify all the steps of data ingestion, like in data ingestion 1st we need to download the dataset , extarct the downloaded dataset which is in zip file format, then split the dataset into train and test data.

import os,sys

from income.logger import logging

from income.exception import IncomeException

from income.config.configuration import DataIngestionConfig

from income.entity.artifact\_entity import DataIngestionArtifact

class DataIngestion:

    def \_\_init\_\_(self, data\_ingestion\_config:DataIngestionConfig) :

        try:

            logging.info(f"{'='\*20}Data Ingestion log started. {'='\*20}")

            self.data\_ingestion\_config = data\_ingestion\_config

        except Exception as e:

            raise IncomeException(e,sys) from e

    def download\_income\_data(self) -> str:

        pass

    def extract\_tgz\_file(self,tgz\_file\_path:str):

        pass

    def split\_data\_as\_train\_test(self):

        pass

    def initiate\_data\_ingestion(self) -> DataIngestionArtifact:

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

Download\_income\_data method:

We need to create a folder to save the zip file

Then we need to import “urlib” from “six.moves” to use urlretrieve to download thee data.

    def download\_income\_data(self) -> str:

        try:

            #Downloading the dataset

            download\_url = self.data\_ingestion\_config.dataset\_download\_url

            #Folder location to download file

            tgz\_download\_dir = self.data\_ingestion\_config.tgz\_download\_dir

            #To cheeck if the folder exists, but in this case it won't bcoz we using timestamp

            if os.path.exists(tgz\_download\_dir):

                os.remove(tgz\_download\_dir)

            #Create the tgz\_download directory

            os.makedirs(tgz\_download\_dir,exist\_ok=True)

            #To get the filename from the url

            income\_file\_name = os.path.basename(download\_url)

            #Download data in the below path

            tgz\_file\_path = os.path.join(tgz\_download\_dir,income\_file\_name)

            logging.info(f"Downloading file from [{download\_url}] into [{tgz\_file\_path}]")

            #Downloading the data from the url & saving it to tgz\_file\_path

            urllib.request.urlretrieve(download\_url,tgz\_file\_path)

            logging.info(f"File : [{tgz\_file\_path}] has beeen downloaded successfully")

            return tgz\_file\_path

        except Exception as e:

            raise IncomeException(e,sys) from e

Extract\_tgz\_file method:

    def extract\_tgz\_file(self,tgz\_file\_path:str):

        try:

            raw\_data\_dir = self.data\_ingestion\_config.raw\_data\_dir

            if os.path.exists(raw\_data\_dir):

                os.remove(raw\_data\_dir)

            #Create the raw\_data\_dir directory

            os.makedirs(raw\_data\_dir,exist\_ok=True)

        logging.info(f"Extracting the tgz\_file : [{tgz\_file\_path}] into the dir: [{raw\_data\_dir}]")

            """with tarfile.open(tgz\_file\_path) as income\_tgz\_file\_obj:

                income\_tgz\_file\_obj.extractall(path=raw\_data\_dir)"""

            shutil.copy(tgz\_file\_path, raw\_data\_dir)

            logging.info(f"Extraction completed successfully")

Split\_data\_as\_train\_test:

    def split\_data\_as\_train\_test(self):

        try:

            raw\_data\_dir = self.data\_ingestion\_config.raw\_data\_dir

            #The below cmnd will fetch all filenames in raw\_data\_dir and return the first file name from the list

            file\_name = os.listdir(raw\_data\_dir)[0]

            income\_file\_path = os.path.join(raw\_data\_dir,file\_name)

            income\_data\_frame = pd.read\_csv(income\_file\_path)

            income\_data\_frame["age\_cat"]=pd.cut(income\_data\_frame['age'],bins=[15,30,45,60,75,np.inf],

                                                labels=[1,2,3,4,5,6])

            strat\_train\_set = None

            strat\_test\_set = None

            split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)

            for train\_index, test\_index in split.split(income\_data\_frame,income\_data\_frame['age\_cat']):

                strat\_train\_set = income\_data\_frame.loc[train\_index].drop("age\_cat",axis=1)

                strat\_test\_set = income\_data\_frame.loc[test\_index].drop("age\_cat",axis=1)

                train\_file\_path = os.path.join(self.data\_ingestion\_config.ingested\_train\_dir,file\_name)

                test\_file\_path = os.path.join(self.data\_ingestion\_config.ingested\_test\_dir,file\_name)

                if strat\_train\_set is not None:

                    os.makedirs(self.data\_ingestion\_config.ingested\_train\_dir)

                    logging.info(f"Exporting training dataset to file : [{train\_file\_path}]")

                    strat\_train\_set.to\_csv(train\_file\_path,index=False)

                if strat\_test\_set is not None:

                    os.makedirs(self.data\_ingestion\_config.ingested\_test\_dir)

                    logging.info(f"Exporting testing dataset to file : [{test\_file\_path}]")

                    strat\_test\_set.to\_csv(test\_file\_path,index=False)

                data\_ingestion\_artifact = DataIngestionArtifact(train\_file\_path=train\_file\_path,

                                                                test\_file\_path=test\_file\_path,

                                                                is\_ingested=True,

                                                                message=f"Data Ingestion completed successfully")

                logging.info(f"Data Ingestion Artifact : [{data\_ingestion\_artifact}]")

                return data\_ingestion\_artifact

        except Exception as e:

            raise IncomeException(e,sys) from e

Destructor:

def \_\_del\_\_(self) -> None:

    logging.info(f"{'='\*20}Data Ingestion log Completed. {'='\*20}")

1. Pipeline – Data Ingestion:

from income.config.configuration import Configuration

from income.entity.config\_entity import DataIngestionConfig

from income.entity.artifact\_entity import DataIngestionArtifact

from income.component.data\_ingestion import DataIngestion

from income.logger import logging

from income.exception import IncomeException

import os,sys

class Pipeline:

    def \_\_init\_\_(self, config: Configuration = Configuration()) -> None:

        try:

            self.config = config

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_ingestion(self) -> DataIngestionArtifact:

        try:

            data\_ingestion = DataIngestion(data\_ingestion\_config=self.config.get\_data\_ingestion\_config())

            return data\_ingestion.initiate\_data\_ingestion()

        except Exception as e:

            raise IncomeException(e,sys) from e

    def run\_pipeline(self):

        try:

            #Dataa Ingestion Piipeline

            data\_inegstion\_artifact = self.start\_data\_ingestion()

        except Exception as e:

            raise IncomeException(e,sys) from e

Data Ingestion Testing:

Create a demo.py file to test the entire data ingestion part as show below.

import os,sys

from income.logger import logging

from income.exception import IncomeException

from income.config.configuration import DataIngestionConfig

from income.entity.artifact\_entity import DataIngestionArtifact

from income.pipeline.pipeline import Pipeline

def main():

    try:

        pipeline = Pipeline()

        pipeline.run\_pipeline()

    except Exception as e:

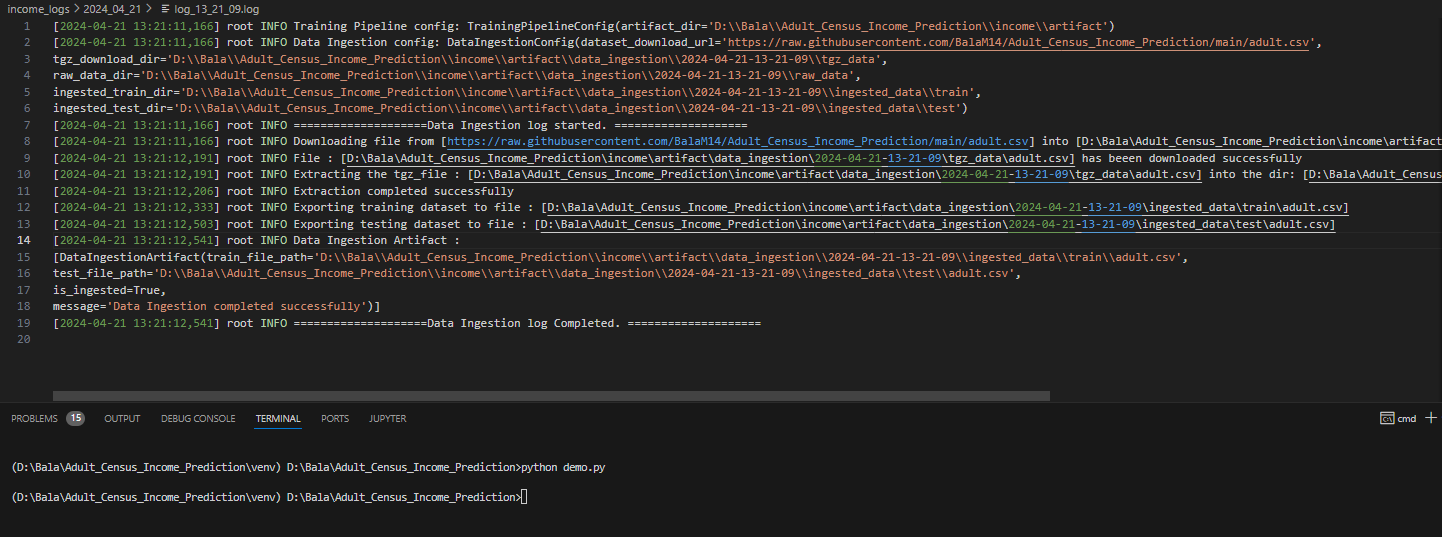
        logging.error(f"{e}")

        raise IncomeException(e,sys)

if \_\_name\_\_=="\_\_main\_\_":

    main()

Test Output:



1. Component - Data Validation:

Create a schema.yaml file to declare the column names, column datatypes, numerical/categorical columns & unique values of categorical columns.



The config.py under the entity folder, configuration.py under the config folder & Constant file – all 3 iss already updated. So we need to cross verify whether everything is right.

Component - Data Validation:

import os,sys

from income.logger import logging

from income.exception import IncomeException

from income.entity.config\_entity import DataValidationConfig

from income.constant import \*

from income.entity.artifact\_entity import DataIngestionArtifact

import pandas as pd

class DataVaidation:

    def \_\_init\_\_(self, data\_validation\_config:DataValidationConfig,data\_ingestion\_artifact:DataIngestionArtifact):

        try:

            logging.info(f"{'='\*20}Data Validation log started. {'='\*20}")

            self.data\_validation\_config = data\_validation\_config

            self.data\_ingestion\_artifact = data\_ingestion\_artifact

        except Exception as e:

            raise IncomeException(e,sys) from e

    def is\_train\_test\_file\_exists(self) -> bool:

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def vaidate\_datatset\_schema(self) -> bool:

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def initiate\_data\_validation(self) -> DataIngestionArtifact:

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def \_\_del\_\_(self) -> None:

        logging.info(f"{'='\*20}Data Validation log Completed. {'='\*20}")

Train\_Test\_file\_exists Method:

def is\_train\_test\_file\_exists(self) -> bool:

        try:

            logging.info(f"Checking if train & test data is available")

            is\_train\_file\_exist = False

            is\_test\_file\_exist = False

            train\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

            test\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

            is\_train\_file\_exist = os.path.exists(train\_file\_path)

            is\_test\_file\_exist = os.path.exists(test\_file\_path)

            is\_available = is\_train\_file\_exist and is\_test\_file\_exist

            logging.info(f"Is train & test file exists ? {is\_available}")

            if not is\_available:

                taining\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

                testing\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

                message = f"Training file : {taining\_file\_path} or Testing file : {testing\_file\_path} is not present"

                logging.info(message)

                raise Exception(message)

            return is\_available

        except Exception as e:

            raise IncomeException(e,sys) from e

Validate\_dataset\_schema Method:

def vaidate\_datatset\_schema(self) -> bool:

        try:

            validation\_status = False

            schema = self.data\_validation\_config.schema\_file\_path

            train\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

            test\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

            train\_df = pd.read\_csv(train\_file\_path)

            test\_df = pd.read\_csv(test\_file\_path)

            #check the number of columns

            if len(schema['columns']) != len(train\_df.columns) or len(schema['columns']) != len(test\_df.columns):

                message = f"The count of dataset columns were mis-matching"

                logging.info(message)

                raise Exception(message)

            #Check the column names are valid

            for i in schema['columns'].keys():

                if i not in train\_df.columns or i not in test\_df.columns:

                    message = f"The Column name [{i}] is not available in the dataset "

                    logging.info(message)

                    raise Exception(message)

            #Check the data type of the columns

            for i in schema['columns'].keys():

                if schema['columns'][i] != train\_df[i].dtype or schema['columns'][i] != test\_df[i].dtype:

                    message = f"The Column name [{i}] is having an invalid datatype in the dataset "

                    logging.info(message)

                    raise Exception(message)

            #check the categorical column eligible values

            for i in schema['domain\_value'].keys():

                unique\_values=[j.strip() for j in train\_df[i].unique()]

                for k in unique\_values:

                    if k not in schema['domain\_value'][i]:

                        message = f"The column name [{i}] is having an invalid value"

                        logging.info(message)

                        raise Exception(message)

            validation\_status = True

            message = f"The datasets of [{train\_df}] & [{test\_df}] are validated successfully"

            logging.info(message)

            return validation\_status

        except Exception as e:

            raise IncomeException(e,sys) from e

* DATA DRIFT:

Data drift refers to changes in the distribution of the features an ML model receives in production, potentially causing a decline in model performance.

Imports:

from evidently.model\_profile import Profile

from evidently.model\_profile.sections import DataDriftProfileSection

from evidently.dashboard import Dashboard

from evidently.dashboard.tabs import DataDriftTab

import json

def get\_and\_save\_data\_drift\_report(self):

    try:

        profile = Profile(sections=[DataDriftProfileSection()])

        train\_df,test\_df = self.get\_train\_test\_df()

        profile.calculate(train\_df,test\_df)

        report = json.loads(profile.json())

        with open(self.data\_validation\_config.report\_file\_path,'w') as report\_file:

            json.dump(report, report\_file, indent=6)

        return report

    except Exception as e:

        raise IncomeException(e,sys) from e

def save\_data\_drift\_report\_page(self):

    try:

        dashboard = Dashboard(tabs=[DataDriftTab()])

        train\_df,test\_df = self.get\_train\_test\_df()

        dashboard.calculate(train\_df,test\_df)

        dashboard.save(self.data\_validation\_config.report\_page\_file\_path)

    except Exception as e:

        raise IncomeException(e,sys) from e

def is\_data\_drift\_found(self) -> bool:

    try:

        report = self.get\_and\_save\_data\_drift\_report()

        self.save\_data\_drift\_report\_page()

        return True

    except Exception as e:

        raise IncomeException(e,sys) from e

Complete code of Data Validation Component:

import os,sys

from income.logger import logging

from income.exception import IncomeException

from income.entity.config\_entity import DataValidationConfig

from income.constant import \*

from income.entity.artifact\_entity import DataIngestionArtifact,DataValidationArtifact

from income.util.util import read\_yaml\_file

import pandas as pd

from evidently.model\_profile import Profile

from evidently.model\_profile.sections import DataDriftProfileSection

from evidently.dashboard import Dashboard

from evidently.dashboard.tabs import DataDriftTab

import json

class DataValidation:

    def \_\_init\_\_(self, data\_validation\_config:DataValidationConfig,data\_ingestion\_artifact:DataIngestionArtifact):

        try:

            logging.info(f"{'='\*20}Data Validation log started. {'='\*20}")

            self.data\_validation\_config = data\_validation\_config

            self.data\_ingestion\_artifact = data\_ingestion\_artifact

        except Exception as e:

            raise IncomeException(e,sys) from e

    def is\_train\_test\_file\_exists(self) -> bool:

        try:

            logging.info(f"Checking if train & test data is available")

            is\_train\_file\_exist = False

            is\_test\_file\_exist = False

            train\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

            test\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

            is\_train\_file\_exist = os.path.exists(train\_file\_path)

            is\_test\_file\_exist = os.path.exists(test\_file\_path)

            is\_available = is\_train\_file\_exist and is\_test\_file\_exist

            logging.info(f"Is train & test file exists ? {is\_available}")

            if not is\_available:

                taining\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

                testing\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

                message = f"Training file : {taining\_file\_path} or Testing file : {testing\_file\_path} is not present"

                logging.info(message)

                raise Exception(message)

            return is\_available

        except Exception as e:

            raise IncomeException(e,sys) from e

    def get\_train\_test\_df(self):

        try:

            train\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

            test\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

            train\_df = pd.read\_csv(train\_file\_path)

            test\_df = pd.read\_csv(test\_file\_path)

            return train\_df,test\_df

        except Exception as e:

            raise IncomeException(e,sys) from e

    def vaidate\_datatset\_schema(self) -> bool:

        try:

            validation\_status = False

            schema\_file\_path = self.data\_validation\_config.schema\_file\_path

            schema = read\_yaml\_file(file\_path=schema\_file\_path)

            train\_df,test\_df = self.get\_train\_test\_df()

            #check the number of columns

            if len(schema['columns']) != len(train\_df.columns) or len(schema['columns']) != len(test\_df.columns):

                message = f"The count of dataset columns were mis-matching"

                logging.info(message)

                raise Exception(message)

            #Check the column names are valid

            for i in schema['columns'].keys():

                if i not in train\_df.columns or i not in test\_df.columns:

                    message = f"The Column name [{i}] is not available in the dataset "

                    logging.info(message)

                    raise Exception(message)

            #Check the data type of the columns

            for i in schema['columns'].keys():

                if schema['columns'][i] != train\_df[i].dtype or schema['columns'][i] != test\_df[i].dtype:

                    message = f"The Column name [{i}] is having an invalid datatype in the dataset "

                    logging.info(message)

                    raise Exception(message)

            #check the categorical column eligible values

            """for i in schema['domain\_value'].keys():

                unique\_values=[j.strip() for j in train\_df[i].unique()]

                for k in unique\_values:

                    if k not in schema['domain\_value'][i]:

                        message = f"The column name [{i}] is having an invalid value"

                        logging.info(message)

                        raise Exception(message)"""

            validation\_status = True

            message = f"The datasets of train & test data are validated successfully"

            logging.info(message)

            return validation\_status

        except Exception as e:

            raise IncomeException(e,sys) from e

    def get\_and\_save\_data\_drift\_report(self):

        try:

            profile = Profile(sections=[DataDriftProfileSection()])

            train\_df,test\_df = self.get\_train\_test\_df()

            profile.calculate(train\_df,test\_df)

            report = json.loads(profile.json())

            report\_file\_path = self.data\_validation\_config.report\_file\_path

            report\_dir = os.path.dirname(report\_file\_path)

            os.makedirs(report\_dir,exist\_ok=True)

            with open(report\_file\_path,'w') as report\_file:

                json.dump(report, report\_file, indent=6)

            return report

        except Exception as e:

            raise IncomeException(e,sys) from e

    def save\_data\_drift\_report\_page(self):

        try:

            dashboard = Dashboard(tabs=[DataDriftTab()])

            train\_df,test\_df = self.get\_train\_test\_df()

            dashboard.calculate(train\_df,test\_df)

            report\_page\_file\_path = self.data\_validation\_config.report\_page\_file\_path

            report\_dir = os.path.dirname(report\_page\_file\_path)

            os.makedirs(report\_dir,exist\_ok=True)

            dashboard.save(report\_page\_file\_path)

        except Exception as e:

            raise IncomeException(e,sys) from e

    def is\_data\_drift\_found(self) -> bool:

        try:

            report = self.get\_and\_save\_data\_drift\_report()

            self.save\_data\_drift\_report\_page()

            return True

        except Exception as e:

            raise IncomeException(e,sys) from e

    def initiate\_data\_validation(self) -> DataIngestionArtifact:

        try:

            self.is\_train\_test\_file\_exists()

            self.vaidate\_datatset\_schema()

            self.is\_data\_drift\_found()

            data\_validation\_artifact = DataValidationArtifact(schema\_file\_path=self.data\_validation\_config.schema\_file\_path,

                                                              report\_file\_path=self.data\_validation\_config.report\_file\_path,

                                                              report\_page\_file\_path=self.data\_validation\_config.report\_page\_file\_path,

                                                              is\_validated=True,

                                                              message=f"Data Validation Performed successfully")

            logging.info(f"Data Validation artifact: {data\_validation\_artifact}")

            return data\_validation\_artifact

        except Exception as e:

            raise IncomeException(e,sys) from e

    def \_\_del\_\_(self) -> None:

        logging.info(f"{'='\*20}Data Validation log Completed. {'='\*20}")

1. Pipeline – Data validation:

from income.config.configuration import Configuration

from income.entity.config\_entity import DataIngestionConfig

from income.entity.artifact\_entity import DataIngestionArtifact

from income.component.data\_ingestion import DataIngestion

from income.component.data\_validation import DataValidation

from income.entity.artifact\_entity import DataValidationArtifact

from income.logger import logging

from income.exception import IncomeException

import os,sys

class Pipeline:

    def \_\_init\_\_(self, config: Configuration = Configuration()) -> None:

        try:

            self.config = config

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_ingestion(self) -> DataIngestionArtifact:

        try:

            data\_ingestion = DataIngestion(data\_ingestion\_config=self.config.get\_data\_ingestion\_config())

            return data\_ingestion.initiate\_data\_ingestion()

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_validation(self,data\_ingestion\_artifact: DataIngestionArtifact) -> DataValidationArtifact:

        try:

            data\_validation = DataValidation(data\_validation\_config=self.config.get\_data\_validation\_config(),

                                             data\_ingestion\_artifact=data\_ingestion\_artifact)

            return data\_validation.initiate\_data\_validation()

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_transformation(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_model\_training(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_model\_evaluation(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_model\_pusher(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def run\_pipeline(self):

        try:

            #Data Ingestion Piipeline

            data\_inegstion\_artifact = self.start\_data\_ingestion()

            data\_validation\_artifact = self.start\_data\_validation(data\_ingestion\_artifact=data\_inegstion\_artifact)

        except Exception as e:

            raise IncomeException(e,sys) from e

Test Code:

import os,sys

from income.logger import logging

from income.exception import IncomeException

from income.config.configuration import DataIngestionConfig

from income.entity.artifact\_entity import DataIngestionArtifact

from income.pipeline.pipeline import Pipeline

from income.config.configuration import Configuration

def main():

    try:

        pipeline = Pipeline()

        pipeline.run\_pipeline()

        #data\_validation\_config = Configuration().get\_data\_validation\_config()

        #print(data\_validation\_config)

    except Exception as e:

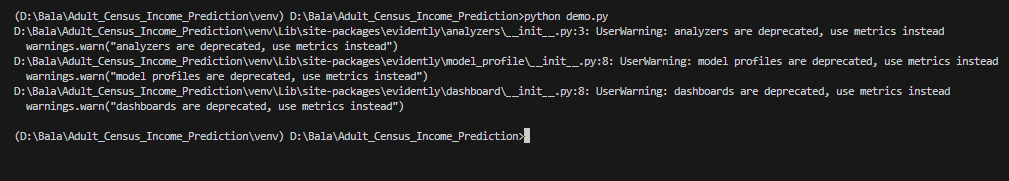
        logging.error(f"{e}")

        raise IncomeException(e,sys)

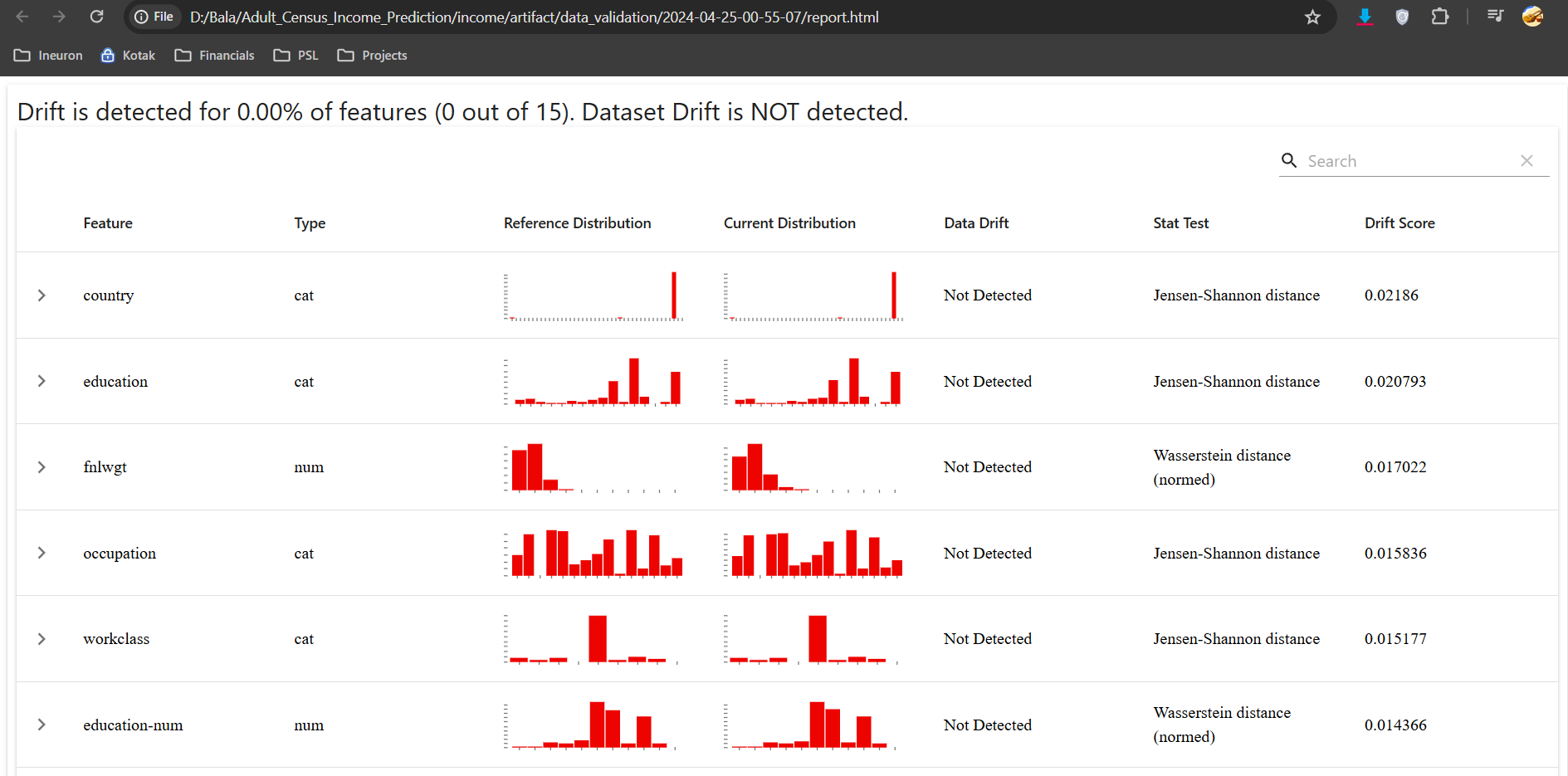
if \_\_name\_\_=="\_\_main\_\_":

    main()

Terminal:



Data Drift Report:



1. Component – Data Transformation:

from income.entity.config\_entity import DataTransformationConfig

from income.entity.artifact\_entity import DataIngestionArtifact,DataValidationArtifact

from income.logger import logging

from income.exception import IncomeException

import os,sys

import numpy as np

import pandas as pd

from income.util.util import read\_yaml\_file, save\_numpy\_array\_data, save\_object

from income.constant import \*

#from sklearn.base import BaseEstimator,TransformerMixin

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

class DataTransformation:

    def \_\_init\_\_(self,data\_transformation\_config: DataTransformationConfig,

                 data\_ingeestion\_artifact: DataIngestionArtifact,

                 data\_validation\_artifact: DataValidationArtifact):

        try:

            self.data\_transformation\_config = data\_transformation\_config

            self.data\_ingestion\_artifact = data\_ingeestion\_artifact

            self.data\_validation\_artifact = data\_validation\_artifact

except Exception as e:

            raise IncomeException(e,sys) from e

    @staticmethod

    def load\_data(file\_path :str, schema\_file\_path) -> pd.DataFrame:

        '''

        This function converts the data types as per the schema file values.

        If it is not ablee to convert then it will rasie an exception'''

        try:

            datset\_schema = read\_yaml\_file(schema\_file\_path)

            schema = datset\_schema[DATASET\_SCHEMA\_COLUMNS\_KEY]

            dataframe = pd.read\_csv(file\_path)

            error\_messgae = ""

            for column in dataframe.columns:

                if column in list(schema.keys()):

                    dataframe[column].astype(schema[column])

                else:

                    error\_messgae = f"{error\_messgae} \nColumn: [{column}] is not in the schema."

            if len(error\_messgae) > 0:

                raise Exception(error\_messgae)

            return dataframe

        except Exception as e:

            raise IncomeException(e,sys) from e

    def get\_data\_\_transformer\_object(self) -> ColumnTransformer:

        try:

            schema\_file\_path = self.data\_validation\_artifact.schema\_file\_path

            dataset\_schema = read\_yaml\_file(schema\_file\_path)

            numerical\_columns = dataset\_schema[NUMERICAL\_COLUMN\_KEY]

            categorical\_columns = dataset\_schema[CATEGORICAL\_COLUMN\_KEY]

            numerical\_pipeline = Pipeline(steps=[

                ('impute', SimpleImputer(strategy='median')),

                ('scaler',StandardScaler())

            ])

            categorical\_pipeline = Pipeline(steps=[

                ('impute',SimpleImputer(strategy='most\_frequent')),

                ('one\_hot\_encoder',OneHotEncoder())

            ])

            logging.info(f"Numerical\_Columns : [{numerical\_columns}]")

            logging.info(f"Categorical\_Columns : [{categorical\_columns}]")

            preprocessing = ColumnTransformer([

                ('numerical\_pipeline',numerical\_pipeline,numerical\_columns),

                ('categorical\_pipeline',categorical\_pipeline,categorical\_columns)

            ])

            return preprocessing

        except Exception as e:

            raise IncomeException(e,sys) from e

Update Utils:

import yaml

from income.exception import IncomeException

import os,sys

import numpy as np

import dill

def read\_yaml\_file(file\_path:str) -> dict:

    """

    Reads a Yaml file and returns the contents as a dictionary.

    file\_path:str

    """

    try:

        with open(file\_path,"rb") as yaml\_file:

            return yaml.safe\_load(yaml\_file)

    except Exception as e:

        raise IncomeException(e,sys) from e

def save\_numpy\_array\_data(file\_path: str, array: np.array):

    '''

    save numpy array data to file

    file\_path: str location of file to save

    array: np.array data to save

    '''

    try:

        dir\_path = os.path.dirname(file\_path)

        os.makedirs(dir\_path,exist\_ok=True)

        with open(file\_path,"wb") as file\_obj:

            np.save(file\_obj,array)

    except Exception as e:

        raise IncomeException(e,sys) from e

def load\_numpy\_array\_data(file\_path: str) -> np.array:

    '''

    load numpy array data from file

    file\_path: str location of file to load

    return : np.array data loaded

    '''

    try:

       with open(file\_path,"rb") as file\_obj:

           return np.load(file\_obj)

    except Exception as e:

        raise IncomeException(e,sys) from e

#To save the object in the pickle format

def save\_object(file\_path: str, obj):

    '''

    file\_path: str location of file

    obj: Any sort of obj

    '''

    try:

        dir\_path = os.path.dirname(file\_path)

        os.makedirs(dir\_path,exist\_ok=True)

        with open(file\_path,"wb") as file\_obj:

            dill.dump(obj, file\_obj)

    except Exception as e:

        raise IncomeException(e,sys) from e

#To load the pickle object

def load\_object(file\_path: str):

    '''

    file\_path: str location of file

    '''

    try:

       with open(file\_path,"rb") as file\_obj:

           return dill.load(file\_obj)

    except Exception as e:

        raise IncomeException(e,sys) from e

Complete Code of DataTransformation Component:

from income.entity.config\_entity import DataTransformationConfig

from income.entity.artifact\_entity import DataIngestionArtifact,DataValidationArtifact,DataTransformationArtifact

from income.logger import logging

from income.exception import IncomeException

import os,sys

import numpy as np

import pandas as pd

from income.util.util import read\_yaml\_file, save\_numpy\_array\_data, save\_object

from income.constant import \*

#from sklearn.base import BaseEstimator,TransformerMixin

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

class DataTransformation:

    def \_\_init\_\_(self,data\_transformation\_config: DataTransformationConfig,

                 data\_ingeestion\_artifact: DataIngestionArtifact,

                 data\_validation\_artifact: DataValidationArtifact):

        try:

            self.data\_transformation\_config = data\_transformation\_config

            self.data\_ingestion\_artifact = data\_ingeestion\_artifact

            self.data\_validation\_artifact = data\_validation\_artifact

        except Exception as e:

            raise IncomeException(e,sys) from e

    @staticmethod

    def load\_data(file\_path :str, schema\_file\_path) -> pd.DataFrame:

        '''

        This function converts the data types as per the schema file values.

        If it is not ablee to convert then it will rasie an exception'''

        try:

            datset\_schema = read\_yaml\_file(schema\_file\_path)

            schema = datset\_schema[DATASET\_SCHEMA\_COLUMNS\_KEY]

            dataframe = pd.read\_csv(file\_path)

            error\_messgae = ""

            for column in dataframe.columns:

                if column in list(schema.keys()):

                    dataframe[column].astype(schema[column])

                else:

                    error\_messgae = f"{error\_messgae} \nColumn: [{column}] is not in the schema."

            if len(error\_messgae) > 0:

                raise Exception(error\_messgae)

            return dataframe

        except Exception as e:

            raise IncomeException(e,sys) from e

    def get\_data\_\_transformer\_object(self) -> ColumnTransformer:

        try:

            schema\_file\_path = self.data\_validation\_artifact.schema\_file\_path

            dataset\_schema = read\_yaml\_file(schema\_file\_path)

            numerical\_columns = dataset\_schema[NUMERICAL\_COLUMN\_KEY]

            categorical\_columns = dataset\_schema[CATEGORICAL\_COLUMN\_KEY]

            numerical\_pipeline = Pipeline(steps=[

                ('impute', SimpleImputer(strategy='median')),

                ('scaler',StandardScaler())

            ])

            categorical\_pipeline = Pipeline(steps=[

                ('impute',SimpleImputer(strategy='most\_frequent')),

                ('one\_hot\_encoder',OneHotEncoder())

            ])

            logging.info(f"Numerical\_Columns : [{numerical\_columns}]")

            logging.info(f"Categorical\_Columns : [{categorical\_columns}]")

            preprocessing = ColumnTransformer([

                ('numerical\_pipeline',numerical\_pipeline,numerical\_columns),

                ('categorical\_pipeline',categorical\_pipeline,categorical\_columns)

            ])

            return preprocessing

        except Exception as e:

            raise IncomeException(e,sys) from e

    def initiate\_data\_transformation(self) -> DataTransformationArtifact:

        try:

            logging.info(f"Obtaining preprocessing object.")

            preprocessing\_obj = self.get\_data\_\_transformer\_object()

            logging.info(f"Obtaining training and test file path.")

            train\_file\_path = self.data\_ingestion\_artifact.train\_file\_path

            test\_file\_path = self.data\_ingestion\_artifact.test\_file\_path

            schema\_file\_path = self.data\_validation\_artifact.schema\_file\_path

            logging.info(f"Loading training and test data as pandas dataframe.")

            train\_df = self.load\_data(train\_file\_path,schema\_file\_path=schema\_file\_path)

            test\_df = self.load\_data(test\_file\_path,schema\_file\_path=schema\_file\_path)

            schema = read\_yaml\_file(schema\_file\_path)

            target\_column\_name = schema[TARGET\_COLUMN\_KEY]

            logging.info(f"Splitting input and target feature from training and testing dataframe.")

            input\_feature\_train\_df = train\_df.drop(columns=[target\_column\_name],axis=1)

            target\_fetaure\_train\_df = train\_df[target\_column\_name]

            input\_feature\_test\_df = test\_df.drop(columns=[target\_column\_name],axis=1)

            target\_fetaure\_test\_df = test\_df[target\_column\_name]

            #For training we need to use fit\_tramsform & for testing we use only the transform

            logging.info(f"Applying preprocessing object on training dataframe and testing dataframe")

            input\_feature\_train\_array = preprocessing\_obj.fit\_transform(input\_feature\_train\_df)

            input\_feature\_test\_array = preprocessing\_obj.transform(input\_feature\_test\_df)

            train\_arr = np.c\_[ input\_feature\_train\_array, np.array(target\_fetaure\_train\_df)]

            test\_arr = np.c\_[input\_feature\_test\_array,np.array(target\_fetaure\_test\_df)]

            transformed\_train\_dir = self.data\_transformation\_config.transformed\_train\_dir

            transformed\_test\_dir = self.data\_transformation\_config.transformed\_test\_dir

            train\_file\_name = os.path.basename(train\_file\_path).replace(".csv",".npz")

            test\_file\_name = os.path.basename(test\_file\_path).replace(".csv",".npz")

            transformed\_train\_file\_path = os.path.join(transformed\_train\_dir,train\_file\_name)

            transformed\_test\_file\_path = os.path.join(transformed\_test\_dir,test\_file\_name)

            logging.info(f"Saving transformed training and testing array.")

            save\_numpy\_array\_data(file\_path=transformed\_train\_file\_path,array=train\_arr)

            save\_numpy\_array\_data(file\_path=transformed\_test\_file\_path,array=test\_arr)

            #Save the Preprocessing object in the pickle format to use it in test data

            preprocessing\_obj\_file\_path = self.data\_transformation\_config.preprocessed\_object\_file\_path

            logging.info(f"Saving preprocessing object.")

            save\_object(file\_path=preprocessing\_obj\_file\_path,obj=preprocessing\_obj)

            data\_transformation\_artifact = DataTransformationArtifact(

                tranformed\_train\_file\_path=transformed\_train\_file\_path,

                transformed\_test\_file\_path=transformed\_test\_file\_path,

                preprocessed\_object\_file\_path=preprocessing\_obj\_file\_path,

                is\_transfomred=True,

                message=f"Data transformation completed successfully")

            logging.info(f"Data tranformation Artifact : [{data\_transformation\_artifact}]")

            return data\_transformation\_artifact

        except Exception as e:

            raise IncomeException(e,sys) from e

    def \_\_del\_\_(self):

        logging.info(f"{'='\*20}Data Transformation log completed.{'='\*20} \n\n")

1. Pipeline – Data Transformation:

from income.config.configuration import Configuration

from income.component.data\_ingestion import DataIngestion

from income.component.data\_validation import DataValidation

from income.component.data\_transformation import DataTransformation

from income.entity.artifact\_entity import DataIngestionArtifact

from income.entity.artifact\_entity import DataValidationArtifact

from income.entity.artifact\_entity import DataTransformationArtifact

from income.logger import logging

from income.exception import IncomeException

import os,sys

class Pipeline:

    def \_\_init\_\_(self, config: Configuration = Configuration()) -> None:

        try:

            self.config = config

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_ingestion(self) -> DataIngestionArtifact:

        try:

            data\_ingestion = DataIngestion(

                data\_ingestion\_config=self.config.get\_data\_ingestion\_config()

                )

            return data\_ingestion.initiate\_data\_ingestion()

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_validation(self,data\_ingestion\_artifact: DataIngestionArtifact) -> DataValidationArtifact:

        try:

            data\_validation = DataValidation(

                data\_validation\_config=self.config.get\_data\_validation\_config(),

                data\_ingestion\_artifact=data\_ingestion\_artifact

                )

            return data\_validation.initiate\_data\_validation()

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_data\_transformation(self,data\_ingestion\_artifact: DataIngestionArtifact,

                                   data\_validation\_artifact: DataValidationArtifact) -> DataTransformationArtifact:

        try:

            data\_transformation = DataTransformation(

                data\_transformation\_config=self.config.get\_data\_transformation\_config(),

                data\_ingestion\_artifact=data\_ingestion\_artifact,

                data\_validation\_artifact=data\_validation\_artifact

                )

            return data\_transformation.initiate\_data\_transformation()

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_model\_training(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_model\_evaluation(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def start\_model\_pusher(self):

        try:

            pass

        except Exception as e:

            raise IncomeException(e,sys) from e

    def run\_pipeline(self):

        try:

            #Data Ingestion Piipeline

            data\_inegstion\_artifact = self.start\_data\_ingestion()

            data\_validation\_artifact = self.start\_data\_validation(data\_ingestion\_artifact=data\_inegstion\_artifact)

            data\_transformation\_artifact = self.start\_data\_transformation(

                                                data\_ingestion\_artifact=data\_inegstion\_artifact,

                                                data\_validation\_artifact=data\_validation\_artifact

                                                )

        except Exception as e:

            raise IncomeException(e,sys) from e