Prerequisite :

Open the necessary folder in VSC

$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

1. AnacondaPrompt -> Specify Path -> **code .**
2. VSC -> terminal -> command Prompt -> **conda create -p venv python==3.8 -y**
3. Clone repository and syncing with Githib [refer after creating new repository]
   1. **git init** and in case /// **echo "# git-practice" >> README.txt and git add README.txt**
   2. **git add README.md**
   3. **git commit -m "first commit"**
   4. **git status**
   5. **cls**
   6. **git branch -M main**
   7. **git remote add origin <https://github.com/MUTHUSELVANV/ML-Project.git>**
   8. **git remote -v** [displays the URLs of the remote repositories associated with]
   9. git push -u origin main
4. Github -> Add file -> +Create New File -> .gitignore -> template = Python -> commit
5. VSC -> **cls** -> **git pull**
6. VSC -> inside **venv** create **setup.py** and **requirements.txt**
7. **Setup.py**

from setuptools import find\_packages, setup

from typing import List

HYPEN\_E\_DOT='-e .'

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

    '''

    this function will return the list of requirements

    '''

    requirements=[]

    with open(file\_path) as file\_obj:

        requirements=file\_obj.readlines()

        requirements=[req.replace("\n", "") for req in requirements]

        if HYPEN\_E\_DOT in requirements:

            requirements.remove(HYPEN\_E\_DOT)

    return requirements

setup

name='mlproject',

version='0.0.1',

author= 'Muthu',

author\_email='muthu281099@gmail.com',

packages=find\_packages(),

install\_requires=get\_requirements('requirements.txt')

)

**8)requirements.txt**

pandas

numpy

seaborn

-e .

1. In **MLProject ->** create folder **SRC ->** creat file inside it called **\_\_init\_\_.py**

**\_\_init\_\_.py ->** the file in which it is present can be will be created as a package and also it can be exported to some other file location

1. Click requirements.txt and start a new terminal
   1. type **activate** (command prompt)
   2. conda activate venv/
   3. **pip install -r requirements.txt** [restart PC in case not able to proceed ]

Data ingestion is the process of importing large, assorted data files from multiple sources into a single, cloud-based storage medium—a data warehouse, data mart or database—where it can be accessed and analyzed.

Data transformation is the process of converting, cleansing, and structuring data into a usable format that can be analyzed to support decision making processes, and to propel the growth of an organization. Data transformation is used when data needs to be converted to match that of the destination system.

1. Create a folder name **‘components’** inside **‘src’** folder

Inside components: Gonna create all modules

* 1. Create file \_\_init\_\_.py
  2. Create file **‘data\_ingestion.py’**, **‘data\_transformation.py’** and **‘model\_trainer.py’**

1. Create a folder name **‘pipeline’** inside **‘src’** folder [there are 2 pipeline - training & Predicting pipeline]
   1. Create file \_\_init\_\_.py
   2. Create 2 files inside pipeline folder - **‘predict\_pipeline.py’** & **‘train\_pipeline.py’**
2. Create 3 important files inside src folder
   1. **‘exception.py’**
   2. **‘logger.py’**
   3. **‘utils,py’**

**‘exception.py’**

import sys

def error\_message\_detail(error,error\_detail:sys):

    \_,\_,exc\_tb=error\_detail.exc\_info()

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

    error\_message="Error occured in python script name [{0}] line number [{1}] error message[{2}]”.format(

    file\_name, exc\_tb.tb\_lineno,str(error))

    return error\_message

class CustomException(Exception):

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

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

        self.error\_message=error\_message\_detail(error\_message,error\_detail=error\_detail)

    def \_\_str\_\_(self):

        return self.error\_message

**CHECK ‘CUSTOM EXCEPTIONAL HANDLING DOCUMENTATION IN PYTHON’**

**‘Logger.py’**

import logging

import os

from datetime import datetime

LOG\_FILE=f"{datetime.now().strftime('%m\_%d\_%Y\_%H\_%M\_%S')}.log"

logs\_path=os.path.join(os.getcwd(),"logs", LOG\_FILE)

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

LOG\_FILE\_PATH=os.path.join(logs\_path,LOG\_FILE)

logging.basicConfig(

    filename=LOG\_FILE\_PATH,

    format="[ %(asctime)s ] %(lineno)d %(name)s - %(levelname)s - %(message)s", level=logging.INFO,

)

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

    logging.info("LOGGING has started")

Then cmd -> python src/logger.py

1. Modify exception.py by adding :

sys.path.append("D:\Muthuselvan\MLProject")

from src.logger import logging

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

    try:

        a=1/0

    except Exception as e:

        logging.info("Divide by zero")

        raise CustomException(e,sys)

The Reason: from src.logger import logging -> to update the error in logging tab

Check log file once to verify error has updated there [INFO DIVIDE BY ZERO]

cmd -> python src/exception.py [you will get:

File "src/exception.py", line 25, in <module>

raise CustomException(e,sys)

\_\_main\_\_.CustomException: Error occured in python script name [src/exception.py] line number [22] error message[division by zero]]

1. Remove if \_\_name\_\_=="\_\_main\_\_": in both exception.py & logger.py
2. cmd->
   1. git status
   2. git add .
   3. git commit -m "logging and exception"
   4. git push -u origin main
3. Cmd-> python src/exception.py # Simply checking
4. Added Jupyter Notebook from Github
5. Install Python kernal
6. Go to venv/requirements and update any package which we are going to use.
   1. Also #-e . because we don’t want to build the package again and again
7. After debugging the code, run all the code in jupyter notebook
8. After running all codes 1 . EDA STUDENT PERFORMANCE.ipynb, go to 2. MODEL TRAINING.ipynb
   1. It will show error since we have to install scikit package
   2. Go to requirements.txt and add scikit-learn, catboost, xgboost
   3. Cmd -> pip install -r requirements.txt

22)

* 1. git status
  2. git add .
  3. git commit -m "EDA and problem statement"
  4. git push -u origin main

1. src/components/data\_ingestion.py

**data\_ingestion.py**

import os

import sys

sys.path.append("D:\Muthuselvan\MLProject")

from src.exception import CustomException

from src.logger import logging

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from dataclasses import dataclass

@dataclass

class DataIngestionConfig:

    train\_data\_path: str=os.path.join('artifacts', "train.csv")

    test\_data\_path: str=os.path.join('artifacts',"test.csv")

    raw\_data\_path: str=os.path.join('artifacts',"data.csv")

class DataIngestion:

    def \_\_init\_\_(self):

        self.ingestion\_config=DataIngestionConfig()

    def initiate\_data\_ingestion(self):

        logging.info("Entered the data ingestion method or component")

        try:

            df=pd.read\_csv('notebook\data\stud.csv')

            logging.info('Read the dataset as dataframe')

            os.makedirs (os.path.dirname(self.ingestion\_config.train\_data\_path), exist\_ok=True)

            df.to\_csv(self.ingestion\_config.raw\_data\_path, index=False, header=True)

            logging.info("Train test split initiated")

            train\_set, test\_set=train\_test\_split(df, test\_size=0.2, random\_state=42)

            train\_set.to\_csv(self.ingestion\_config.train\_data\_path, index=False, header=True)

            test\_set.to\_csv(self.ingestion\_config.test\_data\_path, index=False, header=True)

            logging.info("Ingestion of the data is completed")

            return(

            self.ingestion\_config.train\_data\_path,

            self.ingestion\_config.test\_data\_path

            )

        except Exception as e:

            raise CustomException (e, sys)

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

    obj=DataIngestion ()

    obj.initiate\_data\_ingestion()

CMD-> python src/components/data\_ingestion.py

1. Inside .gitignore file -> Inside #Environments add **.artifacts**

24)

* 1. git add .
  2. git commit -m "Data Ingestion"
  3. git status
  4. git push -u origin main

1. Go to requirements.txt and add **dill**

CMD -> pip install -r requirements.txt

25)

**data\_transformation.py**

import sys

from dataclasses import dataclass

import numpy as np

import pandas as pd

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from src.exception import CustomException

from src.logger import logging

import os

from src.utils import save\_object

@dataclass

class DataTransformationConfig:

    preprocessor\_obj\_file\_path=os.path.join('artifacts', "proprocessor.pkl")

class DataTransformation:

    def \_\_init\_\_(self):

         self.data\_transformation\_config=DataTransformationConfig()

    def get\_data\_transformer\_object(self):

        '''

        This function si responsible for data trnasformation

        '''

        try:

            numerical\_columns = ["writing\_score", "reading\_score"]

            categorical\_columns = [

            "gender",

            "race\_ethnicity",

            "parental\_level\_of\_education",

            "lunch",

            "test\_preparation\_course",

            ]

            num\_pipeline= Pipeline(

                steps=[

                ("imputer", SimpleImputer(stategy="median")),

                ("scaler", StandardScaler(with\_mean=False))

                ]

            )

            cat\_pipeline=Pipeline(

                steps=[

                ("imputer", SimpleImputer (strategy="most\_frequent")),

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

                ("scaler", StandardScaler(with\_mean=False))

                ]

            )

            logging.info("Numerical columns standard scaling completed")

            logging.info("Categorical columns encoding completed")

            preprocessor=ColumnTransformer (

                [

                ("num\_pipeline", num\_pipeline, numerical\_columns),

                ("cat\_pipelines", cat\_pipeline, categorical\_columns)

                ]

            )

            return preprocessor

        except Exception as e:

            raise CustomException(e, sys)

    def initiate\_data\_transformation (self, train\_path, test\_path):

        try:

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

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

            logging.info("Read train and test data completed")

            logging.info("Obtaining preprocessing object")

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

            target\_column\_name="math\_score"

            numerical\_columns = ["writing\_score", "reading\_score"]

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

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

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

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

            logging.info(

                f"Applying preprocessing object on training dataframe and testing dataframe."

            )

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

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

            train\_arr = np.c\_[

                input\_feature\_train\_arr, np.array(target\_feature\_train\_df)

            ]

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

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

            save\_object(

                file\_path=self.data\_transformation\_config.preprocessor\_obj\_file\_path,

                obj=preprocessing\_obj

            )

            return (

                train\_arr,

                test\_arr,

                self.data\_transformation\_config.preprocessor\_obj\_file\_path,

            )

except Exception as e:

            raise CustomException(e,sys)

26)

**utils.py**

import os

import sys

import numpy as np

import pandas as pd

import dill

sys.path.append("D:\Muthuselvan\MLProject")

from src.exception import CustomException

def save\_object(file\_path, 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 CustomException(e, sys)

1. Edit **data\_ingestion.py**

from src.components.data\_transformation import DataTransformation

from src.components.data\_transformation import DataTransformationConfig

**AND**

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

    obj=DataIngestion()

    train\_data,test\_data=obj.initiate\_data\_ingestion()

    data\_transformation=DataTransformation()

    data\_transformation.initiate\_data\_transformation(train\_data, test\_data)

CMD->python src/components/data\_ingestion.py

* 1. git add .
  2. git commit -m "Data Transformation Done"
  3. git push -u origin main
  4. git status

**MODEL TRAINING**

**model\_trainer.py**

import os

import sys

from dataclasses import dataclass

from catboost import CatBoostRegressor

from sklearn.ensemble import (

    AdaBoostRegressor,

    GradientBoostingRegressor,

    RandomForestRegressor,

)

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor

from xgboost import XGBRegressor

from src.exception import CustomException

from src.logger import logging

from src.utils import save\_object, evaluate\_models

@dataclass

class ModelTrainerConfig:

    trained\_model\_file\_path=os.path.join("artifacts", "model.pkl")

class ModelTrainer:

    def \_\_init\_\_(self):

        self.model\_trainer\_config=ModelTrainerConfig()

    def initiate\_model\_trainer(self, train\_array, test\_array):

        try:

            logging.info("Split training and test input data")

            X\_train,y\_train, X\_test,y\_test=(

            train\_array[:,:-1],

            train\_array[:,-1],

            test\_array[:,:-1],

            test\_array[:,-1]

            )

            models = {

            "Random Forest": RandomForestRegressor(),

            "Decision Tree": DecisionTreeRegressor(),

            "Gradient Boosting": GradientBoostingRegressor(),

            "Linear Regression": LinearRegression(),

            "K-Neighbors Classifier": KNeighborsRegressor(),

            "XGBClassifier": XGBRegressor(),

            "CatBoosting Classifier": CatBoostRegressor(verbose=False),

            "AdaBoost Classifier": AdaBoostRegressor(),

            }

            model\_report:dict=evaluate\_models(X\_train=X\_train,y\_train=y\_train,X\_test=X\_test,y\_test=y\_test,

                                             models=models)

            ## To get best model score from dict

            best\_model\_score = max(sorted (model\_report.values()))

            ## To get best model name from dict

            best\_model\_name = list(model\_report.keys())[

                list (model\_report. values()).index (best\_model\_score)

            ]

            best\_model = models[best\_model\_name]

            if best\_model\_score<0.6:

                raise CustomException("No best model found")

            logging.info(f"Best found model on both training and testing dataset")

            save\_object(

                file\_path=self.model\_trainer\_config.trained\_model\_file\_path,

                obj=best\_model

            )

            predicted= best\_model.predict(X\_test)

            r2\_square = r2\_score(y\_test,predicted)

            return r2\_square

        except Exception as e:

            raise CustomException(e, sys)

**utils.py:**

from sklearn.metrics import r2\_score

def evaluate\_models(X\_train,y\_train, X\_test,y\_test,models):

    try:

        report = {}

        for i in range(len(list (models))):

            model = list(models.values())[i]

            model.fit(X\_train, y\_train) # Train model

            y\_train\_pred = model.predict(X\_train)

            y\_test\_pred = model.predict(X\_test)

            train\_model\_score = r2\_score(y\_train, y\_train\_pred)

            test\_model\_score = r2\_score(y\_test, y\_test\_pred)

            report[list (models.keys())[i]] = test\_model\_score

        return report

    except Exception as e:

        raise CustomException(e, sys)

**data\_ingestion.py**

from src.components.model\_trainer import ModelTrainerConfig

from src.components.model\_trainer import ModelTrainer

Update this:

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

    obj=DataIngestion ()

    train\_data,test\_data=obj.initiate\_data\_ingestion()

    data\_transformation=DataTransformation()

    train\_arr,test\_arr,\_ = data\_transformation.initiate\_data\_transformation(train\_data, test\_data)

    modeltrainer = ModelTrainer()

    print(modeltrainer.initiate\_model\_trainer(train\_arr,test\_arr))

CMD->python src/components/data\_ingestion.py

* 1. git add .
  2. git status

1. Remove **.artifacts** in .gitignore in #Environments
   1. git add .
   2. git commit -m "Model Trainer"
   3. git push -u origin main

**Hyper Parameter Tuning**

Model\_trainer.py

Add & Update

            models = {

                "Random Forest": RandomForestRegressor(),

                "Decision Tree": DecisionTreeRegressor(),

                "Gradient Boosting": GradientBoostingRegressor(),

                "Linear Regression": LinearRegression(),

                "XGBRegressor": XGBRegressor(),

                "CatBoosting Regressor": CatBoostRegressor(verbose=False),

                "AdaBoost Regressor": AdaBoostRegressor(),

            }

            params={

                "Decision Tree": {

                    'criterion':['squared\_error', 'friedman\_mse', 'absolute\_error', 'poisson'],

                    # 'splitter':['best','random'],

                    # 'max\_features':['sqrt','log2'],

                },

                "Random Forest":{

                    # 'criterion':['squared\_error', 'friedman\_mse', 'absolute\_error', 'poisson'],

                    # 'max\_features':['sqrt','log2',None],

                    'n\_estimators': [8,16,32,64,128,256]

                },

                "Gradient Boosting":{

                    # 'loss':['squared\_error', 'huber', 'absolute\_error', 'quantile'],

                    'learning\_rate':[.1,.01,.05,.001],

                    'subsample':[0.6,0.7,0.75,0.8,0.85,0.9],

                    # 'criterion':['squared\_error', 'friedman\_mse'],

                    # 'max\_features':['auto','sqrt','log2'],

                    'n\_estimators': [8,16,32,64,128,256]

                },

                "Linear Regression":{},

                "XGBRegressor":{

                    'learning\_rate':[.1,.01,.05,.001],

                    'n\_estimators': [8,16,32,64,128,256]

                },

                "CatBoosting Regressor":{

                    'depth': [6,8,10],

                    'learning\_rate': [0.01, 0.05, 0.1],

                    'iterations': [30, 50, 100]

                },

                "AdaBoost Regressor":{

                    'learning\_rate':[.1,.01,0.5,.001],

                    # 'loss':['linear','square','exponential'],

                    'n\_estimators': [8,16,32,64,128,256]

                }

            }

            model\_report:dict=evaluate\_models(X\_train=X\_train,y\_train=y\_train,X\_test=X\_test,y\_test=y\_test,models=models,param=params)

**utils.py**

Add & Update

from sklearn.model\_selection import GridSearchCV

def evaluate\_models(X\_train,y\_train, X\_test,y\_test,models,param):

    try:

        report = {}

        for i in range(len(list(models))):

            model = list(models.values())[i]

            para=param[list(models.keys())[i]]

            gs = GridSearchCV(model,para,cv=3)

            gs.fit(X\_train,y\_train)

            model.set\_params(\*\*gs.best\_params\_)

            model.fit(X\_train,y\_train)

            #model.fit(X\_train, y\_train) # Train model

            y\_train\_pred = model.predict(X\_train)

            y\_test\_pred = model.predict(X\_test)

            train\_model\_score = r2\_score(y\_train, y\_train\_pred)

            test\_model\_score = r2\_score(y\_test, y\_test\_pred)

            report[list (models.keys())[i]] = test\_model\_score

        return report

CMD->python src/components/data\_ingestion.py