Project flow:

Step1- setup github repository and create a folder where we want to store product

Step-2 in command prompt do A black screen with white text

Description automatically generated

Step-3 create a virtual environment





Step-4 – git init



Step-5 – create a readme fiile and add it in github

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A computer screen shot of a black screen

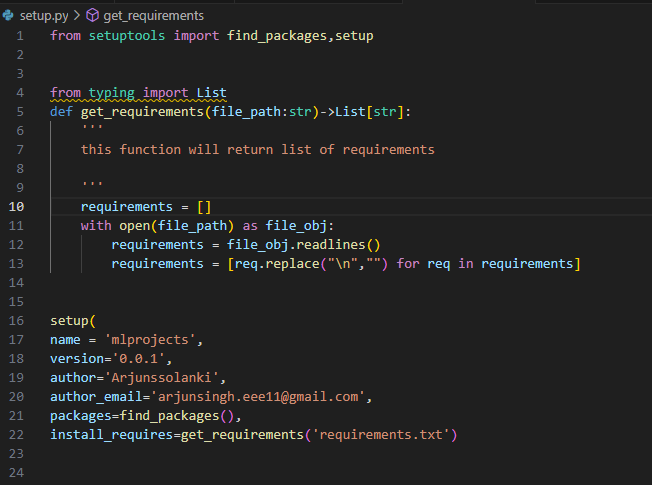
Description automatically generated

Step-6 create a requirements.txt and setup.py

A screen shot of a computer program

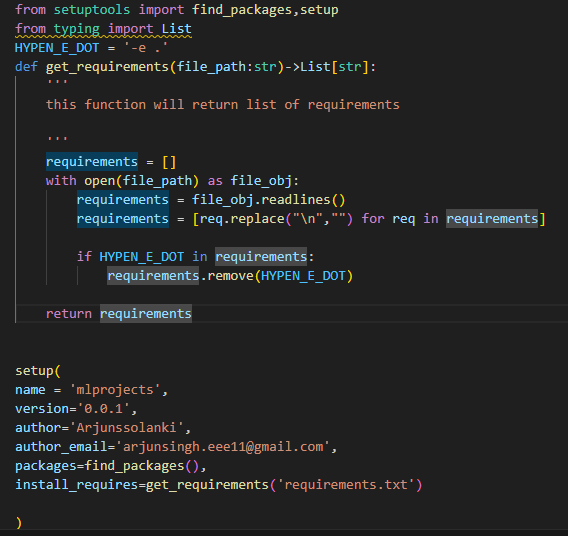
Description automatically generated

Modifiying setup.py



Step-7 src (source) is use to find out how many packages there are

create a source folder and put \_\_init\_\_.py in it

Step-8 modify the setup tool 

Then in terminal write pip install -r requirements.txt

Poject structure: maximum work is on src (source) folder

Step-1 create a folder name component ,in src with

\_\_init\_\_.py

Create data\_ingestion.py – consist the code like get the data set, split into train and test set

data\_tranformation.py- consist all the steps related to data transformation

model\_trainer.py – training code, different kinds of model etc

step 2- create a folder name pipeline cosisting in src

training\_pipeline.py – consist of training pipeline

prediction\_pipeline.py – consit of prediction pipeline

step-3 in src create a logger.py file and exception.py

1. Exception.py-

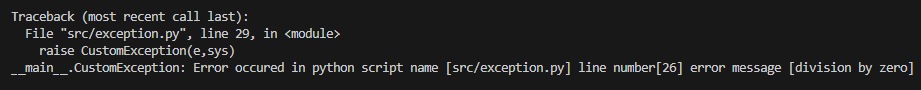
A screen shot of a computer program

Description automatically generated

Now run the file In the terminal

A black screen with white text

Description automatically generated



New log file is created

A black background with numbers and numbers

Description automatically generated

1. Logger.py –

A computer screen shot of text

Description automatically generated

Now we will run the code:

A black background with white text

Description automatically generated

Error occurred : Since logger.py is under src so we need to run it python src/logging.py

A black background with white text

Description automatically generated

New folder is created named log

A black background with numbers

Description automatically generated

A black background with white text

Description automatically generated

Implementing the project:

Do the EDA and model training part in notebook folder

Tutorial-4: https://youtu.be/\_0v1UK7smBc?si=pCiCYH22UuiTbCFI

Data ingestion :

Goal is to create a

artifact folder is to keep all these files organized in a central location, making it easier for developers to manage and track the different components of the project.

SRC/components/data\_ingestion.py

Step-1 read data from specific data source

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

Run the code on the terminal

A black background with white text

Description automatically generated

New folder name artifact is cretated consiting of train,test and raw data:

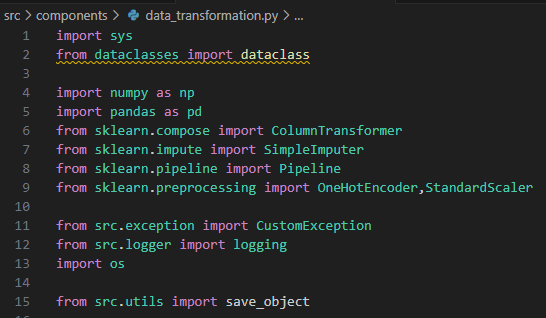
A screenshot of a computer program

Description automatically generated

Tutorial-5 <https://youtu.be/Zs2BZkgoivM?si=gpkuEmURcCTGW9Ll>

Data \_transformation.py

Feature engineering, data cleaning,etc for categorical and numerical features

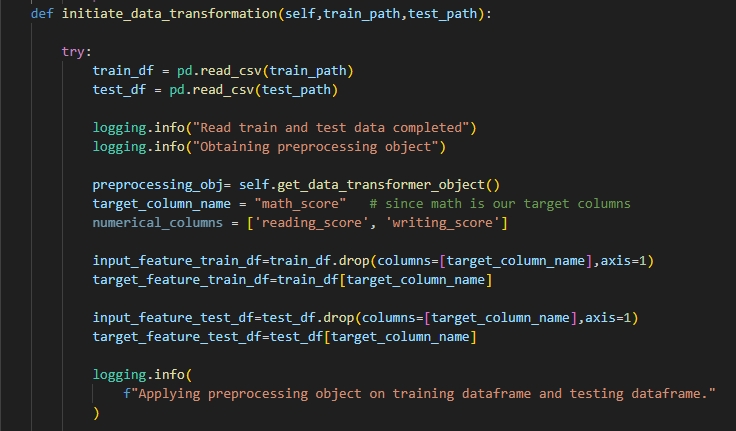


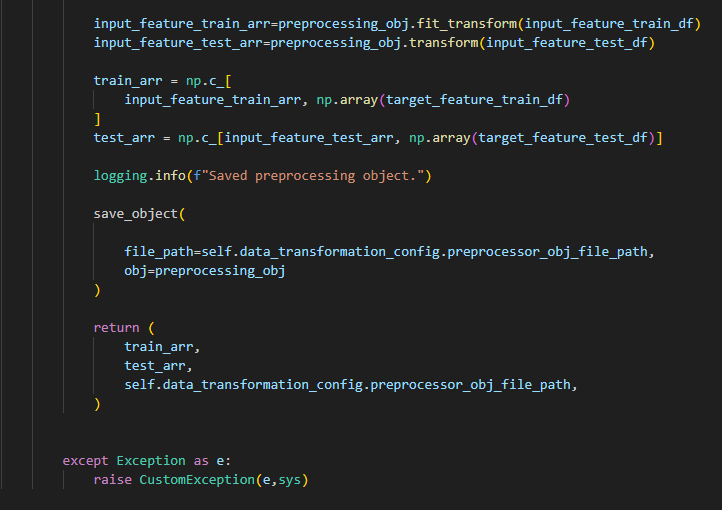
A computer screen shot of a program code

Description automatically generated

A computer screen shot of code

Description automatically generated





After doing all this fo to utils.py:

A screen shot of a computer program

Description automatically generated

Run the data\_ingestion.py in terminal

A black screen with white text

Description automatically generated

Now we can see in our log :

A screen shot of a computer program

Description automatically generated

And new pickle file is created in artifact:

A screenshot of a computer program

Description automatically generated

Tutorial -6 <https://youtu.be/EAWR1kFtEGo?si=5_Mwdk2FhYA3PPKl>

src\components\model\_trainer.py

A screenshot of a computer program

Description automatically generated

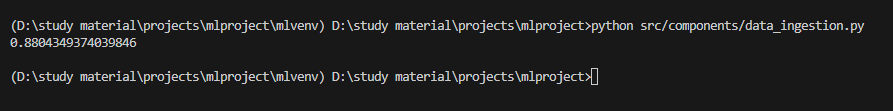
**A computer screen with many colorful text

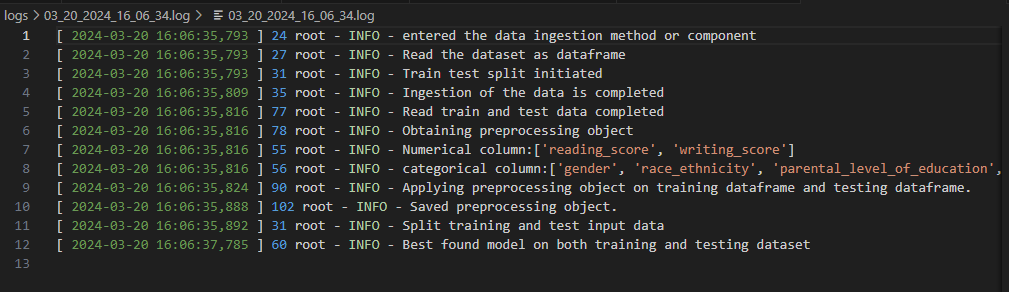
Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Now running the data\_ingestion.py**





And we will get model.pkl and preprocesssor.pkl file

Tutorial-7 Hyperparameter tuning- <https://youtu.be/oMZA8v4JECY?si=KFefRAZ97bGlpzQW>

A computer screen shot of a program

Description automatically generated

Adding all this parameter in model\_trainer.py



Adding params feature both in utility.py and model\_trainer.py

In model\_trainer.py



In utility.py modified code is

A computer screen shot of a program code

Description automatically generated



Executing data\_ingestion.py we get the best r-square

A screen shot of a computer program

Description automatically generated

Here we found that we found the best model to do prediction now we will create prediction pipeline using flask

Tutorial-8 creating prediction pipeline using flask on web- <https://youtu.be/d6Jw5hGb65Y?si=YHAD_wHNbB4EUz5T>

Create a new file app.py and we are going to work in src\pipeline\prediction\_pipeline.py and src\pipeline\train\_pipeline.py

A computer screen shot of a program

Description automatically generated

Create a template folder inside it create a file index.html and home.html

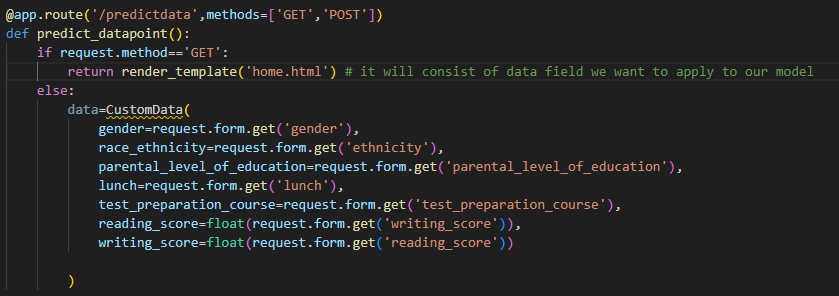
A screenshot of a computer

Description automatically generated

In home.html predict\_datapoint we will get from app.py

A black screen with white text

Description automatically generated



This custom data source we will get from predict pipeline

Go to predict pipeline:

import sys

import pandas as pd

from src.exception import CustomException

from src.utils import load\_object

import os

class PredictPipeline:

    def \_\_init\_\_(self):

        pass

    def predict(self,features): # this is model prediction file

        try:

            model\_path=os.path.join("artifact","model.pkl")

            preprocessor\_path=os.path.join('artifact','preprocessor.pkl')

            print("Before Loading")

            model=load\_object(file\_path=model\_path) # loading model(import pickle file) load\_object is created in utils.py)

            preprocessor=load\_object(file\_path=preprocessor\_path)

            print("After Loading")

            data\_scaled=preprocessor.transform(features)

            preds=model.predict(data\_scaled)

            return preds

        except Exception as e:

            raise CustomException(e,sys)

class CustomData:

    def \_\_init\_\_(  self,

        gender: str,

        race\_ethnicity: str,

        parental\_level\_of\_education,

        lunch: str,

        test\_preparation\_course: str,

        reading\_score: int,

        writing\_score: int):

        self.gender = gender

        self.race\_ethnicity = race\_ethnicity

        self.parental\_level\_of\_education = parental\_level\_of\_education

        self.lunch = lunch

        self.test\_preparation\_course = test\_preparation\_course

        self.reading\_score = reading\_score

        self.writing\_score = writing\_score

    def get\_data\_as\_data\_frame(self):

        try:

            custom\_data\_input\_dict = {

                "gender": [self.gender],

                "race\_ethnicity": [self.race\_ethnicity],

                "parental\_level\_of\_education": [self.parental\_level\_of\_education],

                "lunch": [self.lunch],

                "test\_preparation\_course": [self.test\_preparation\_course],

                "reading\_score": [self.reading\_score],

                "writing\_score": [self.writing\_score],

            }

            return pd.DataFrame(custom\_data\_input\_dict)

        except Exception as e:

            raise CustomException(e, sys)

finished utils.py:

import os

import sys

import pickle

import dill

import numpy as np

import pandas as pd

from src.exception import CustomException

from sklearn.metrics import r2\_score

from sklearn.model\_selection import GridSearchCV

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)

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

    except Exception as e:

        raise CustomException(e, sys)

    # defining a function which will load the pickle file in readby mode

def load\_object(file\_path):

    try:

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

            return pickle.load(file\_obj)

    except Exception as e:

        raise CustomException(e, sys)

finished 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("artifact","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(),

                "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)

             ## 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)

Finished app.py:

from flask import Flask,request,render\_template

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from src.pipeline.prediction\_pipeline import CustomData,PredictPipeline

application=Flask(\_\_name\_\_)

app=application

## Route for a home page

@app.route('/')

def index():

    return render\_template('index.html')

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

def predict\_datapoint():

    if request.method=='GET':

        return render\_template('home.html') # it will consist of data field we want to apply to our model

    else:

        data=CustomData(

            gender=request.form.get('gender'), # request will get the data from prediction\_pipeline custom data

            race\_ethnicity=request.form.get('ethnicity'),

            parental\_level\_of\_education=request.form.get('parental\_level\_of\_education'),

            lunch=request.form.get('lunch'),

            test\_preparation\_course=request.form.get('test\_preparation\_course'),

            reading\_score=float(request.form.get('writing\_score')),

            writing\_score=float(request.form.get('reading\_score'))

        )

        pred\_df=data.get\_data\_as\_data\_frame() # get\_data\_as\_data\_frame is present in predict\_pipeline.py

        print(pred\_df)

        print("Before Prediction")

        predict\_pipeline=PredictPipeline()

        print("Mid Prediction")

        results=predict\_pipeline.predict(pred\_df) # prepresent in predict\_pipeline.py

        print("after Prediction")

        return render\_template('home.html',results=results[0]) # returning output and we will copy it in html file to get the result

    # now we will run the app.py

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

    app.run(host="0.0.0.0",debug=True)

Now we will run app.py:

A black screen with white text

Description automatically generated

Go to goole and write 127.0.0.1:5000

A screenshot of a computer

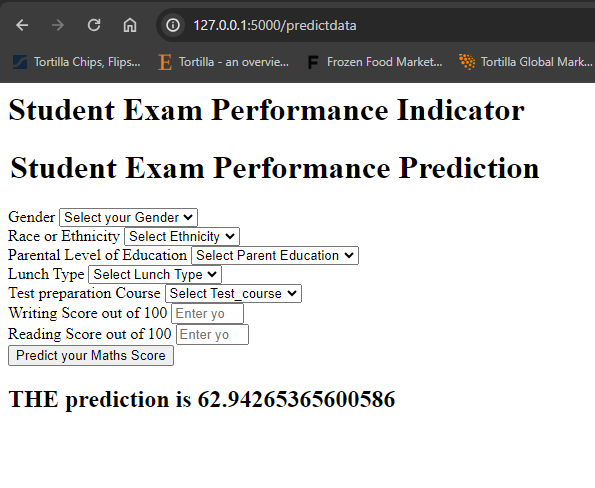
Description automatically generated

Go to google and write - 127.0.0.1:5000/predictdata

A screenshot of a computer

Description automatically generated

Here we can edit the field and get the prediction:



Here we get our maths predicted score.