1 Data Science Pipelne

1. **Data Ingestion**
2. **Data Validation**
3. **Model Training**
4. **Model Analysis Validation**
5. **Model Deployment**

2 ML Project Insurance

2.1 Problem Statement

The purpose of this data is to look into the different features to observe their relationship, ML model is based on several features of individuals such as age, physical/family condition, and location against their existing medical expense to be used for predicting future medical expenses of individuals that help medical insurance to make a decision on charging the premium.

2.2 Tech Stack Used

- **Python Modular Coding**
- **Machine Learning**
- **MongoDB Database**
- **AWS Cloud**
- **Apache**
- **Docker**
- **Grafana**
- **DVC**
- **MLFLOW**

Part- 1

GitHub

1. Create a new GitHub Repository
2. Choose the repository **`Public`** or **`Private`**
3. Add a **`README.md`** file
4. Add a **`.gitignore`** file and Choose **`Python`**
5. Chosse a license, generally we choose the **`Apache license 2.0`**
6. Now our repository is created.
7. Click on **`<> Code`** and copy the `repository link`
8. Open the **`Command Prompt`**
9. Copy the **`Project Folder`** and paste that in Command Prompt,
10. Run the command in command prompt

```
git clone https://github.com/
```

11. To Open **`VS Code`** , run the following command

```
code .
```

12. Get the data into the same folder.

Now start to write the code

1. Create a new file and name it **`data_dump.py`** and write the folllowing code

```python
import pymongo # pip install pymongo
import pandas as pd
import json

client = pymongo.MongoClient("mongodb+srv://gl0427:wasiq123@cluster0.afuprqr.mo
ngodb.net/?retryWrites=true&w=majority")

DATA_FILE_PATH= (r"E:/wasiq/Insurance_Project/insurance.csv")
DATABASE_NAME= "INSURANCE"
COLLECTION_NAME= "INSURANCE_PROJECT"

if __name__ == "__main__":
    df= pd.read_csv(DATA_FILE_PATH)
    print(f"Rows and Columns: {df.shape}")

    df.reset_index(drop=True, inplace=True)

    json_record = list(json.loads(df.T.to_json()).values())
    print(json_record[0])

    client[DATABASE_NAME][COLLECTION_NAME].insert_many(json_record)
```

```python
import pymongo # pip install pymongo
import pandas as pd
import json

client = pymongo.MongoClient("<mongodb_url>")

DATA_FILE_PATH= (r"DataFilePath")
DATABASE_NAME= "INSURANCE"
COLLECTION_NAME= "INSURANCE_PROJECT"

if __name__ == "__main__":
    df= pd.read_csv(DATA_FILE_PATH)
    print(f"Rows and Columns: {df.shape}")

    df.reset_index(drop=True, inplace=True)

    json_record = list(json.loads(df.T.to_json()).values())
    print(json_record[0])

    client[DATABASE_NAME][COLLECTION_NAME].insert_many(json_record)
```

2. Create a new file and name it **`template.py`** and write the folllowing code

```
import os
from pathlib import Path
import logging
```

logging.basicConfig(level = logging.INFO, format = "[%(asctime)s: %(levelname)s]: %(message)s")

while True: project_name = input("Enter your project name: ") if project_name !='': break

logging.info(f"Creating project by name: {project_name}")

list_of_files = [".github/workflows/.gitkeep", ".github/workflows/main.yaml", # f"src/{project_name}/**init**.py", f"{project_name}/**init**.py", f"{project_name}/components/**init**.py", f"{project_name}/components/data_ingestion.py", f"{project_name}/components/data_transformation.py", f"{project_name}/components/data_validation.py", f"{project_name}/components/data_evaluation.py", f"{project_name}/components/data_pusher.py", f"{project_name}/components/data_trainer.py", f"{project_name}/entity/**init**.py", f"{project_name}/exception/**init**.py", f"{project_name}/pipeline/**init**.py", f"{project_name}/logger/**init**.py", f"{project_name}/config.py", f"{project_name}/exception.py", f"{project_name}/predictor.py", f"{project_name}/utils.py", f"{project_name}/entity/**init**.py", f"{project_name}/entity/artifact_entity.py", f"{project_name}/entity/config_entity.py", f"configs/config.yaml", "requirements.txt", "setup.py", "main.py" "data_dump.py", "README.md"]

for filepath in list_of_files: filepath = Path(filepath) filedir, filename = os.path.split(filepath) if filedir !="": os.makedirs(filedir, exist_ok=True) logging.info(f"Creating a new directory at : {filedir} for file: {filename}") if (not os.path.exists(filepath)) or (os.path.getsize(filepath) == 0): with open(filepath, "w") as f: pass logging.info(f"Creating a new file: {filename} for path: {filepath}") else: logging.info(f"file is already present at: {filepath}")

```
It ask the name of the project
```bash
Enter project name: project_name
```

It Create all required files and folders for the project.

Run the following command:

```
git add .
git commit -m "folder structure and dump in database completed"
```

Part - 2

Project Building

- **Setup file understanding & Implementation**
- **Logger understanding & Implementation**
- **Exception understanding & Implementation**
```

Start the coding in `setup.py`

```
from setuptools import find_packages, setup
from typing import List

requriment_file_name = "requirements.txt"
REMOVE_PACKAGE = "-e ."

def get_requirements()->List[str]:
    with open(requriment_file_name) as requirement_file:
        requriment_list = requirement_file.readline()
    requriment_list = [requriment_name.replace("\n", "") for requriment_name in
requriment_list]

    if REMOVE_PACKAGE in requriment_list:
        requriment_list.remove(REMOVE_PACKAGE)
    return requriment_list



setup(name='Insurance',
      version='0.0.1',
      description='Insurance Industry lavel project',
      author='Mohammad Wasiq',
      author_email='mohammadwasiq0786@gmail.com',
      packages=find_packages(),
      install_reqires = get_requirements()
     )
```

Write the `requirements.txt` file

```
dnspython==2.2.1
fastapi==0.78.0
httptools==0.5.0
imblearn==0.0
pip-chill==1.0.1
#python-dotenv==0.21.0
uvicorn==0.18.3
watchfiles==0.17.0
websockets==10.3
wincertstore==0.2
xgboost==1.6.2
pandas
PyYAML
numpy
scikit-learn
apache-airflow
-e .
```

Install the requirements.txt

```
pip install -r requirements.txt
```

Update the GitHub repository

```
git add .
git commit -m "setup file completed"
git push origin main
```

3.5 Write the following code into `logger` folder's `___init__.py` file

```
import logging
from datetime import datetime
import os

# Creating logs directory to store log in files
LOG_DIR = "Insurance_log"

# Creating file name for log file based on current timestamp
CURRENT_TIME_STAMP = f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}"

# Here, We are going to define the path to store log with folder_name
LOG_FIlE_NAME = f"log_{CURRENT_TIME_STAMP}.log"


#Creating LOG_DIR if it does not exists.
os.makedirs(LOG_DIR, exist_ok=True)

#Creating file path for projects.
LOG_FIlE_PATH = os.path.join(LOG_DIR, LOG_FIlE_NAME)

# If you want to read log select baseconfig and press f12 from your system.
logging.basicConfig(filename=LOG_FIlE_PATH,
filemode = "w",
format = '[%(asctime)s] %(name)s - %(levelname)s - %(message)s',
#level=logging.INFO,
level=logging.DEBUG,
)
```

3.6 Write the following code into `exception` folder's

___init__.py file

```
import os
import sys


class InsuranceException(Exception):


    def __init__(self, error_message:Exception, error_detail:sys):
        super().__init__(error_message)
        self.error_message = InsuranceException.error_message_detail(error_mess
age, error_detail=error_detail)

    @staticmethod
    def error_message_detail(error:Exception, error_detail:sys)->str:
        """
        error: Exception object raise from module
        error_detail: is sys module contains detail information about system ex
ecution information.
        """
        _, _, exc_tb = error_detail.exc_info()
        line_number = exc_tb.tb_frame.f_lineno

        #extracting file name from exception traceback
        file_name = exc_tb.tb_frame.f_code.co_filename

        #preparing error message
        error_message = f"Error occurred python script name [{file_name}]" \
                        f" line number [{exc_tb.tb_lineno}] error message [{err
or}]."

        return error_message

    def __str__(self):
        """
        Formating how a object should be visible if used in print statement.
        """
        return self.error_message


    def __repr__(self):
        """
        Formating object of AppException
        """
        return InsuranceException.__name__.__str__()
```

3.7 Write the following code into `main.py` file

```
from Insurance.logger import logging
from Insurance.exception import InsuranceException
from Insurance.utils import get_collection_as_dataframe
import sys, os
from Insurance.entity.config_entity import DataIngestionConfig
from Insurance.entity import config_entity
from Insurance.components.data_ingestion import DataIngestion
from Insurance.components.data_validation import DataValidation
from Insurance.components.model_trainer import ModelTrainer
from Insurance.components.data_transformation import DataTransformation
from Insurance.components.model_evaluation import ModelEvaluation
from Insurance.components.model_pusher import ModelPusher

#def test_logger_and_expection():
   # try:
        # logging.info("Starting the test_logger_and_exception")
         #result = 3/0
        # print(result)
        # logging.info("Stoping the test_logger_and_exception")
    #except Exception as e:
      #  logging.debug(str(e))
        # raise InsuranceException(e, sys)

if __name__=="__main__":
     try:
          #start_training_pipeline()
          #test_logger_and_expection()
        # get_collection_as_dataframe(database_name ="INSURANCE", collection_nam
e = 'INSURANCE_PROJECT')
        training_pipeline_config = config_entity.TrainingPipelineConfig()

      #data ingestion
        data_ingestion_config  = config_entity.DataIngestionConfig(training_pipe
line_config=training_pipeline_config)
        print(data_ingestion_config.to_dict())
        data_ingestion = DataIngestion(data_ingestion_config=data_ingestion_conf
ig)
        data_ingestion_artifact = data_ingestion.initiate_data_ingestion()

        # Data Validation
        data_validation_config = config_entity.DataValidationConfig(training_pip
eline_config=training_pipeline_config)
        data_validation = DataValidation(data_validation_config=data_validation_
config,
                         data_ingestion_artifact=data_ingestion_artifact)

        data_validation_artifact = data_validation.initiate_data_validation()

      #Data Transformation
```

```
        data_transformation_config = config_entity.DataTransformationConfig(trai
ning_pipeline_config=training_pipeline_config)
        data_transformation = DataTransformation(data_transformation_config=data
_transformation_config,
        data_ingestion_artifact=data_ingestion_artifact)
        data_transformation_artifact = data_transformation.initiate_data_transfo
rmation()


      # Model Trainer
        model_trainer_config = config_entity.ModelTrainingConfig(training_pipeli
ne_config = training_pipeline_config)
        model_trainer = ModelTrainer(model_trainer_config=model_trainer_config,
data_transformation_artifact=data_transformation_artifact)
        model_trainer_artifact = model_trainer.initiate_model_trainer()


      # Model Evaluation
        model_eval_config = config_entity.ModelEvaluationConfig(training_pipelin
e_config = training_pipeline_config)
        model_eval = ModelEvaluation(model_eval_config = model_eval_config,
        data_ingestion_artifact = data_ingestion_artifact,
        data_transformation_artifact = data_transformation_artifact,
        model_trainer_artifact = model_trainer_artifact)
        model_evl_artifact = model_eval.intitate_model_evaluation()


      # Model Pusher
        model_pusher_config = config_entity.ModelPusherConfig(training_pipeline_
config=training_pipeline_config)
        model_pusher = ModelPusher(model_pusher_config=model_pusher_config,
                        data_transformation_artifact= data_transformation_artifac
t,
                        model_trainer_artifact= model_trainer_artifact)
        Model_pusher_artifact = model_pusher.initiate_model_pusher()


    except Exception as e:
        print(e)
```

Part - 3

Write the following code in `utils.py` file of Insurance

Folder

```
import pandas as pd
from Insurance.logger import logging
from Insurance.exception import InsuranceException
from Insurance.config import mongo_client
import os,sys
import yaml
import numpy as np
import dill

def get_collection_as_dataframe(database_name:str,collection_name:str)->pd.Data
Frame:
    """
    Description: This function return collection as dataframe
    ========================================================
    Params:
    database_name: database name
    collection_name: collection name
    ========================================================
    return Pandas dataframe of a collection
    """
    try:
        logging.info(f"Reading data from database: {database_name} and collecti
on: {collection_name}")
        df = pd.DataFrame(list(mongo_client[database_name][collection_name].fin
d()))
        logging.info(f"Found columns: {df.columns}")
        if "_id" in df.columns:
            logging.info(f"Dropping column: _id ")
            df = df.drop("_id",axis=1)
        logging.info(f"Row and columns in df: {df.shape}")
        return df
    except Exception as e:
        raise InsuranceException(e, sys)

#**********************************## Data_Validation**********************
**********
def write_yaml_file(file_path,data:dict):
    try:
        file_dir = os.path.dirname(file_path)
        os.makedirs(file_dir,exist_ok=True)
        with open(file_path,"w") as file_writer:
            yaml.dump(data,file_writer)
    except Exception as e:
        raise InsuranceException(e, sys)

def convert_columns_float(df:pd.DataFrame,exclude_columns:list)->pd.DataFrame:
    try:
        for column in df.columns:
            if column not in exclude_columns:
                if df[column].dtypes != 'O':
```

```
                    df[column]=df[column].astype('float')
        return df
    except Exception as e:
        raise e

#********************************** Data_Transformation**********************
******************

def save_object(file_path: str, obj: object) -> None:
    try:
        logging.info("Entered the save_object method of utils")
        os.makedirs(os.path.dirname(file_path), exist_ok=True)
        with open(file_path, "wb") as file_obj:
            dill.dump(obj, file_obj)
        logging.info("Exited the save_object method of utils")
    except Exception as e:
        raise InsuranceException(e, sys) from e


def load_object(file_path: str, ) -> object:
    try:
        if not os.path.exists(file_path):
            raise Exception(f"The file: {file_path} is not exists")
        with open(file_path, "rb") as file_obj:
            return dill.load(file_obj)
    except Exception as e:
        raise InsuranceException(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 InsuranceException(e, sys) from e

#**********************************## Model Training**************************
****************

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 InsuranceException(e, sys) from e
```

4.2 Create a file `.env` and write the following code

```
MONGO_DB_URL= "mongodb+srv://gl0427:wasiq123@cluster0.afuprqr.mongodb.net/?retr
yWrites=true&w=majority"
```

4.3 Write the following code in `__init__.py` file of Insurance Folder

```
from dotenv import load_dotenv
print(f"loading env variable from .env")
load_dotenv()
```

4.4 Write the following code in `config.py` file of Insurance Folder

```
import pymongo
import pandas as pd
import json
from dataclasses import dataclass
# Provide the mongodb localhost url to connect python to mongodb.
import os

@dataclass
class EnvironmentVariable:
    mongo_db_url:str = os.getenv("MONGO_DB_URL")
    #aws_access_key_id:str = os.getenv("AWS_ACCESS_KEY_ID")
    #aws_access_secret_key:str = os.getenv("AWS_SECRET_ACCESS_KEY")


env_var = EnvironmentVariable()
mongo_client = pymongo.MongoClient(env_var.mongo_db_url)
TARGET_COLUMN = "expenses"
print(env_var.mongo_db_url)
```

4.4.1 Run `main.py` file

4.4.2 GitHub Commands are as follows:

```
git add .
git commit -m "Utils file config file updated"
git push origin main
```

4.5 Creaate a Folder and named `entity` in Insurance Folder

4.5.1 Creatre a file `__init__.py` in entity folder

4.5.2 Creatre a file `artifact_entity.py` in entity folder and write the following code

```python
from dataclasses import dataclass

@dataclass
class DataIngestionArtifact:
    feature_store_file_path:str
    train_file_path:str
    test_file_path:str

@dataclass
class DataValidationArtifact:
    report_file_path:str

@dataclass
class DataTransformationArtifact:
    transform_object_path:str
    transformed_train_path:str
    transformed_test_path:str
    target_encoder_path:str

@dataclass
class ModelTrainerArtifact:
    model_path:str
    r2_train_score:float
    r2_test_score:float

@dataclass
class ModelEvaluationArtifact:
    is_model_accepted:bool
    improved_accuracy:float

@dataclass
class ModelPusherArtifact:
    pusher_model_dir:str
    saved_model_dir:str
```

4.5.3 Creatre a file `config_entity.py` in entity folder and write the

following code

```
import os,sys
from Insurance.exception import InsuranceException
from Insurance.logger import logging
from datetime import datetime

FILE_NAME = "insurance.csv"
TRAIN_FILE_NAME = "train.csv"
TEST_FILE_NAME = "test.csv"
TRANSFORMER_OBJECT_FILE_NAME = "transformer.pkl"
MODEL_FILE_NAME = "model.pkl"

TARGET_ENCODER_OBJECT_FILE_NAME = "target_encoder.pkl"

class TrainingPipelineConfig:

    def __init__(self):
        try:
            self.artifact_dir = os.path.join(os.getcwd(),"artifact",f"{datetim
e.now().strftime('%m%d%Y__%H%M%S')}")
        except Exception  as e:
            raise InsuranceException(e,sys)


class DataIngestionConfig:

    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        try:
            self.database_name="INSURANCE"
            self.collection_name="INSURANCE_PROJECT"
            self.data_ingestion_dir = os.path.join(training_pipeline_config.art
ifact_dir , "data_ingestion")
            self.feature_store_file_path = os.path.join(self.data_ingestion_di
r,"feature_store",FILE_NAME)
            self.train_file_path = os.path.join(self.data_ingestion_dir,"datase
t",TRAIN_FILE_NAME)
            self.test_file_path = os.path.join(self.data_ingestion_dir,"datase
t",TEST_FILE_NAME)
            self.test_size = 0.2
        except Exception  as e:
            raise InsuranceException(e,sys)


# Convert data into dict
    def to_dict(self,)->dict:
        try:
            return self.__dict__
        except Exception  as e:
            raise InsuranceException(e,sys)

class DataValidationConfig:
```

```
    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.data_validation_dir = os.path.join(training_pipeline_config.artifa
ct_dir , "data_validation")
        self.report_file_path=os.path.join(self.data_validation_dir, "report.ya
ml")
        self.missing_threshold:float = 0.2
        self.base_file_path = os.path.join("insurance.csv")


class DataTransformationConfig:

    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.data_transformation_dir = os.path.join(training_pipeline_config.ar
tifact_dir , "data_transformation")
        self.transform_object_path = os.path.join(self.data_transformation_di
r,"transformer",TRANSFORMER_OBJECT_FILE_NAME)
        self.transformed_train_path =  os.path.join(self.data_transformation_di
r,"transformed",TRAIN_FILE_NAME.replace("csv","npz"))
        self.transformed_test_path =os.path.join(self.data_transformation_di
r,"transformed",TEST_FILE_NAME.replace("csv","npz"))
        self.target_encoder_path = os.path.join(self.data_transformation_dir,"t
arget_encoder",TARGET_ENCODER_OBJECT_FILE_NAME)


class ModelTrainerConfig:

    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.model_trainer_dir = os.path.join(training_pipeline_config.artifact
_dir , "model_trainer")
        self.model_path = os.path.join(self.model_trainer_dir,"model",MODEL_FIL
E_NAME)
        self.expected_score = 0.7
        self.overfitting_threshold = 0.3 # overfiting score


class ModelEvaluationConfig:
    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.change_threshold = 0.01


class ModelPusherConfig:

    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.model_pusher_dir = os.path.join(training_pipeline_config.artifact_
dir , "model_pusher")
        self.saved_model_dir = os.path.join("saved_models")
        self.pusher_model_dir = os.path.join(self.model_pusher_dir,"saved_model
s")
        self.pusher_model_path = os.path.join(self.pusher_model_dir,MODEL_FILE_
NAME)
        self.pusher_transformer_path = os.path.join(self.pusher_model_dir,TRANS
```

```
FORMER_OBJECT_FILE_NAME)
        self.pusher_target_encoder_path = os.path.join(self.pusher_model_dir,TA
RGET_ENCODER_OBJECT_FILE_NAME)
```

4.5.4 And Update the `main.py` file

```
from Insurance.logger import logging
from Insurance.exception import InsuranceException
from Insurance.utils import get_collection_as_dataframe
import sys, os
from Insurance.entity.config_entity import DataIngestionConfig
from Insurance.entity import config_entity
from Insurance.components.data_ingestion import DataIngestion
from Insurance.components.data_validation import DataValidation
from Insurance.components.data_transformation import DataTransformation
from Insurance.components.model_trainer import ModelTrainer
from Insurance.components.model_evaluation import ModelEvaluation
from Insurance.components.model_pusher import ModelPusher




#def test_logger_and_expection():
   # try:
        # logging.info("Starting the test_logger_and_exception")
         #result = 3/0
        # print(result)
        # logging.info("Stoping the test_logger_and_exception")
    #except Exception as e:
      #  logging.debug(str(e))
        # raise InsuranceException(e, sys)

if __name__=="__main__":
     try:
          #start_training_pipeline()
          #test_logger_and_expection()
        # get_collection_as_dataframe(database_name ="INSURANCE", collection_nam
e = 'INSURANCE_PROJECT')
        training_pipeline_config = config_entity.TrainingPipelineConfig()

       #data ingestion
        data_ingestion_config  = config_entity.DataIngestionConfig(training_pipe
line_config=training_pipeline_config)
        print(data_ingestion_config.to_dict())
        data_ingestion = DataIngestion(data_ingestion_config=data_ingestion_conf
ig)
        data_ingestion_artifact = data_ingestion.initiate_data_ingestion()

        #data validation
        data_validation_config = config_entity.DataValidationConfig(training_pip
eline_config=training_pipeline_config)
        data_validation = DataValidation(data_validation_config=data_validation_
config,
                         data_ingestion_artifact=data_ingestion_artifact)

        data_validation_artifact = data_validation.initiate_data_validation()
```

```
        # data transformation
        data_transformation_config = config_entity.DataTransformationConfig(trai
ning_pipeline_config=training_pipeline_config)
        data_transformation = DataTransformation(data_transformation_config=data
_transformation_config,
        data_ingestion_artifact=data_ingestion_artifact)
        data_transformation_artifact = data_transformation.initiate_data_transfo
rmation()

        #model trainer
        model_trainer_config = config_entity.ModelTrainerConfig(training_pipelin
e_config=training_pipeline_config)
        model_trainer = ModelTrainer(model_trainer_config=model_trainer_config,
data_transformation_artifact=data_transformation_artifact)
        model_trainer_artifact = model_trainer.initiate_model_trainer()


       #model evaluation
        model_eval_config = config_entity.ModelEvaluationConfig(training_pipelin
e_config=training_pipeline_config)
        model_eval  = ModelEvaluation(model_eval_config=model_eval_config,
        data_ingestion_artifact=data_ingestion_artifact,
        data_transformation_artifact=data_transformation_artifact,
        model_trainer_artifact=model_trainer_artifact)
        model_eval_artifact = model_eval.initiate_model_evaluation()


       # model pusher
        model_pusher_config = config_entity.ModelPusherConfig(training_pipeline_
config=training_pipeline_config)
        model_pusher = ModelPusher(model_pusher_config=model_pusher_config,
                                   data_transformation_artifact=data_transfor
mation_artifact,
                                   model_trainer_artifact=model_trainer_artif
act)
        model_pusher_artifact = model_pusher.initiate_model_pusher()

    except Exception as e:
        print(e)
```

5 Part- 4 & 5

5.1 Create a folder `components` in the existing folder

5.1.1 Create a file `__init__.py` in the components folder

5.1.2 Create a file `data_ingestion.py` in the components folder and

write the following code

```
from Insurance import utils
from Insurance.entity import config_entity
from Insurance.entity import artifact_entity
from Insurance.exception import InsuranceException
from Insurance.logger import logging
import os,sys
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

class DataIngestion:

    def __init__(self,data_ingestion_config:config_entity.DataIngestionConfig
):
        try:
            self.data_ingestion_config = data_ingestion_config
        except Exception as e:
            raise InsuranceException(e, sys)

    def initiate_data_ingestion(self)->artifact_entity.DataIngestionArtifact:
        try:
            logging.info(f"Exporting collection data as pandas dataframe")
            #Exporting collection data as pandas dataframe
            df:pd.DataFrame  = utils.get_collection_as_dataframe(
                database_name=self.data_ingestion_config.database_name,
                collection_name=self.data_ingestion_config.collection_name)

            logging.info("Save data in feature store")

            #replace na with Nan
            df.replace(to_replace="na",value=np.NAN,inplace=True)

            #Save data in feature store
            logging.info("Create feature store folder if not available")
            #Create feature store folder if not available
            feature_store_dir = os.path.dirname(self.data_ingestion_config.feat
ure_store_file_path)
            os.makedirs(feature_store_dir,exist_ok=True)
            logging.info("Save df to feature store folder")
            #Save df to feature store folder
            df.to_csv(path_or_buf=self.data_ingestion_config.feature_store_file
_path,index=False,header=True)


            logging.info("split dataset into train and test set")
            #split dataset into train and test set
            train_df,test_df = train_test_split(df,test_size=self.data_ingestio
n_config.test_size, random_state = 1)
```

```
            logging.info("create dataset directory folder if not available")
            #create dataset directory folder if not available
            dataset_dir = os.path.dirname(self.data_ingestion_config.train_file
_path)
            os.makedirs(dataset_dir,exist_ok=True)

            logging.info("Save df to feature store folder")
            #Save df to feature store folder
            train_df.to_csv(path_or_buf=self.data_ingestion_config.train_file_p
ath,index=False,header=True)
            test_df.to_csv(path_or_buf=self.data_ingestion_config.test_file_pat
h,index=False,header=True)

            #Prepare artifact

            data_ingestion_artifact = artifact_entity.DataIngestionArtifact(
                feature_store_file_path=self.data_ingestion_config.feature_stor
e_file_path,
                train_file_path=self.data_ingestion_config.train_file_path,
                test_file_path=self.data_ingestion_config.test_file_path)

            logging.info(f"Data ingestion artifact: {data_ingestion_artifact}")
            return data_ingestion_artifact

        except Exception as e:
            raise InsuranceException(error_message=e, error_detail=sys)
```

5.1.3 Create a file `data_transformation.py` in the components folder

and write the following code

```
from Insurance.entity import artifact_entity,config_entity
from Insurance.exception import InsuranceException
from Insurance.logger import logging
from typing import Optional
import os,sys
from sklearn.pipeline import Pipeline
import pandas as pd
from Insurance import utils
import numpy as np
from sklearn.preprocessing import LabelEncoder
from imblearn.combine import SMOTETomek
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from Insurance.config import TARGET_COLUMN

# Missing values imputation
# Outliers Handling
# Imbalanced data handling
# Convert Categorical data into numerical data

class DataTransformation:


    def __init__(self,data_transformation_config:config_entity.DataTransformati
onConfig,
                 data_ingestion_artifact:artifact_entity.DataIngestionArtifa
ct):
        try:
            logging.info(f"{'>>'*20} Data Transformation {'<<'*20}")
            self.data_transformation_config=data_transformation_config
            self.data_ingestion_artifact=data_ingestion_artifact
        except Exception as e:
            raise InsuranceException(e, sys)


    @classmethod
    def get_data_transformer_object(cls)->Pipeline: # Create cls class
        try:
            simple_imputer = SimpleImputer(strategy='constant', fill_value=0)
            robust_scaler =  RobustScaler()
            pipeline = Pipeline(steps=[
                    ('Imputer',simple_imputer),
                    ('RobustScaler',robust_scaler)
                ])
            return pipeline
        except Exception as e:
            raise InsuranceException(e, sys)


    def initiate_data_transformation(self,) -> artifact_entity.DataTransformati
```

```
onArtifact:
        try:
            #reading training and testing file
            train_df = pd.read_csv(self.data_ingestion_artifact.train_file_pat
h)
            test_df = pd.read_csv(self.data_ingestion_artifact.test_file_path)

            #selecting input feature for train and test dataframe
            input_feature_train_df=train_df.drop(TARGET_COLUMN,axis=1)
            input_feature_test_df=test_df.drop(TARGET_COLUMN,axis=1)

            #selecting target feature for train and test dataframe
            target_feature_train_df = train_df[TARGET_COLUMN]
            target_feature_test_df = test_df[TARGET_COLUMN]

            label_encoder = LabelEncoder()
            # label_encoder.fit(target_feature_train_df)

            #transformation on target columns
            target_feature_train_arr = target_feature_train_df.squeeze()
            target_feature_test_arr = target_feature_test_df.squeeze()

            #transformation on categorical columns
            for col in input_feature_train_df.columns:
                if input_feature_test_df[col].dtypes == 'O':
                    input_feature_train_df[col] = label_encoder.fit_transform(i
nput_feature_train_df[col])
                    input_feature_test_df[col] = label_encoder.fit_transform(in
put_feature_test_df[col])
                else:
                    input_feature_train_df[col] = input_feature_train_df[col]
                    input_feature_test_df[col] = input_feature_test_df[col]


            transformation_pipleine = DataTransformation.get_data_transformer_o
bject()
            transformation_pipleine.fit(input_feature_train_df)

            #transforming input features
            input_feature_train_arr = transformation_pipleine.transform(input_f
eature_train_df)
            input_feature_test_arr = transformation_pipleine.transform(input_fe
ature_test_df)


            # smt = SMOTETomek(random_state=42)
            # logging.info(f"Before resampling in training set Input: {input_fe
ature_train_arr.shape} Target:{target_feature_train_arr.shape}")
            # input_feature_train_arr, target_feature_train_arr = smt.fit_resam
ple(input_feature_train_arr, target_feature_train_arr)
            # logging.info(f"After resampling in training set Input: {input_fea
ture_train_arr.shape} Target:{target_feature_train_arr.shape}")
```

```
            # logging.info(f"Before resampling in testing set Input: {input_fea
ture_test_arr.shape} Target:{target_feature_test_arr.shape}")
            # input_feature_test_arr, target_feature_test_arr = smt.fit_resampl
e(input_feature_test_arr, target_feature_test_arr)
            # logging.info(f"After resampling in testing set Input: {input_feat
ure_test_arr.shape} Target:{target_feature_test_arr.shape}")

            #target encoder
            train_arr = np.c_[input_feature_train_arr, target_feature_train_arr
]
            test_arr = np.c_[input_feature_test_arr, target_feature_test_arr]


            #save numpy array
            utils.save_numpy_array_data(file_path=self.data_transformation_conf
ig.transformed_train_path,
                                        array=train_arr)

            utils.save_numpy_array_data(file_path=self.data_transformation_conf
ig.transformed_test_path,
                                        array=test_arr)


            utils.save_object(file_path=self.data_transformation_config.transfo
rm_object_path,
             obj=transformation_pipleine)

            utils.save_object(file_path=self.data_transformation_config.target_
encoder_path,
            obj=label_encoder)



            data_transformation_artifact = artifact_entity.DataTransformationAr
tifact(
                transform_object_path=self.data_transformation_config.transform
_object_path,
                transformed_train_path = self.data_transformation_config.transf
ormed_train_path,
                transformed_test_path = self.data_transformation_config.transfo
rmed_test_path,
                target_encoder_path = self.data_transformation_config.target_en
coder_path

            )

            logging.info(f"Data transformation object {data_transformation_arti
fact}")
            return data_transformation_artifact
        except Exception as e:
            raise InsuranceException(e, sys)
```

5.1.4 Create a file `data_validation.py` in the components folder and

write the following code

```
from Insurance.entity import artifact_entity,config_entity
from Insurance.exception import InsuranceException
from Insurance.logger import logging
from scipy.stats import ks_2samp
from typing import Optional
import os,sys
import pandas as pd
from Insurance import utils
import numpy as np
from Insurance.config import TARGET_COLUMN




class DataValidation:


    def __init__(self,
                 data_validation_config:config_entity.DataValidationConfig,
                 data_ingestion_artifact:artifact_entity.DataIngestionArtifa
ct):
        try:
            logging.info(f"{'>>'*20} Data Validation {'<<'*20}")
            self.data_validation_config = data_validation_config
            self.data_ingestion_artifact=data_ingestion_artifact
            self.validation_error=dict()
        except Exception as e:
            raise InsuranceException(e, sys)



    def drop_missing_values_columns(self,df:pd.DataFrame,report_key_name:str)->
Optional[pd.DataFrame]:
        """
        This function will drop column which contains missing value more than s
pecified threshold

        df: Accepts a pandas dataframe
        threshold: Percentage criteria to drop a column
        ==========================================================================
==============
        returns Pandas DataFrame if atleast a single column is available after
missing columns drop else None
        """
        try:

            threshold = self.data_validation_config.missing_threshold
            null_report = df.isna().sum()/df.shape[0]
            #selecting column name which contains null
            logging.info(f"selecting column name which contains null above to
```

```
{threshold}")
            drop_column_names = null_report[null_report>threshold].index

            logging.info(f"Columns to drop: {list(drop_column_names)}")
            self.validation_error[report_key_name]=list(drop_column_names)
            df.drop(list(drop_column_names),axis=1,inplace=True)

            #return None no columns left
            if len(df.columns)==0:
                return None
            return df
        except Exception as e:
            raise InsuranceException(e, sys)
    #     print("dsgfg")
    def is_required_columns_exists(self,base_df:pd.DataFrame,current_df:pd.Data
Frame,report_key_name:str)->bool:
        try:

            base_columns = base_df.columns
            current_columns = current_df.columns

            missing_columns = []
            for base_column in base_columns:
                if base_column not in current_columns:
                    logging.info(f"Column: [{base} is not available.]")
                    missing_columns.append(base_column)

            if len(missing_columns)>0:
                self.validation_error[report_key_name]=missing_columns
                return False
            return True
        except Exception as e:
            raise InsuranceException(e, sys)

    def data_drift(self,base_df:pd.DataFrame,current_df:pd.DataFrame,report_key
_name:str):
        try:
            drift_report=dict()

            base_columns = base_df.columns
            current_columns = current_df.columns

            for base_column in base_columns:
                base_data,current_data = base_df[base_column],current_df[base_c
olumn]
                #Null hypothesis is that both column data drawn from same distr
ubtion

                logging.info(f"Hypothesis {base_column}: {base_data.dtype}, {cu
rrent_data.dtype} ")
                same_distribution =ks_2samp(base_data,current_data)
```

```
                if same_distribution.pvalue>0.05:
                    #We are accepting null hypothesis
                    drift_report[base_column]={
                        "pvalues":float(same_distribution.pvalue),
                        "same_distribution": True
                        }
                else:
                    drift_report[base_column]={
                        "pvalues":float(same_distribution.pvalue),
                        "same_distribution":False
                        }
                    #different distribution

            self.validation_error[report_key_name]=drift_report
        except Exception as e:
            raise InsuranceException(e, sys)

    def initiate_data_validation(self)->artifact_entity.DataValidationArtifact:
        try:
            logging.info(f"Reading base dataframe")
            base_df = pd.read_csv(self.data_validation_config.base_file_path)
            base_df.replace({"na":np.NAN},inplace=True)
            logging.info(f"Replace na value in base df")
            #base_df has na as null
            logging.info(f"Drop null values colums from base df")
            base_df=self.drop_missing_values_columns(df=base_df,report_key_name
="missing_values_within_base_dataset")

            logging.info(f"Reading train dataframe")
            train_df = pd.read_csv(self.data_ingestion_artifact.train_file_pat
h)
            logging.info(f"Reading test dataframe")
            test_df = pd.read_csv(self.data_ingestion_artifact.test_file_path)

            logging.info(f"Drop null values colums from train df")
            train_df = self.drop_missing_values_columns(df=train_df,report_key_
name="missing_values_within_train_dataset")
            logging.info(f"Drop null values colums from test df")
            test_df = self.drop_missing_values_columns(df=test_df,report_key_na
me="missing_values_within_test_dataset")

            exclude_columns = [TARGET_COLUMN]
            base_df = utils.convert_columns_float(df=base_df, exclude_columns=e
xclude_columns)
            train_df = utils.convert_columns_float(df=train_df, exclude_columns
=exclude_columns)
            test_df = utils.convert_columns_float(df=test_df, exclude_columns=e
xclude_columns)


            logging.info(f"Is all required columns present in train df")
            train_df_columns_status = self.is_required_columns_exists(base_df=b
```

```
ase_df, current_df=train_df,report_key_name="missing_columns_within_train_datas
et")
            logging.info(f"Is all required columns present in test df")
            test_df_columns_status = self.is_required_columns_exists(base_df=ba
se_df, current_df=test_df,report_key_name="missing_columns_within_test_datase
t")

            if train_df_columns_status:
                logging.info(f"As all column are available in train df hence de
tecting data drift")
                self.data_drift(base_df=base_df, current_df=train_df,report_key
_name="data_drift_within_train_dataset")
            if test_df_columns_status:
                logging.info(f"As all column are available in test df hence det
ecting data drift")
                self.data_drift(base_df=base_df, current_df=test_df,report_key_
name="data_drift_within_test_dataset")

            #write the report
            logging.info("Write reprt in yaml file")
            utils.write_yaml_file(file_path=self.data_validation_config.report_
file_path,
            data=self.validation_error)

            data_validation_artifact = artifact_entity.DataValidationArtifact(r
eport_file_path=self.data_validation_config.report_file_path,)
            logging.info(f"Data validation artifact: {data_validation_artifac
t}")
            return data_validation_artifact
        except Exception as e:
            raise InsuranceException(e, sys)
```

5.1.5 Create a file `data_evaluation.py` in the components folder and

write the following code

```
from Insurance.predictor import ModelResolver
from Insurance.entity import config_entity,artifact_entity
from Insurance.exception import InsuranceException
from Insurance.logger import logging
from Insurance.utils import load_object
from sklearn.metrics import r2_score
import pandas  as pd
import sys,os
from Insurance.config import TARGET_COLUMN

class ModelEvaluation:

    def __init__(self,
        model_eval_config:config_entity.ModelEvaluationConfig,
        data_ingestion_artifact:artifact_entity.DataIngestionArtifact,
        data_transformation_artifact:artifact_entity.DataTransformationArtifac
t,
        model_trainer_artifact:artifact_entity.ModelTrainerArtifact
        ):
        try:
            logging.info(f"{'>>'*20}  Model Evaluation {'<<'*20}")
            self.model_eval_config=model_eval_config
            self.data_ingestion_artifact=data_ingestion_artifact
            self.data_transformation_artifact=data_transformation_artifact
            self.model_trainer_artifact=model_trainer_artifact
            self.model_resolver = ModelResolver()
        except Exception as e:
            raise InsuranceException(e,sys)



    def initiate_model_evaluation(self)->artifact_entity.ModelEvaluationArtifac
t:
        try:
            #if saved model folder has model the we will compare
            #which model is best trained or the model from saved model folder

            logging.info("if saved model folder has model the we will compare "
            "which model is best trained or the model from saved model folder")
            latest_dir_path = self.model_resolver.get_latest_dir_path()
            if latest_dir_path==None:
                model_eval_artifact = artifact_entity.ModelEvaluationArtifact(i
s_model_accepted=True,
                improved_accuracy=None)
                logging.info(f"Model evaluation artifact: {model_eval_artifac
t}")
                return model_eval_artifact


            #Finding location of transformer model and target encoder
```

```
            logging.info("Finding location of transformer model and target enco
der")
            transformer_path = self.model_resolver.get_latest_transformer_path
()
            model_path = self.model_resolver.get_latest_model_path()
            target_encoder_path = self.model_resolver.get_latest_target_encoder
_path()

            logging.info("Previous trained objects of transformer, model and ta
rget encoder")
            #Previous trained  objects
            transformer = load_object(file_path=transformer_path)
            model = load_object(file_path=model_path)
            target_encoder = load_object(file_path=target_encoder_path)


            logging.info("Currently trained model objects")
            #Currently trained model objects
            current_transformer = load_object(file_path=self.data_transformatio
n_artifact.transform_object_path)
            current_model  = load_object(file_path=self.model_trainer_artifact.
model_path)
            current_target_encoder = load_object(file_path=self.data_transforma
tion_artifact.target_encoder_path)


            # take tyest data for testing test data

            test_df = pd.read_csv(self.data_ingestion_artifact.test_file_path)
            target_df = test_df[TARGET_COLUMN]
            y_true = target_df
            # target_encoder.transform(target_df)
            # accuracy using previous trained model

            """We need to create label encoder object for each categorical vari
able. We will check later"""
            input_feature_name = list(transformer.feature_names_in_)
            for i in input_feature_name:
                if test_df[i].dtypes =='object':
                    test_df[i] =target_encoder.fit_transform(test_df[i])

            input_arr =transformer.transform(test_df[input_feature_name])
            y_pred = model.predict(input_arr)
            print(f"Prediction using previous model: {y_pred[:5]}")
            previous_model_score = r2_score(y_true=y_true, y_pred=y_pred)
            logging.info(f"Accuracy using previous trained model: {previous_mod
el_score}")

            # accuracy using current trained model
            input_feature_name = list(current_transformer.feature_names_in_)
            input_arr =current_transformer.transform(test_df[input_feature_nam
e])
```

```
            y_pred = current_model.predict(input_arr)
            y_true = target_df
            # current_target_encoder.transform(target_df)
            # current_target_encoder.inverse_transform(y_pred[:5])
            print(f"Prediction using trained model: {y_pred[:5]}")
            current_model_score = r2_score(y_true=y_true, y_pred=y_pred)
            logging.info(f"Accuracy using current trained model: {current_model
_score}")
            if current_model_score<=previous_model_score:
                logging.info(f"Current trained model is not better than previou
s model")
                raise Exception("Current trained model is not better than previ
ous model")


            model_eval_artifact = artifact_entity.ModelEvaluationArtifact(is_mo
del_accepted=True,
            improved_accuracy=current_model_score-previous_model_score)
            logging.info(f"Model eval artifact: {model_eval_artifact}")
            return model_eval_artifact
        except Exception as e:
            raise InsuranceException(e,sys)
```

5.1.6 Create a file `data_pusher.py` in the components folder and write

the following code

```
from Insurance.predictor import ModelResolver
from Insurance.entity.config_entity import ModelPusherConfig
from Insurance.exception import InsuranceException
import os,sys
from Insurance.utils import load_object,save_object
from Insurance.logger import logging
from Insurance.entity.artifact_entity import DataTransformationArtifact,ModelTr
ainerArtifact,ModelPusherArtifact

class ModelPusher:

    def __init__(self,model_pusher_config:ModelPusherConfig,
    data_transformation_artifact:DataTransformationArtifact,
    model_trainer_artifact:ModelTrainerArtifact):
        try:
            logging.info(f"{'>>'*20} Data Transformation {'<<'*20}")
            self.model_pusher_config=model_pusher_config
            self.data_transformation_artifact=data_transformation_artifact
            self.model_trainer_artifact=model_trainer_artifact
            self.model_resolver = ModelResolver(model_registry=self.model_pushe
r_config.saved_model_dir)
        except Exception as e:
            raise InsuranceException(e, sys)

    def initiate_model_pusher(self,)->ModelPusherArtifact:
        try:
            #load object
            logging.info(f"Loading transformer model and target encoder")
            transformer = load_object(file_path=self.data_transformation_artifa
ct.transform_object_path)
            model = load_object(file_path=self.model_trainer_artifact.model_pat
h)
            target_encoder = load_object(file_path=self.data_transformation_art
ifact.target_encoder_path)

            #model pusher dir
            logging.info(f"Saving model into model pusher directory")
            save_object(file_path=self.model_pusher_config.pusher_transformer_p
ath, obj=transformer)
            save_object(file_path=self.model_pusher_config.pusher_model_path, o
bj=model)
            save_object(file_path=self.model_pusher_config.pusher_target_encode
r_path, obj=target_encoder)


            #saved model dir
            logging.info(f"Saving model in saved model dir")
            transformer_path=self.model_resolver.get_latest_save_transformer_pa
th()
            model_path=self.model_resolver.get_latest_save_model_path()
```

```
            target_encoder_path=self.model_resolver.get_latest_save_target_enco
der_path()

            save_object(file_path=transformer_path, obj=transformer)
            save_object(file_path=model_path, obj=model)
            save_object(file_path=target_encoder_path, obj=target_encoder)

            model_pusher_artifact = ModelPusherArtifact(pusher_model_dir=self.m
odel_pusher_config.pusher_model_dir,
             saved_model_dir=self.model_pusher_config.saved_model_dir)
            logging.info(f"Model pusher artifact: {model_pusher_artifact}")
            return model_pusher_artifact
        except Exception as e:
            raise InsuranceException(e, sys)
```

5.1.7 Create a file `data_trainer.py` in the components folder and write

the following code

```
from Insurance.entity import artifact_entity,config_entity
from Insurance.exception import InsuranceException
from Insurance.logger import logging
from typing import Optional
import os,sys
import xgboost as xg
from sklearn.linear_model import LinearRegression
from Insurance import utils
from sklearn.metrics import r2_score


class ModelTrainer:


    def __init__(self,model_trainer_config:config_entity.ModelTrainerConfig,
                 data_transformation_artifact:artifact_entity.DataTransformation
Artifact
                 ):
        try:
            logging.info(f"{'>>'*20} Model Trainer {'<<'*20}")
            self.model_trainer_config=model_trainer_config
            self.data_transformation_artifact=data_transformation_artifact

        except Exception as e:
            raise InsuranceException(e, sys)

    def fine_tune(self):
        try:
            #Wite code for Grid Search CV
            pass


        except Exception as e:
            raise InsuranceException(e, sys)

    def train_model(self,x,y):
        # try:
        #     xgb_r = xg.XGBRegressor()
        #     xgb_r.fit(x,y)
        #     return xgb_r
        # except Exception as e:
        #     raise InsuranceException(e, sys)

        try:
            lr = LinearRegression()
            lr.fit(x,y)
            return lr
        except Exception as e:
            raise InsuranceException(e, sys)
```

```
    def initiate_model_trainer(self,)->artifact_entity.ModelTrainerArtifact:
        try:
            logging.info(f"Loading train and test array.")
            train_arr = utils.load_numpy_array_data(file_path=self.data_transfo
rmation_artifact.transformed_train_path)
            test_arr = utils.load_numpy_array_data(file_path=self.data_transfor
mation_artifact.transformed_test_path)

            logging.info(f"Splitting input and target feature from both train a
nd test arr.")
            x_train,y_train = train_arr[:,:-1],train_arr[:,-1]
            x_test,y_test = test_arr[:,:-1],test_arr[:,-1]

            logging.info(f"Train the model")
            model = self.train_model(x=x_train,y=y_train)

            logging.info(f"Calculating f1 train score")
            yhat_train = model.predict(x_train)
            r2_train_score  =r2_score(y_true=y_train, y_pred=yhat_train)

            logging.info(f"Calculating f1 test score")
            yhat_test = model.predict(x_test)
            r2_test_score  =r2_score(y_true=y_test, y_pred=yhat_test)

            logging.info(f"train score:{r2_train_score} and tests score {r2_tes
t_score}")
            #check for overfitting or underfiiting or expected score
            logging.info(f"Checking if our model is underfitting or not")
            if r2_test_score<self.model_trainer_config.expected_score:
                raise Exception(f"Model is not good as it is not able to give \
                expected accuracy: {self.model_trainer_config.expected_score}:
model actual score: {r2_test_score}")

            logging.info(f"Checking if our model is overfiiting or not")
            diff = abs(r2_train_score-r2_test_score)

            if diff>self.model_trainer_config.overfitting_threshold:
                raise Exception(f"Train and test score diff: {diff} is more tha
n overfitting threshold {self.model_trainer_config.overfitting_threshold}")

            #save the trained model
            logging.info(f"Saving mode object")
            utils.save_object(file_path=self.model_trainer_config.model_path, o
bj=model)

            #prepare artifact
            logging.info(f"Prepare the artifact")
            model_trainer_artifact  = artifact_entity.ModelTrainerArtifact(mode
l_path=self.model_trainer_config.model_path,
            r2_train_score=r2_train_score, r2_test_score=r2_test_score)
```

```
            logging.info(f"Model trainer artifact: {model_trainer_artifact}")
            return model_trainer_artifact
        except Exception as e:
            raise InsuranceException(e, sys)
```

GitHub Commands

```
git add .
git commit -m "data componenets"
git push -u origin main
```

6 Part - 6

6.1 Update `main.py` file of Insurance folder

6.2 Run `main.py` file

6.3 Create a file `predictor.py` in `Insurance` folder write

the following code

```
import os
from Insurance.entity.config_entity import TRANSFORMER_OBJECT_FILE_NAME,MODEL_F
ILE_NAME,TARGET_ENCODER_OBJECT_FILE_NAME
from glob import glob
from typing import Optional
import os

# Now lets start model validation


class ModelResolver:

    def __init__(self,model_registry:str = "saved_models",
                 transformer_dir_name="transformer",
                 target_encoder_dir_name = "target_encoder",
                 model_dir_name = "model"):

        self.model_registry=model_registry
        os.makedirs(self.model_registry,exist_ok=True)
        self.transformer_dir_name = transformer_dir_name
        self.target_encoder_dir_name=target_encoder_dir_name
        self.model_dir_name=model_dir_name

# 1
    def get_latest_dir_path(self)->Optional[str]:
        try:
            dir_names = os.listdir(self.model_registry)
            if len(dir_names)==0:
                return None
            dir_names = list(map(int,dir_names))
            latest_dir_name = max(dir_names)
            return os.path.join(self.model_registry,f"{latest_dir_name}")
        except Exception as e:
            raise e

# 2

    def get_latest_model_path(self):
        try:
            latest_dir = self.get_latest_dir_path()
            if latest_dir is None:
                raise Exception(f"Model is not available")
            return os.path.join(latest_dir,self.model_dir_name,MODEL_FILE_NAME)
        except Exception as e:
            raise e
# 3
    def get_latest_transformer_path(self):
        try:
            latest_dir = self.get_latest_dir_path()
            if latest_dir is None:
```

```
                raise Exception(f"Transformer is not available")
            return os.path.join(latest_dir,self.transformer_dir_name,TRANSFORME
R_OBJECT_FILE_NAME)
        except Exception as e:
            raise e
# 4
    def get_latest_target_encoder_path(self):
        try:
            latest_dir = self.get_latest_dir_path()
            if latest_dir is None:
                raise Exception(f"Target encoder is not available")
            return os.path.join(latest_dir,self.target_encoder_dir_name,TARGET_
ENCODER_OBJECT_FILE_NAME)
        except Exception as e:
            raise e


# 5
    def get_latest_save_dir_path(self)->str:
        try:
            latest_dir = self.get_latest_dir_path()
            if latest_dir==None:
                return os.path.join(self.model_registry,f"{0}")
            latest_dir_num = int(os.path.basename(self.get_latest_dir_path()))
            return os.path.join(self.model_registry,f"{latest_dir_num+1}")
        except Exception as e:
            raise e
# 6
    def get_latest_save_model_path(self):
        try:
            latest_dir = self.get_latest_save_dir_path()
            return os.path.join(latest_dir,self.model_dir_name,MODEL_FILE_NAME)
        except Exception as e:
            raise e
# 7
    def get_latest_save_transformer_path(self):
        try:
            latest_dir = self.get_latest_save_dir_path()
            return os.path.join(latest_dir,self.transformer_dir_name,TRANSFORME
R_OBJECT_FILE_NAME)
        except Exception as e:
            raise e
# 8
    def get_latest_save_target_encoder_path(self):
        try:
            latest_dir = self.get_latest_save_dir_path()
            return os.path.join(latest_dir,self.target_encoder_dir_name,TARGET_
ENCODER_OBJECT_FILE_NAME)
        except Exception as e:
            raise e
```

6.4 Rum `main.py` file

Part - 7

GitHub Commands

```
git add .
git commit -m "Updated predictor file"
git push -u origin main
```

8 Part - 8

8.1 Crete a `batch_prediction.py` in pipeline folder of

Insurance folder

```
import numpy as np
from Insurance.exception import InsuranceException
from Insurance.logger import logging
from Insurance.predictor import ModelResolver
import pandas as pd
from Insurance.utils import load_object
import os
import sys
from datetime import datetime
PREDICTION_DIR = "prediction"


def start_batch_prediction(input_file_path):
    try:
        os.makedirs(PREDICTION_DIR, exist_ok=True)
        logging.info(f"Creating model resolver object")
        model_resolver = ModelResolver(model_registry="saved_models")
        logging.info(f"Reading file :{input_file_path}")
        df = pd.read_csv(input_file_path)
        df.replace({"na": np.NAN}, inplace=True)
        # validation

        logging.info(f"Loading transformer to transform dataset")
        transformer = load_object(
            file_path=model_resolver.get_latest_transformer_path())


        logging.info(f"Target encoder to convert predicted column into categori
cal")
        target_encoder = load_object(file_path=model_resolver.get_latest_target
_encoder_path())


        """We need to create label encoder object for each categorical variabl
e. We will check later"""
        input_feature_names = list(transformer.feature_names_in_)
        for i in input_feature_names:
            if df[i].dtypes =='object':
                df[i] =target_encoder.fit_transform(df[i])

        input_arr = transformer.transform(df[input_feature_names])

        logging.info(f"Loading model to make prediction")
        model = load_object(file_path=model_resolver.get_latest_model_path())
        prediction = model.predict(input_arr)


        # cat_prediction = target_encoder.inverse_transform(prediction)

        df["prediction"]=prediction
```

```
            # df["cat_pred"]=cat_prediction


            prediction_file_name = os.path.basename(input_file_path).replace(".cs
v",f"{datetime.now().strftime('%m%d%Y__%H%M%S')}.csv")
            prediction_file_path = os.path.join(PREDICTION_DIR,prediction_file_nam
e)
            df.to_csv(prediction_file_path,index=False,header=True)
            return prediction_file_path
        except Exception as e:
            raise InsuranceException(e, sys)
```

8.2 Update `data_prediction.py` file

8.3 Run `main.py` file

8.4 Create `demo.py` file and write the following code

```
from Insurance.pipeline.batch_prediction import start_batch_prediction
from Insurance.pipeline.training_pipeline import start_training_pipeline

file_path=r"E:/wasiq/Insurance_Project/insurance.csv"
print(__name__)
if __name__=="__main__":
    try:
        #output_file = start_training_pipeline()
        output_file = start_batch_prediction(input_file_path=file_path)
        print(output_file)
    except Exception as e:
        print(e)
```

8.4.1 Run `demo.py` file

8.5 Create `training_pipeline.py` file in the pipeline

folder of Insurance and write the following code

```
from Insurance.logger import logging
from Insurance.exception import InsuranceException
from Insurance.utils import get_collection_as_dataframe
import sys,os
from Insurance.entity import config_entity
from Insurance.components.data_ingestion import DataIngestion
from Insurance.components.data_validation import DataValidation
from Insurance.components.data_transformation import DataTransformation
from Insurance.components.model_trainer import ModelTrainer
from Insurance.components.model_evaluation import ModelEvaluation
from Insurance.components.model_pusher import ModelPusher


def start_training_pipeline():
    try:
        training_pipeline_config = config_entity.TrainingPipelineConfig()

        #data ingestion
        data_ingestion_config  = config_entity.DataIngestionConfig(training_pip
eline_config=training_pipeline_config)
        print(data_ingestion_config.to_dict())
        data_ingestion = DataIngestion(data_ingestion_config=data_ingestion_con
fig)
        data_ingestion_artifact = data_ingestion.initiate_data_ingestion()

        #data validation
        data_validation_config = config_entity.DataValidationConfig(training_pi
peline_config=training_pipeline_config)
        data_validation = DataValidation(data_validation_config=data_validation
_config,
                         data_ingestion_artifact=data_ingestion_artifact)

        data_validation_artifact = data_validation.initiate_data_validation()

        #data transformation
        data_transformation_config = config_entity.DataTransformationConfig(tra
ining_pipeline_config=training_pipeline_config)
        data_transformation = DataTransformation(data_transformation_config=dat
a_transformation_config,
        data_ingestion_artifact=data_ingestion_artifact)
        data_transformation_artifact = data_transformation.initiate_data_transf
ormation()

        #model trainer
        model_trainer_config = config_entity.ModelTrainerConfig(training_pipeli
ne_config=training_pipeline_config)
        model_trainer = ModelTrainer(model_trainer_config=model_trainer_config,
data_transformation_artifact=data_transformation_artifact)
        model_trainer_artifact = model_trainer.initiate_model_trainer()
```

```
        #model evaluation
        model_eval_config = config_entity.ModelEvaluationConfig(training_pipeli
ne_config=training_pipeline_config)
        model_eval  = ModelEvaluation(model_eval_config=model_eval_config,
        data_ingestion_artifact=data_ingestion_artifact,
        data_transformation_artifact=data_transformation_artifact,
        model_trainer_artifact=model_trainer_artifact)
        model_eval_artifact = model_eval.initiate_model_evaluation()

        #model pusher
        model_pusher_config = config_entity.ModelPusherConfig(training_pipeline
_config)
        model_pusher = ModelPusher(model_pusher_config=model_pusher_config,
                data_transformation_artifact=data_transformation_artifact,
                model_trainer_artifact=model_trainer_artifact)
        model_pusher_artifact = model_pusher.initiate_model_pusher()
    except Exception as e:
        raise InsuranceException(e, sys)
```

9 Streamlit code for Application

9.1 Create a file app.py

```
import streamlit as st
# import preprocessor,helper
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import pickle
import xgboost as xg
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

model = pickle.load(open('model_final.pkl','rb'))
encoder = pickle.load(open('target_encoder.pkl','rb'))
transformer = pickle.load(open('transformer.pkl','rb'))

st.title("Insurance Premium Prediction")
age = st.text_input('Enter Age', 18)
age = int(age)

sex = st.selectbox(
    'Please select gender',
    ('male', 'female'))
# gender = encoder.transform(np.array([sex]))

bmi = st.text_input('Enter BMI', 18)
bmi = float(bmi)

children = st.selectbox(
    'Please select number of children ',
    (0,1,2,3,4,5))
children = int(children)


smoker = st.selectbox(
    'Please select smoker category ',
    ("yes","no"))
# smoker = encoder.transform(smoker)

region = st.selectbox(
    'Please select region ',
    ("southwest", "southeast", "northeast", "northwest"))


l = {}
l['age'] = age
l['sex'] = sex
l['bmi'] = bmi
l['children'] = children
l['smoker'] = smoker
l['region'] = region
```

```
df = pd.DataFrame(l, index=[0])

df['region'] = encoder.transform(df['region'])
df['sex'] = df['sex'].map({'male':1, 'female':0})
df['smoker'] = df['smoker'].map({'yes':1, 'no':0})

df = transformer.transform(df)
# dtrain = xg.DMatrix(df)
y_pred = model.predict(df)
# st.write(age, gender, bmi, children, smoker, region)

if st.button("Show Result"):
    # col1,col2, col3,col4 = st.columns(4)
    st.header(f"{round(y_pred[0],2)} INR")
```