**MODULAR CODING**

**components**

1. **Data Ingestion**
2. **Data Validation**
3. **Data Transformation**
4. **Model Trainer**
5. **Model Evaluation**
6. **Model pusher**
7. **Prediction Pipeline**
8. **User APP**
9. **Data Ingestion**

**Workflow**

1. Us-visa 🡪 constants 🡪 copy the code

For all the components you need to assigned in the constant folder , so that instead of changing it in each file , we can change it from the single file

File contain important variables

1. In us-visa 🡪 configuration 🡪 create a mongo db connection.py file

All the folder and file path should be managed in the config

1. Us visa 🡪 create folder data\_access 🡪 create \_\_init\_\_.py and usvisa\_data.py 🡪
2. Us visa 🡪 entity🡪 config entity🡪
3. Us visa 🡪 entity 🡪 artifact (return type of the component . Here : data ingestion)🡪

Output from the components

Here train.py and test.py

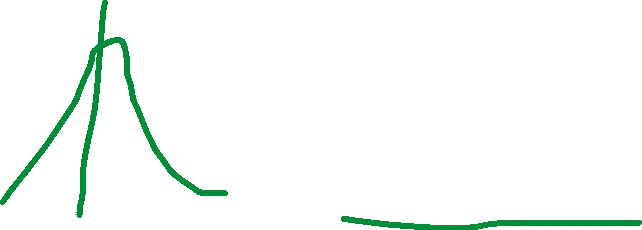
1. Us visa 🡪 Data ingestion.py 🡪
2. Us visa 🡪 components 🡪 Pipeline🡪Training\_pipeline
3. Before that set the MONGODB\_URL in env variable or set MONGODB\_URL=”-------------"
4. End point

**Data Drift**

Data drift, also known as concept drift or dataset shift, refers to the gradual or abrupt change in the statistical properties of the data used to train a machine learning model. These statistical properties can include changes in the distribution of features, target labels, or the relationships between them.



No drift occurs



Drift occurs

Training and testing data should be follows the same distribution . To detect the data drift we can use the MLOPs tool Evidently AI

Train data:Old data Test data : new data

1. **Data Validation**

* In schema.yaml file , we need to mention the data schema like the column name , data types (meta data )
* After that drift prediction occurs and the status is stored as a yaml file
* In schema.yaml file ,

Mention all the features like columns, numercal columns , categorical columns , drop\_columns , numerical features , ordinal encoding , onehot encoding , transform columns

* Yaml file is used to manage the configuration

1. Us – visa 🡪 constants
2. Us visa 🡪 entity 🡪 config entity.py
3. Us Visa 🡪 entity 🡪 artifact entity

* It returns the Boolean value ie., True or False
* It also returns the path of the report file

1. Us visa 🡪 components 🡪 data validation 🡪
2. Us visa 🡪 components 🡪 Pipeline🡪Training\_pipeline
3. Update the main\_utils .py file
4. And schema file before that
5. End point
6. **Data Transformation**
7. Us visa 🡪 Constants
8. Us visa 🡪 entity 🡪 config entity.py
9. Us visa 🡪 entity 🡪 artifact entity.py

It returns 3 files

* train.npy
* test.npy
* preprocessor.pkl

Now we need to convert the target variable (here category ) into number

1. Us visa 🡪 entity 🡪 create a file ‘estimator.py’
2. Us visa 🡪 components 🡪 data transformation
3. Us visa 🡪 pipeline

* Load the DataTransformation component
* “ “ DataTransformationconfig
* “ “ DataTransformationArtifact
* Initialize it in the class variable
* Define start\_data\_transformation
* Call data\_transformation inside the run pipeline

1. End point
2. **Model Trainer**

* All the parameters for the hyperparameter tuning
* And the best models are stored in the model.yaml file
* Update Yaml file ; you can mention multiple model and corresponding parameters

1. Us visa 🡪 Constants
2. Us visa 🡪 entity 🡪 config enitity.py
3. Us visa 🡪 entity 🡪artifact entity.py

* ClassificationMetricArtifact and then ModelTrainerArtifact
* Model.pkl file will be created

1. Us visa 🡪 entity 🡪 update estimator.py {add UsVisaModel}
2. Us visa 🡪 components 🡪 Model Trainer
3. Us visa 🡪 pipeline

* Load the ModelTrainer component
* “ “ ModelTrainerConfig
* “ “ ModelTrainer Artifact
* Initialize it in the class variable
* Define start\_Model\_Trainer
* Call model\_trainer inside the run pipeline

1. End point

**AWS setup**

1. Login to AWS 🡪 IAM 🡪Users 🡪 create user 🡪 <username:usvisaproject> next🡪 attach policies directly 🡪 Administrators access 🡪 next🡪create user
2. usvisaproject🡪security credentials 🡪 Access Keys 🡪 create access key 🡪 command Line Interface(CLI)🡪 Consent 🡪 next 🡪create access key🡪
   * Access key : ----------------------------------
   * Secret access key : -----------------------------------

🡪

Download .csv file

🡪Open the file in notepad 🡪

1. Us visa 🡪 constants 🡪 set the aws access key and secret key
2. Set AWS\_SECRET\_ACCESS\_KEY="----------------------------------------" in cmd
3. Set AWS\_ACCESS\_KEY\_ID="-----------------------------------------" in cmd

Or

1. Set the environment variable
2. Us visa 🡪 Configuration🡪 create a file “aws\_connection.py” {code used from google}
3. Us visa 🡪 constants 🡪 added some constants

Copy the MODEL\_BUCKET\_NAME

1. AWS🡪 S3🡪 create bucket 🡪provide bucket name : usvisa-model2024 🡪 unmark Block all public access 🡪 acknowledge it 🡪 create bucket

* Then the model will be pushed to this bucket

1. Us visa 🡪 create a folder “cloud\_storage”🡪 cloud\_storage 🡪 create \_\_init\_\_.py 🡪create aws\_storage.py
2. Us visa🡪 entity 🡪 create a file “s3\_estimator.py” 🡪code

* Whenever we do the model evaluation we should get\_the\_model

In s3\_estimator.py ;

* Upload\_file will be used for the pusher component
* Load\_model will be used for model evaluation

1. **Model Evaluation**

1. Us visa 🡪 constants

2. Us visa 🡪 entity 🡪 config.py

3. Us visa 🡪 entity 🡪 artifact.py

4. Us visa 🡪 components 🡪 model evaluation.py

5. Us visa 🡪 pipeline🡪 training pipeline

* Load the model evaluation component
* “ “ model evaluation Config
* “ “ model evaluation Artifact
* Initialize it in the class variable
* Define start\_ model-evaluation
* Call model\_evaluation inside the run pipeline

6.End point

1. **Model Pusher**
2. Us visa 🡪 entity 🡪 config.py
3. Us visa 🡪entity 🡪 artifact.py
4. Us visa 🡪 components 🡪 model pusher.py
5. Us visa 🡪 pipeline 🡪

* Load the model pusher component
* “ “ model pusher Config
* “ “ model pusher Artifact
* Initialize it in the class variable
* Define start\_ model-pusher
* Call model\_pusher inside the run pipeline

1. End point

**Now model.pkl is in S3 Bucket**

**Prediction Pipeline**

**User App**