End to End Machine Learning project

## Tutorial 1: GitHub & Code setup

* Steup a new environment
* Setup.py
* Requirements.txt
* Steps:

1. Create a new repository “mlproject”
2. Create a new folder in which entire project will be developed
3. Copy the path of the folder & open anaconda prompt
4. Code . this will open vscode
5. First we need to make sure that we are in sync with the GitHub repository
6. Open new terminal in vs code and create new environment

(select cmd in terminal and use below command)

conda create -p venv python==3.8 -y

activate the virtual environment

conda activate venv/

1. Clone the repository and sync with GitHub

* Initialize git repository (git init)
* Add README.md file, before adding create README.md file (either in git or in vs code) (in README.md file we can write descriptions, what are all the steps required)
* Add README.md file in GitHub repository

git add README.md

commit the file,

git commit -m "First commit"

* Push the file to git repository

git branch – M main

* Add origin (so it is in sync with the GitHub repository)

git remote add origin <https://github.com/AD/mlproject.git>

git remote – v

to push the data into github repo we need to set git global

(check git global config and add username and email of your own)

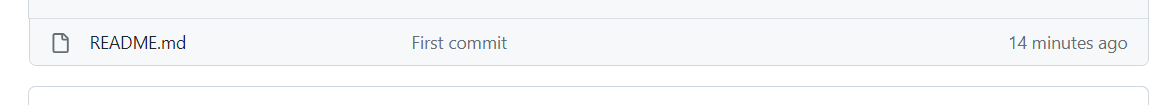
git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

* Push the data

git push -u origin main

by this now we can see the file in GitHub repository.



* Create a new .gitignore file in GitHub repository, select python and commit changes

git pull   
  
so all the updation will be done in vscode as well.

* Setup a setup.py and requirements.txt

(setup.py will be responsible for making or ML application as a package, which can be used further.)

from setuptools import find\_packages , setup #find all the packages

setup (

    name = "mlproject",

    version = "0.0.1",

    author= "yourname",

    author\_email= "youremail id",

    packages=find\_packages()

    install\_requires = ['pandas', 'numpy', 'seaborn '] #automatic installation can be done

)

* Create a new folder “src” in vs code (source) and inside it create \_\_init\_\_.py
* When in setup.py find\_packages() is running it will go and find in how many files have \_\_init\_\_.py running. Considering source as a package , it will try to build it.

1. We create a function , when we need to install many packages,

get\_requirements () #which will take requirements.txt it should be able to read all those files.

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()  #the line will get readed, but \n will get added

        requirements = [ req.replace("\n","") for req in requirements.txt]  #replace \n with blank

A screenshot of a computer

Description automatically generated

Here -e. will automatically trigger setup.py

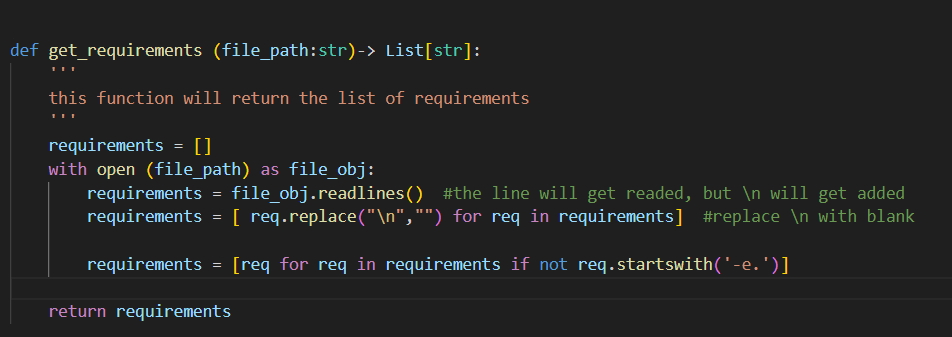
When we install requirements.txt the setup.py will also run.

A black background with white text

Description automatically generated

To solve the above problem use :

pip install -r requirements.txt



A screen shot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

Build is happening,

mlproject.egg-info will tell us the packages which are getting installed.

We have also created a source folder and build the packages.

Add the files – git add .

git commit -m “setup”

git push -u origin main

## Tutorial 2: Project structure, Logging, & Exception Handling

* Create components folder in “src” in that folder again create \_\_init\_\_.py file.
* Create data\_ingestion.py file (which will be for reading the data)
* Create data\_transformation.py file (for data transformation)
* Create model\_trainer.py file (for model training )
* Create pipeline folder in “src” and create train\_pipeline.py file
* Create “\_\_init\_\_.py” and “predict\_pipeline.py” file in pipeline folder.
* Create logger.py, exception.py and utils.py file in “src” folder.

(check exception handling python documentation)

* Create function & class of your won to do exception handling (exception.py)

#we are going to write our own exception

############ exception handling ############

import sys

#the sys module in python provides various functions and variables that are used

#  to manipulate different parts of the python runtime environment

def error\_message\_details(error, error\_detail: sys):  #error\_details will be present in sys

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

    file\_name = exe\_tb.tb\_frame.f\_code.co\_filename  #to get the filename

     #execution info, this will give us three info we are interested in the last one.

    #on which line exception has occured, in which file exception  has occured

    error\_message = "Error occured in python script name [{0}] line number [{1}] error message [{2}]".format(file\_name, exe\_tb.tb\_lineno, str(error))

    return error\_message

    #whenever error occurs we are going to call this function.

#created own exception class

class CustomException(Exception):

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

        #inheriting from exception

        super.\_\_init\_\_(error\_message) #inherit exception class

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

#whenever we raise custom exception, it is inheriting from parent exception, whatever error msg is coming,

# it will initialize and gets assigned to a class variable

    def \_\_str\_\_(self):

        return self.error\_message   #to print the error message is called.

* Similarly do in logger.py file.

Logger.py file will be used to log the information.

Create log file.

import logging

import os

from datetime import datetime

#create log file

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

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

#whatever logs will be created it will be respect to current working directory.

#logs folder will get created,

os.makedirs(logs\_path, exist\_ok=True) #even when there is file, keep on appending it.

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

#whenever we want to create the log, we have to set this up in basic config

#give the file name, where you want to store it

#format

#which level

logging.basicConfig(

    filename = LOG\_FILE\_PATH,

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

    level = logging.INFO,

)# any print msg will use this config, wrt msg

If \_\_name\_\_ == “\_\_main\_\_”:

Logging.info(“Logging has started”)

If we run the logger.py file, it gets executed successfully, and logs gets created as below”:

A screenshot of a computer

Description automatically generated

* Similarly check for exception.py file

(add below code in exception.py)

import logging

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

    try:

        a = 1/0

    except:

        logging.info("Divide by zero ")

        raise CustomException

when we run the code , an error message printed:

A screenshot of a computer program

Description automatically generated

Solved it by :

 try:

        a = 1/0

    except Exception as e:

        logging.info("Divide by zero ")

        raise CustomException(e, sys)

another error occurred:

A screen shot of a computer program

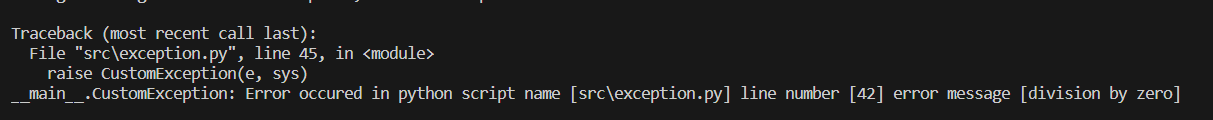
Description automatically generated

Changed super to super() as below

 super().\_\_init\_\_(error\_message) #inherit exception class

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

we get the following exception.



So the custom exception has been recorded in the logging.(check again)

We are getting the custom exception. (find why it is not logged in logging file)

git add .

git commit -m “project structure – logging and exception”

git push -u origin main

we were trying to raise an exception which was not getting stored, because we must import logging from logger.py file in exception.py file. If we import it and run the exception.py again, we get exception in the log folder.

## Tutorial 3: Project Problem Statement , EDA and Model Training

**Project : students’ performance indicator**

**Problem statement:** This project understands how the students' performance (test scores) is affected by other variables such as Gender, Ethnicity, Parental level of education, Lunch and Test preparation course

Data : students performance in Exams (8 columns 1000 rows)

All eda part must be done in jupyter notebook.

Add required libraries in requirement.txt

We complete the EDA and model training in jupyter notebooks.

**EDA code:**

# ###### import data and packages

# In[1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import warnings

warnings.filterwarnings('ignore')

# ##### import csv data as pandas dataframe

# In[2]:

df = pd.read\_csv("stud.csv")

df.head()

# In[3]:

df.shape

# ##### dataset information

# - gender : sex of student -> (Male/Female)

# - race/ ethnicity : ethnicity of students -> (Group A, B, C, D, E)

# - parental level of education : parents final education -> (bachelors degree, some college, masters degree, associates degree, high school)

# - lunch : having lunch before test (standard or free/reduced)

# - test preparation course : complete or not complete before the test

# - math score

# - reading score

# - writing score

# ##### Data checks to perform

# - check missing values

# - check duplicates

# - check data types

# - check number of unique values for each column

# - check statistics of dataset

# - check various various categories present in the different categorical column

#

# In[4]:

df.isna().sum()

# In[5]:

#there are no missing values in the dataset

# In[7]:

df.duplicated().sum() #no duplicate values

# In[8]:

df.info() #check null and dtypes

# In[9]:

df.nunique() #check unique values

# In[10]:

df.describe()

# #### insights

# - from above description of numerical data, all means are very close to each other.between 66-68

# - all standard deviations are close - between 14.6 and 15.9

# - while there is a minimum score 0 for math, for writing it is 10, for reading it is 17

# #### Exploring data

# In[15]:

print("categories in 'gender' variable : '", end = " ")

print(df['gender'].unique())

print("categories in 'race/ethnicity' variable : '", end = " ")

print(df['race/ethnicity'].unique())

print("categories in 'parental level of education' variable : '", end = " ")

print(df['parental level of education'].unique())

print("categories in 'lunch'variable : '", end = " ")

print(df['lunch'].unique())

print("categories in 'test preparation course 'variable : '", end = " ")

print(df['test preparation course'].unique())

# In[16]:

#define numerical / categorical features

numeric\_features = [feature for feature in df.columns if df[feature].dtype != 'O']

categorical\_features = [feature for feature in df.columns if df[feature].dtype == 'O']

#print columns

print('we have {} numerical features : {}'.format(len(numeric\_features),numeric\_features))

print('\nwe have {} categorical features : {}'.format(len(categorical\_features),categorical\_features))

# #### Adding columns "total score" & "average"

# In[17]:

df['total score'] = df['math score'] + df['reading score'] + df['writing score']

df['Average'] = df['total score']/3

df.head()

#this is kind of feature engineering we are doing.

# In[19]:

reading\_full = df[df['reading score'] == 100]['Average'].count()

writing\_full = df[df['writing score'] ==100]['Average'].count()

math\_full = df[df['math score'] ==100 ]['Average'].count()

print("Number of students with full marks in math ", math\_full)

print("Number of students with full marks in writing ", writing\_full)

print("Number of students with full marks in reading ", reading\_full)

# In[21]:

reading\_less\_20 = df[df['reading score'] <=20]['Average'].count()

writing\_less\_20 = df[df['writing score'] <=20]['Average'].count()

math\_less\_20 = df[df['math score'] <=20]['Average'].count()

print("Number of students with less than 20 marks in math ", math\_less\_20)

print("Number of students with less than 20 marks in writing ", writing\_less\_20)

print("Number of students with less than 20 marks in reading ", reading\_less\_20)

# #### insights

# - From above valueswe get students have performed the worst in Maths

# - Best performance in reading section

# #### Exploring Data (Visualization)

# In[23]:

#visualize average score distribution to make some conclusions

# - Histogram

# - Kernel Distrubution Function

# In[24]:

fig, axs = plt.subplots(1,2, figsize = (15,7))

plt.subplot(121)

sns.histplot(data = df, x= "Average", bins = 30, kde = True, color = "g")

plt.subplot(122)

sns.histplot(data = df, x = "Average", kde = True, hue = "gender")

plt.show()

# ##### female students tend to perform well than male students

# In[25]:

fig, axs = plt.subplots(1,2, figsize = (15,7))

plt.subplot(121)

sns.histplot(data = df, x= "total score", bins = 30, kde = True, color = "g")

plt.subplot(122)

sns.histplot(data = df, x = "total score", kde = True, hue = "gender")

plt.show()

# In[26]:

plt.subplots(1,3, figsize = (25,6))

plt.subplot(141)

sns.histplot(data = df, x = "Average", kde = True, hue = "lunch")

plt.subplot(142)

sns.histplot(data = df[df.gender == "female"], x = "Average", kde = True, hue = "lunch")

plt.subplot(143)

sns.histplot(data = df[df.gender == "male"], x = "Average", kde = True, hue = "lunch")

plt.show()

# #### insights

# - standard lunch is more effective

**Model training code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.svm import SVR

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error

from sklearn.model\_selection import RandomizedSearchCV

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

# In[2]:

df = pd.read\_csv("stud.csv")

df.head()

# In[3]:

#preparing X & y columns

X = df.drop(columns = ['math score'], axis = 1)

y = df['math score']

# In[4]:

X.head()

# In[5]:

#create column transformer with 3 types of transformers

num\_features = X.select\_dtypes(exclude = "object").columns

cat\_features = X.select\_dtypes(include = "object").columns

# In[6]:

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

# In[12]:

numeric\_transformer = StandardScaler()

oh\_transformer = OneHotEncoder()

preprocessor = ColumnTransformer(

[

("OneHotEncoder", oh\_transformer, cat\_features),

("StandardScaler", numeric\_transformer, num\_features)

]

)

#this is a pipeline

#here columntransformer will combine one hot encoder

# In[13]:

X = preprocessor.fit\_transform(X)

# In[15]:

#separate dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train.shape, y\_train.shape

# In[16]:

X

# # Create an evaluate Function to give all metrics after model training

# In[17]:

def evaluate\_model(true, predicted):

mae = mean\_absolute\_error(true, predicted)

mse = mean\_squared\_error(true, predicted)

rmse = np.sqrt(mean\_squared\_error(true, predicted))

r2\_square = r2\_score(true, predicted)

return mae, rmse, r2\_square

# In[19]:

models = {

"Linear Regression" : LinearRegression(),

"Lasso" : Lasso(),

"Ridge" : Ridge(),

"Decision Tree" : DecisionTreeRegressor(),

"Random Forest Regressor" : RandomForestRegressor()

}

model\_list = []

r2\_list = []

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

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

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

#make predictions

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

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

#Evaluate Train and Test dataset

model\_train\_mae, model\_train\_rmse, model\_train\_r2 = evaluate\_model(y\_train, y\_train\_pred)

model\_test\_mae , model\_test\_rmse, model\_test\_r2 = evaluate\_model(y\_test, y\_test\_pred)

print(list(models.keys())[i])

model\_list.append(list(models.keys())[i])

print("Model performance for Training set")

print("- Root Mean Squared Error : {:.4f}".format(model\_train\_rmse))

print("- Mean Absoulte Error : {:.4f}".format(model\_train\_mae))

print("- R2 Score: {:.4f} ".format(model\_train\_r2))

print("--------------------------------------")

print("Model performance for Test set")

print("- Root Mean Squared Error : {:.4f}".format(model\_test\_rmse))

print("- Mean Absoulte Error : {:.4f}".format(model\_test\_mae))

print("- R2 Score: {:.4f} ".format(model\_test\_r2))

r2\_list.append(model\_test\_r2)

print("\*"\*35)

print("\n")

# Results

# In[20]:

pd.DataFrame(list(zip(model\_list,r2\_list)), columns = ['Model Name', 'R2\_score']).sort\_values(by= ['R2\_score'], ascending=False)

# In[21]:

#Linear Regression

lin\_model = LinearRegression(fit\_intercept=True)

lin\_model = lin\_model.fit(X\_train, y\_train)

y\_pred = lin\_model.predict(X\_test)

score = r2\_score(y\_test, y\_pred)\*100

print("Accuracy of the model is ", score)

# In[22]:

#plot y\_pred and y\_test

plt.scatter(y\_test, y\_pred);

plt.xlabel("Actual");

plt.ylabel('predicted');

# In[23]:

sns.regplot(x=y\_test, y= y\_pred, ci = None, color = "red")

# In[24]:

#difference between actual and predicted values

pred\_df = pd.DataFrame({"Actual value": y\_test, "predicted value ": y\_pred, "Difference ": y\_test-y\_pred})

pred\_df

## Tutorial 4: Data Ingestion Implementation

* Data\_ingestion.py will play an important role.
* Data sources can be of various kind, from which we need to extract the data.

