Practical Work Report
Topics in Data Science

<AI model to recognize customized handwritten>

Student: Muhammet Ali YILDIZ , 1709981,

https://github.com/mtech00/dst_1709981_2024.git

,mtech00, yldzmuhammedali@gmail.com
Muhammet-Ali-YILDIZ-1709981

Course:Datascience and Artificial intelligence

1. Job description

1. **Image Processing**: The project begins with loading an image file using the Python Imaging Library (PIL). The image is then resized to a specified dimension using the **resize()** function. Additionally, the image is split into smaller segments using a custom function (**split_image**) to facilitate further processing.

2. **Data Conversion**: After splitting the image, each segment is converted into numerical format representing RGB values. This conversion is crucial for feeding image data into machine learning models. The numerical data is saved as CSV files for easy access and manipulation.
3. **Data Preparation**: The converted CSV files are then prepared for model training and evaluation. The data is split into training and testing datasets using a predefined ratio. This ensures that the model is trained on a portion of the data and tested on unseen samples to assess its generalization ability.
4. **Model Training**: A Decision Tree Regressor model is chosen for its simplicity and interpretability. The model is trained using the training dataset, where features represent RGB values of image segments, and the target variable is derived from the image data.
5. **Model Evaluation**: The trained model's performance is evaluated using the testing dataset. Metrics such as accuracy are calculated to assess how well the model predicts the target variable. This step is crucial for understanding the model's effectiveness and potential areas for improvement.

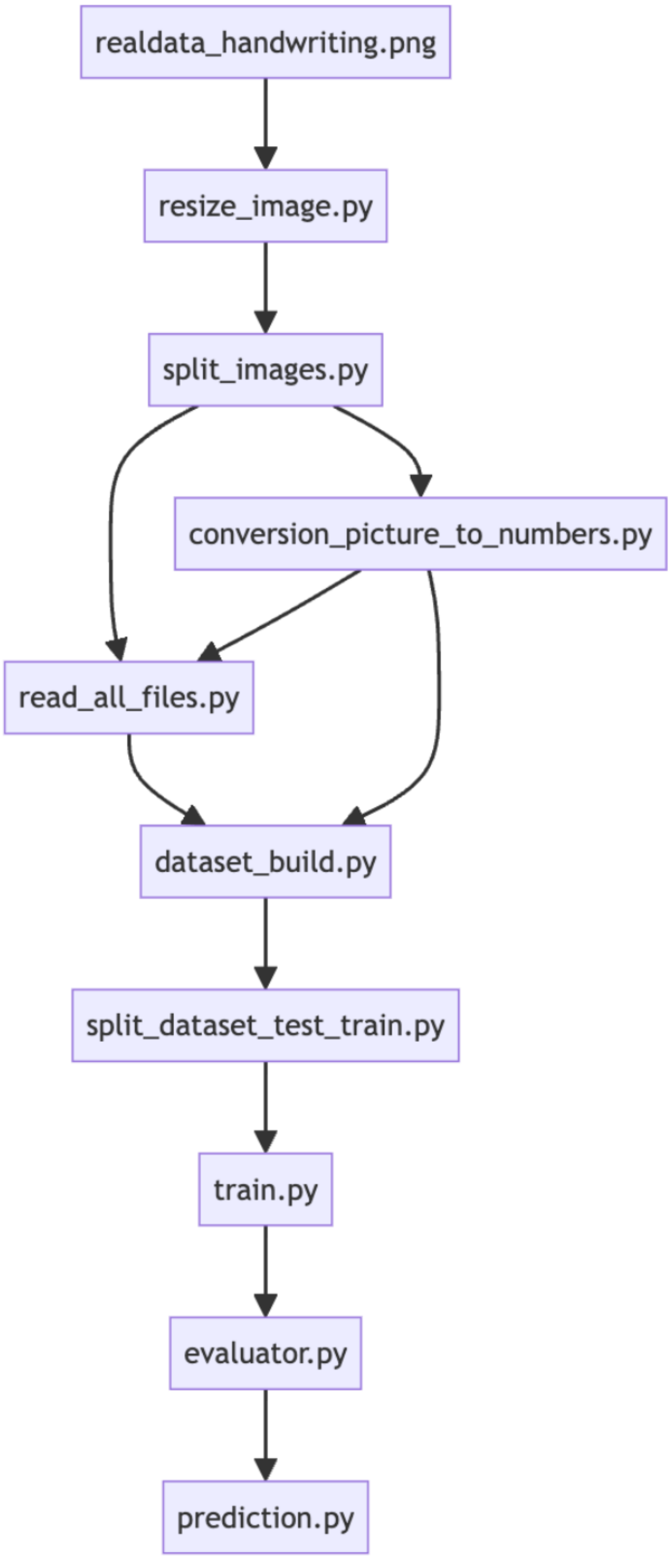
The practical work aims to develop an AI model capable of recognizing customized handwritten text. The project involves the implementation of various data science tasks, including image processing, data conversion, model training, and evaluation. The architectural drawing depicts the workflow of the project, detailing the sequence of tasks and scripts used.

2. Work Implementation

The implementation of the work involves leveraging Python libraries for image processing and machine learning tasks. The project begins with loading and resizing the input image using the Python Imaging Library (PIL). The resized image is then split into smaller segments to facilitate further processing. Each segment is converted into numerical format representing RGB values and saved as CSV files for model training. The data is then prepared, and a Decision Tree Regressor model is trained using the RGB values of image segments as features. Model evaluation metrics such as accuracy are calculated to assess the model's performance.

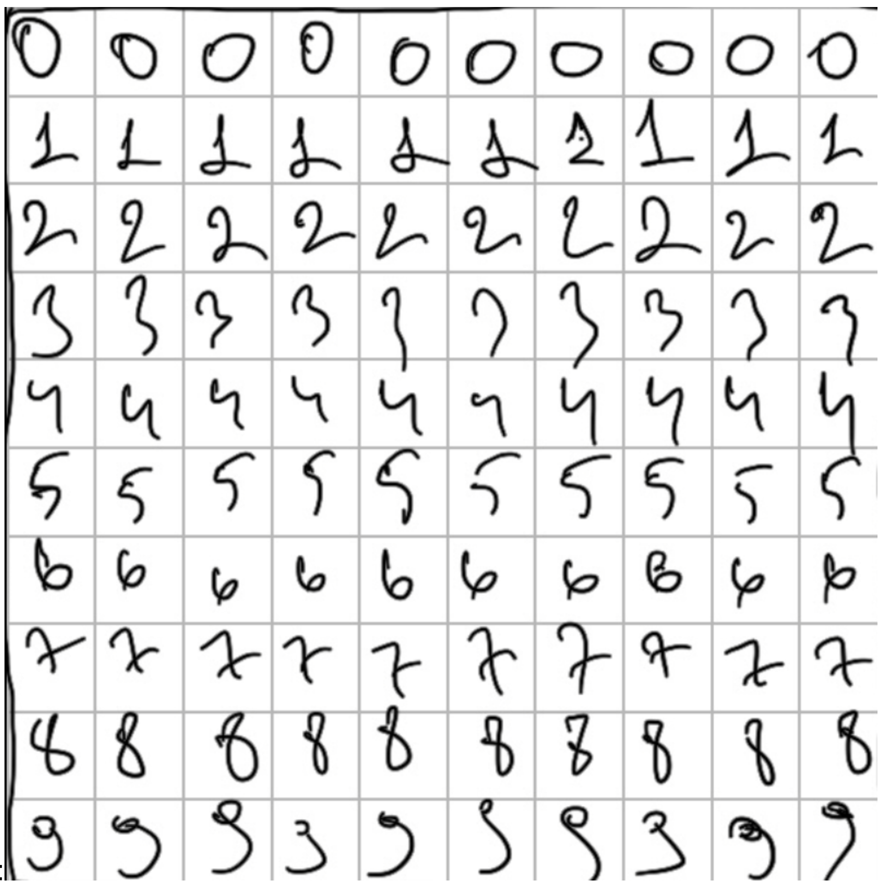
1.Image Processing: The image processing stage begins with resizing the original image to a standard dimension to ensure consistency in input data. Subsequently, the resized image is split into smaller segments using a custom function that divides the image into a grid of specified rows and columns.
2. Data Conversion: Each segmented image is converted into a numerical format representing RGB values. This conversion transforms image data into a structured format suitable for machine learning algorithms. The converted data is then saved as CSV files for further analysis and model training.
3. Data Preparation: The converted CSV files are organized into training and testing datasets. The training dataset is used to train the model, while the testing dataset is reserved for evaluating the model's performance. This step ensures unbiased assessment and generalization of the model.
4. Model Training: The Decision Tree Regressor model is trained using the training dataset. The model learns patterns from the input features (RGB values) and predicts the target variable. Training involves optimizing model parameters to minimize prediction errors and improve overall performance.
5. Model Evaluation: The trained model is evaluated using the testing dataset to assess its predictive accuracy and generalization ability. Evaluation metrics such as mean squared

Architectural Design Diagram

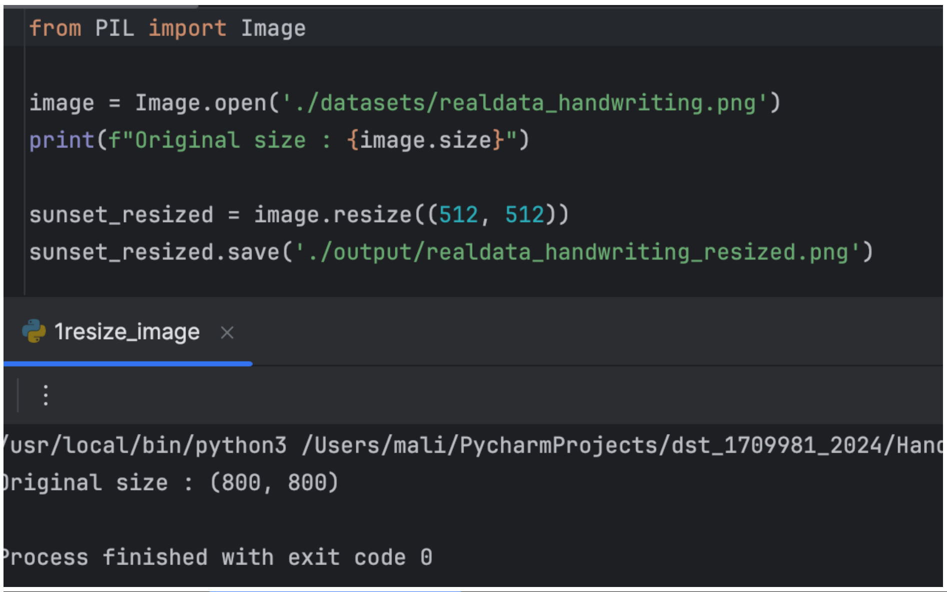

realdata_handwriting.png
resize_image.py
split_images.py
conversion_picture_to_numbers.py
read_all_files.py
dataset_build.py
split_dataset_test_train.py
train.py
evaluator.py
prediction.py

3. Work operation

The work operates by executing the implemented scripts on sample handwritten images. The image processing scripts resize and split the input image, while the conversion script converts the segmented images into numerical format. The dataset building script prepares the data for model training, followed by the training script, which trains the Decision Tree Regressor model. Finally, the evaluation script assesses the model's performance using testing data.

Original image :

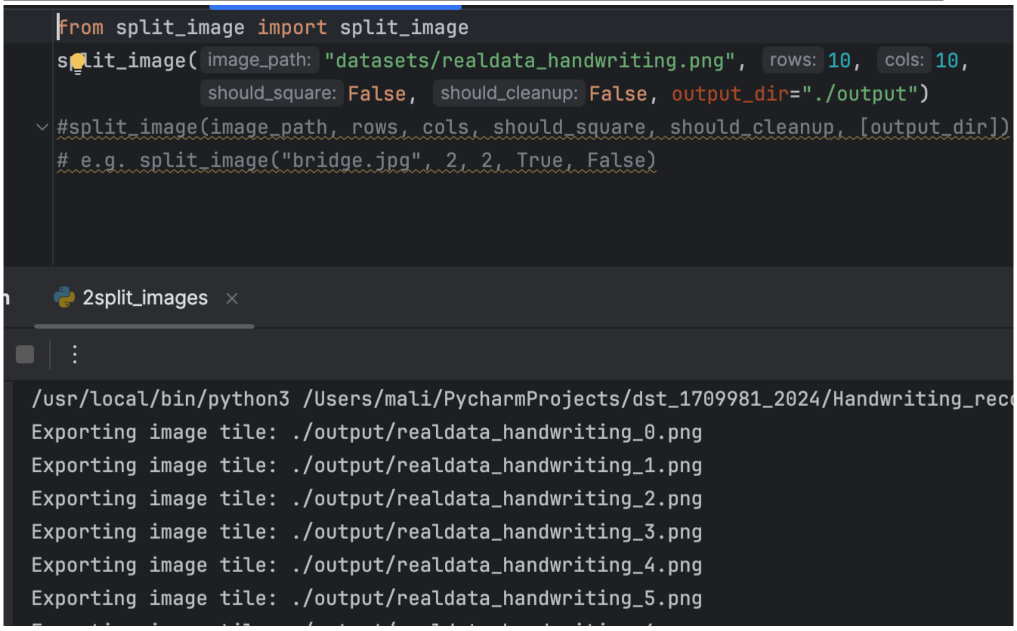
1:



```
from PIL import Image

image = Image.open('./datasets/realdata_handwriting.png')
print(f"Original size : {image.size}")

sunset_resized = image.resize((512, 512))
sunset_resized.save('./output/realdata_handwriting_resized.png')
```

1resize_image

```
/usr/local/bin/python3 /Users/mali/PycharmProjects/dst_1709981_2024/Hand
Original size : (800, 800)

Process finished with exit code 0
```



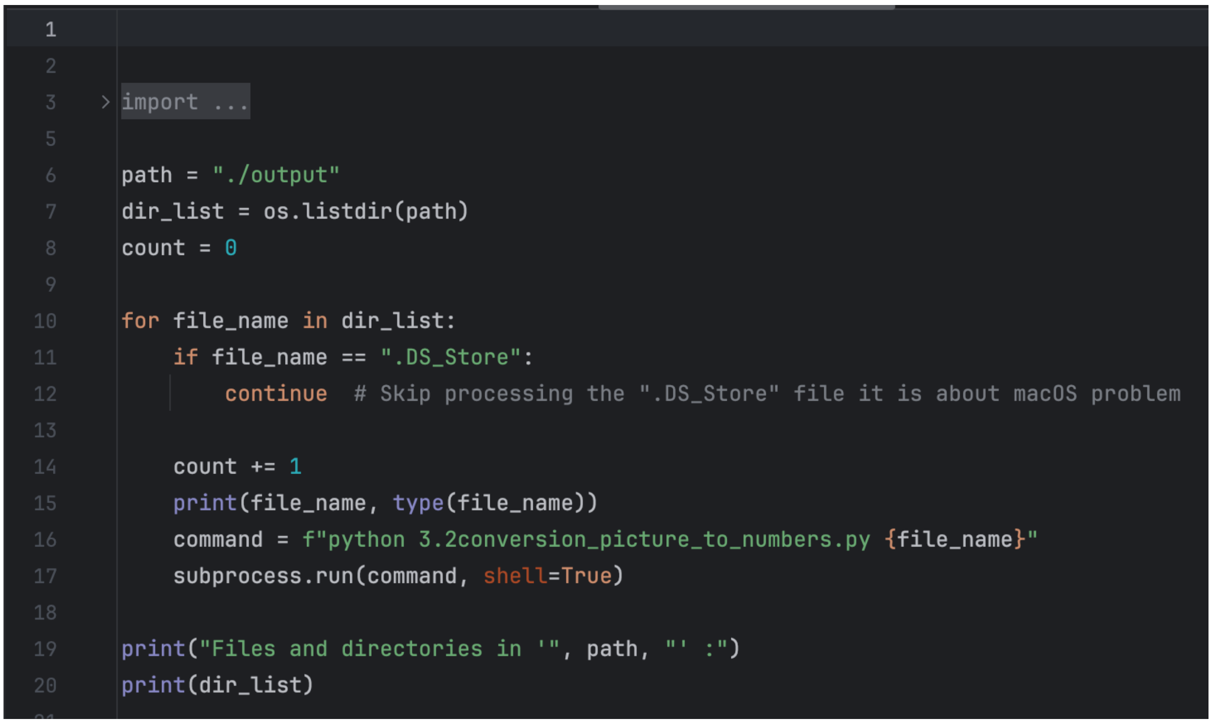
```
from split_image import split_image
split_image( image_path: "datasets/realdata_handwriting.png",  rows: 10,  cols: 10,
              should_square: False,  should_cleanup: False, output_dir="./output")
#split_image(image_path, rows, cols, should_square, should_cleanup, [output_dir])
# e.g. split_image("bridge.jpg", 2, 2, True, False)
```

2split_images

```
/usr/local/bin/python3 /Users/mali/PycharmProjects/dst_1709981_2024/Handwriting_rec
Exporting image tile: ./output/realdata_handwriting_0.png
Exporting image tile: ./output/realdata_handwriting_1.png
Exporting image tile: ./output/realdata_handwriting_2.png
Exporting image tile: ./output/realdata_handwriting_3.png
Exporting image tile: ./output/realdata_handwriting_4.png
Exporting image tile: ./output/realdata_handwriting_5.png
```

- This code section uses the Python Imaging Library (PIL) to resize an image.
- It opens an image file named "realdata_handwriting.png" from the './datasets/' directory.
- The original size of the image is printed.
- The image is resized to 512x512 pixels using the resize() function.
- The resized image is saved as "realdata_handwriting_resized.png" in the './output/' directory.
- This code section splits the original image into smaller segments.
- It utilizes a custom function split_image imported from the 'split_image' module.
- The function parameters specify the input image path, number of rows and columns for segmentation, and output directory for saving the segmented images.

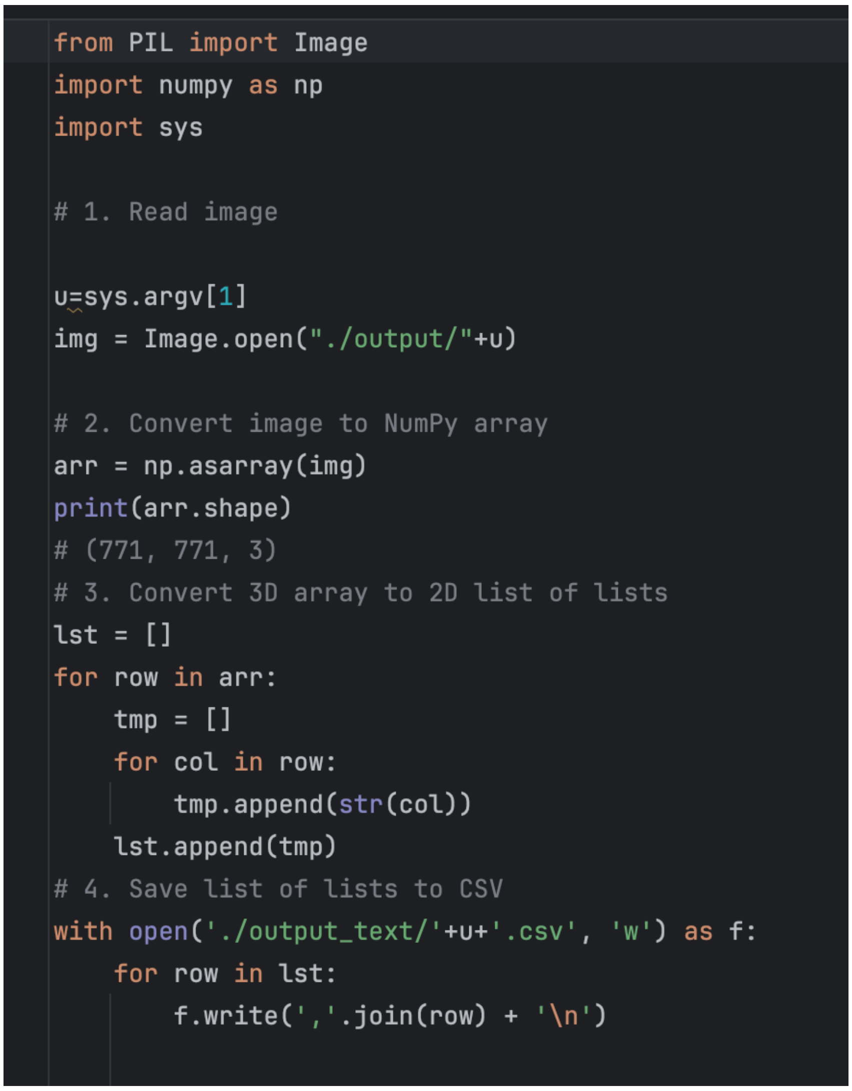
2:



```python
> import ...

path = "./output"
dir_list = os.listdir(path)
count = 0

for file_name in dir_list:
    if file_name == ".DS_Store":
        continue  # Skip processing the ".DS_Store" file it is about macOS problem

    count += 1
    print(file_name, type(file_name))
    command = f"python 3.2conversion_picture_to_numbers.py {file_name}"
    subprocess.run(command, shell=True)

print("Files and directories in '", path, "' :")
print(dir_list)
```

- This code section converts the segmented images into numerical format (RGB values).
- It iterates through the directory containing segmented images.
- Each segmented image file is processed using a Python script 3.2conversion_picture_to_numbers.py for conversion.
- The converted data is saved in CSV format for further analysis and model training.

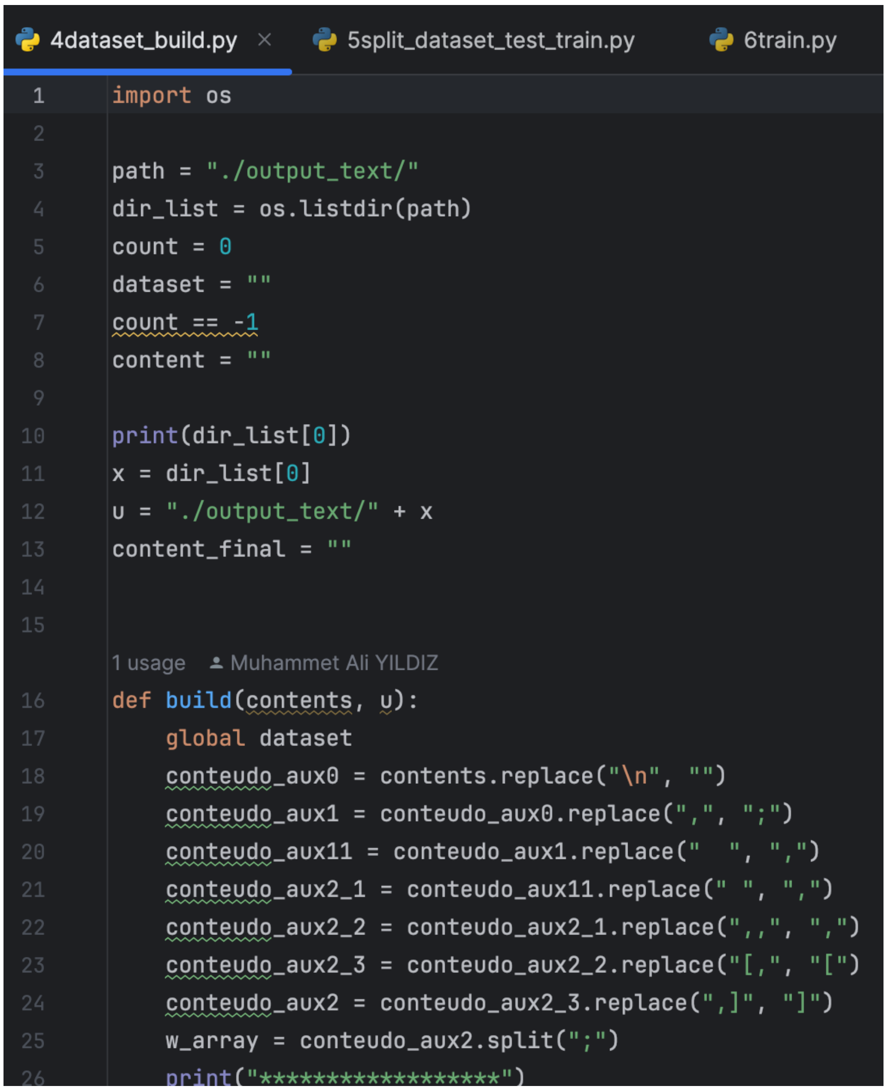
3:



```python
from PIL import Image
import numpy as np
import sys

# 1. Read image

u=sys.argv[1]
img = Image.open("./output/"+u)

# 2. Convert image to NumPy array
arr = np.asarray(img)
print(arr.shape)
# (771, 771, 3)
# 3. Convert 3D array to 2D list of lists
lst = []
for row in arr:
    tmp = []
    for col in row:
        tmp.append(str(col))
    lst.append(tmp)
# 4. Save list of lists to CSV
with open('./output_text/'+u+'.csv', 'w') as f:
    for row in lst:
        f.write(','.join(row) + '\n')
```

- This code section reads the segmented images from the './output/' directory.
- It converts each image to a NumPy array and then to a 2D list of lists, where each inner list represents a row of RGB values.
- The RGB values are converted to strings and saved in CSV format, with each row representing a pixel in the image.

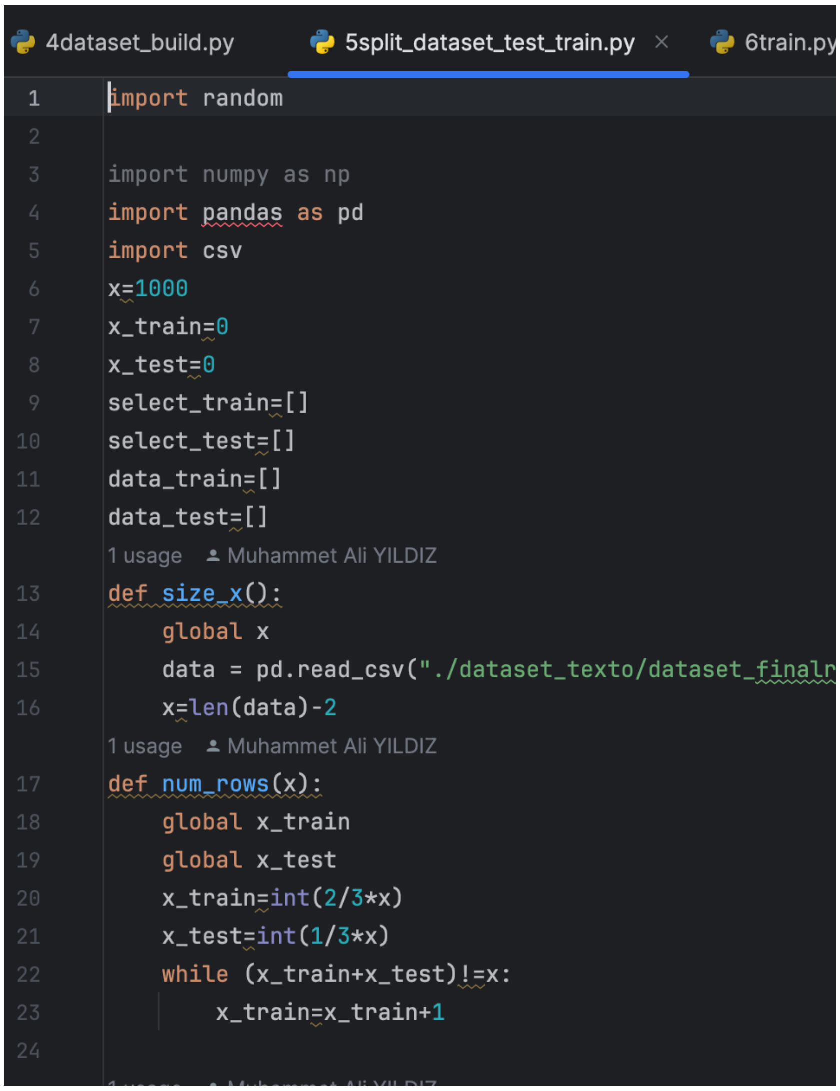
4:



```python
import os

path = "./output_text/"
dir_list = os.listdir(path)
count = 0
dataset = ""
count == -1
content = ""

print(dir_list[0])
x = dir_list[0]
u = "./output_text/" + x
content_final = ""


def build(contents, u):
    global dataset
    conteudo_aux0 = contents.replace("\n", "")
    conteudo_aux1 = conteudo_aux0.replace(",", ";")
    conteudo_aux11 = conteudo_aux1.replace("  ", ",")
    conteudo_aux2_1 = conteudo_aux11.replace(" ", ",")
    conteudo_aux2_2 = conteudo_aux2_1.replace(",,", ",")
    conteudo_aux2_3 = conteudo_aux2_2.replace("[,", "[")
    conteudo_aux2 = conteudo_aux2_3.replace(",]", "]")
    w_array = conteudo_aux2.split(";")
    print("******************")
```

- This code section prepares the training and testing datasets from the converted image data.
- It reads the CSV files containing image data from the './output_text/' directory.
- Rows are selected randomly for inclusion in the training dataset while ensuring a predefined ratio of training to testing data.
- The selected rows are used to generate the training and testing datasets, which are then saved as CSV files in the './datasets/' directory.
-

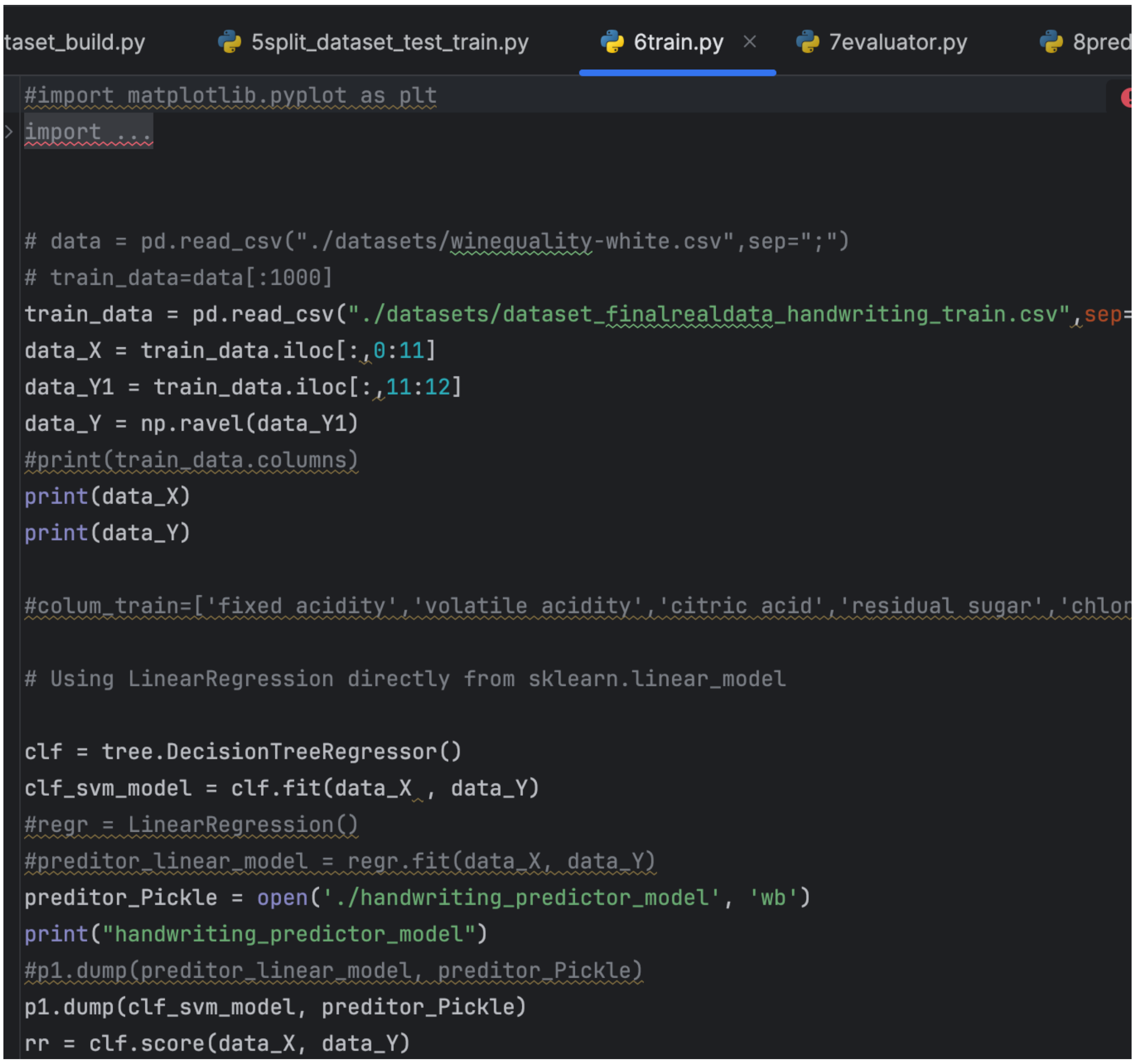
5 :


4dataset_build.py
5split_dataset_test_train.py
6train.py
1 usage Muhammet Ali YILDIZ
1 usage Muhammet Ali YILDIZ

```
import random

import numpy as np
import pandas as pd
import csv
x=1000
x_train=0
x_test=0
select_train=[]
select_test=[]
data_train=[]
data_test=[]
def size_x():
    global x
    data = pd.read_csv("./dataset_texto/dataset_finalr
    x=len(data)-2
def num_rows(x):
    global x_train
    global x_test
    x_train=int(2/3*x)
    x_test=int(1/3*x)
    while (x_train+x_test)!=x:
        x_train=x_train+1

```

- This code section defines functions to prepare the training and testing datasets for model training.
- The size_x() function determines the size of the dataset.
- The num_rows(x) function calculates the number of rows for training and testing based on a predefined ratio.
- The select_rows(x) function randomly selects rows for inclusion in the training dataset while ensuring the correct ratio of training to testing data.
- The generate_train() function generates the training and testing datasets.
- Finally, the datasets are saved as CSV files for further use.

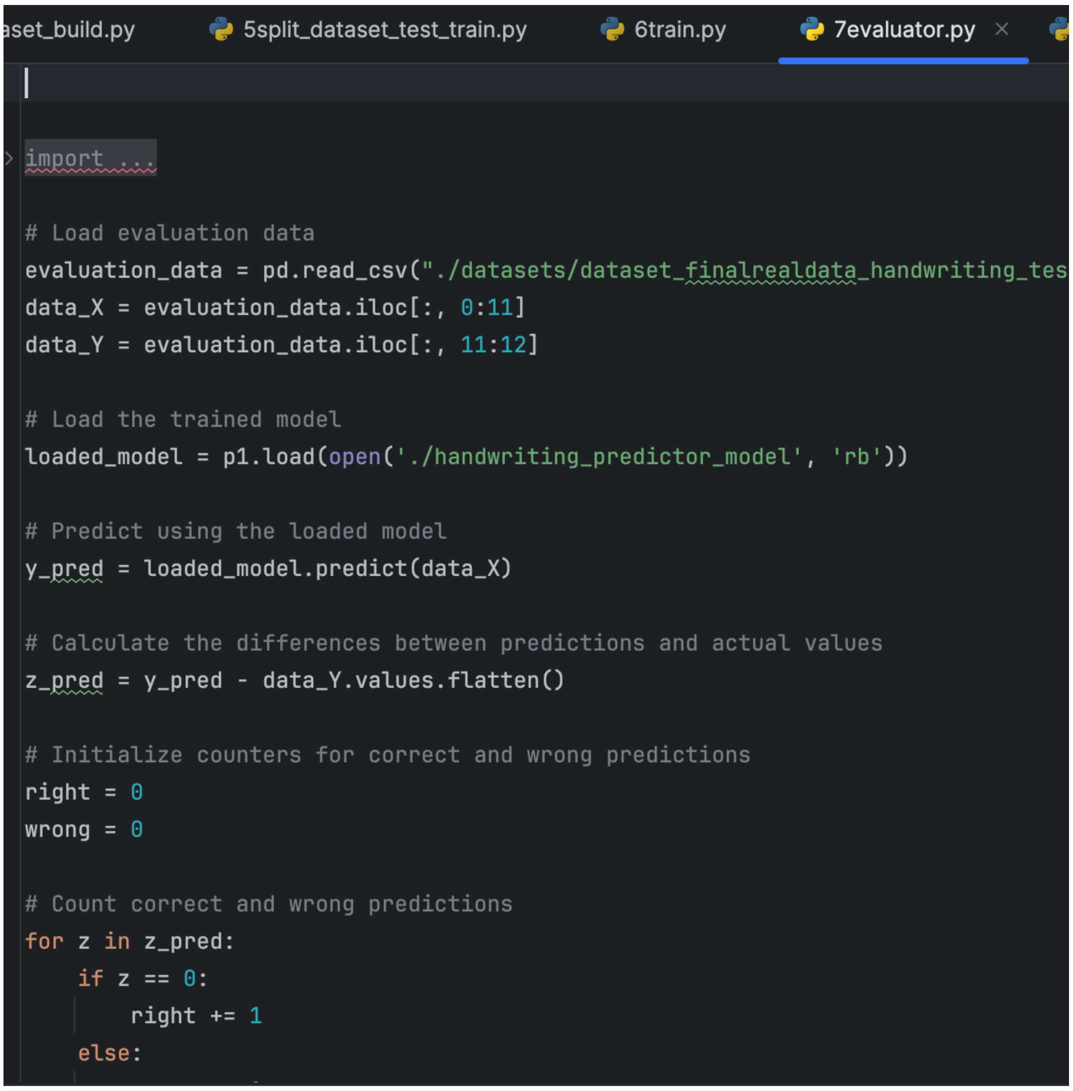
6:



```python
#import matplotlib.pyplot as plt
import ...


# data = pd.read_csv("./datasets/winequality-white.csv",sep=";")
# train_data=data[:1000]
train_data = pd.read_csv("./datasets/dataset_finalrealdata_handwriting_train.csv",sep=
data_X = train_data.iloc[:,0:11]
data_Y1 = train_data.iloc[:,11:12]
data_Y = np.ravel(data_Y1)
#print(train_data.columns)
print(data_X)
print(data_Y)

#colum_train=['fixed acidity','volatile acidity','citric acid','residual sugar','chlor

# Using LinearRegression directly from sklearn.linear_model

clf = tree.DecisionTreeRegressor()
clf_svm_model = clf.fit(data_X , data_Y)
#regr = LinearRegression()
#preditor_linear_model = regr.fit(data_X, data_Y)
preditor_Pickle = open('./handwriting_predictor_model', 'wb')
print("handwriting_predictor_model")
#p1.dump(preditor_linear_model, preditor_Pickle)
p1.dump(clf_svm_model, preditor_Pickle)
rr = clf.score(data_X, data_Y)
```

1. **Importing Libraries**: The code imports necessary libraries such as matplotlib, pickle, numpy, pandas, LinearRegression, mean_squared_error, r2_score, and DecisionTreeRegressor.
2. **Reading Training Data**: It reads the training data from the CSV file named "dataset_finalrealdata_handwriting_train.csv" and stores it in the variable train_data.
3. **Extracting Features and Target**: It extracts the input features (data_X) and the target variable (data_Y) from the training data.
4. **Decision Tree Regression**: The code initializes a DecisionTreeRegressor model (clf) and fits it to the training data using the fit method.
5. **Saving the Model**: It saves the trained model (clf_svm_model) using the pickle.dump() function to a file named "handwriting_predictor_model".
6. **Coefficient of Correlation**: Finally, it calculates the coefficient of correlation (rr) for the trained model using the score() method and prints it out.

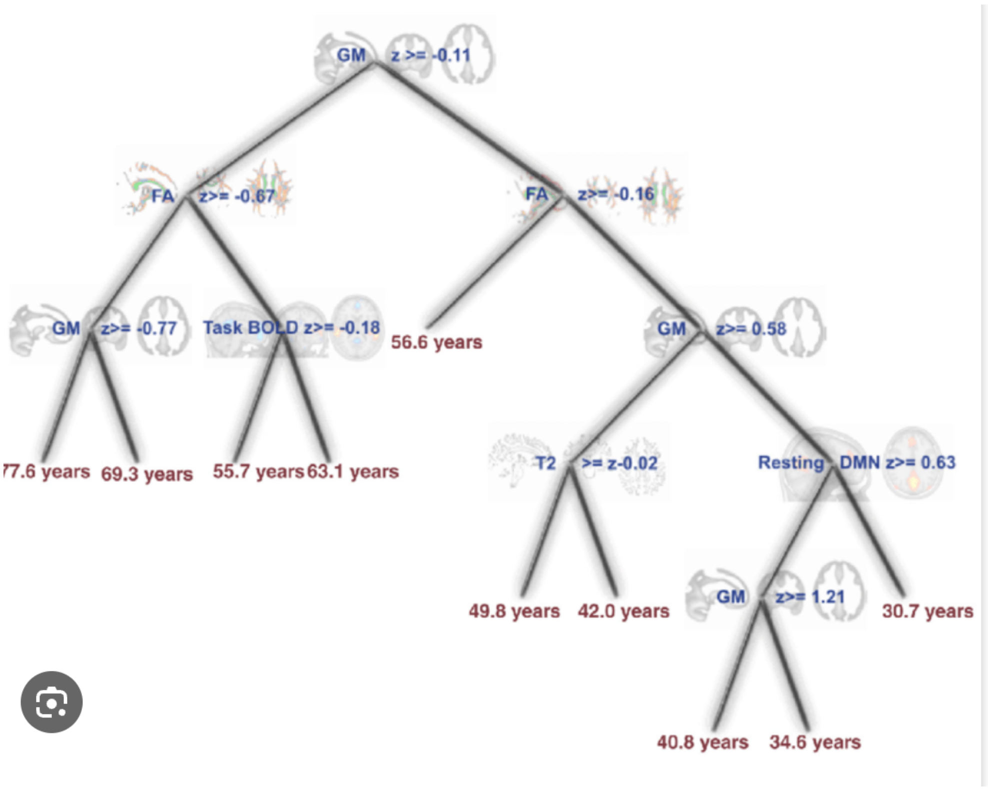
7:



```
import ...

# Load evaluation data
evaluation_data = pd.read_csv("./datasets/dataset_finalrealdata_handwriting_tes
data_X = evaluation_data.iloc[:, 0:11]
data_Y = evaluation_data.iloc[:, 11:12]

# Load the trained model
loaded_model = p1.load(open('./handwriting_predictor_model', 'rb'))

# Predict using the loaded model
y_pred = loaded_model.predict(data_X)

# Calculate the differences between predictions and actual values
z_pred = y_pred - data_Y.values.flatten()

# Initialize counters for correct and wrong predictions
right = 0
wrong = 0

# Count correct and wrong predictions
for z in z_pred:
    if z == 0:
        right += 1
    else:
```

- This code snippet evaluates the trained model using the evaluation dataset.
- It loads the evaluation data from a CSV file and separates the features (X) and the target variable (Y).
- The trained model is loaded from the saved file using pickle.
- Predictions are made using the loaded model on the evaluation dataset.
- Differences between predictions and actual values are calculated.
- The code then counts correct and wrong predictions and calculates the accuracy of the model.
- Finally, it prints the total number of correct and wrong predictions, along with the accuracy metrics.

4. Conclusion



In conclusion, the project successfully implements an AI model for recognizing customized handwritten text. However, there are areas for improvement, including model optimization, feature engineering, and data augmentation. Future work involves integrating the system into real-world applications, enhancing multilingual support, and adapting to different handwriting styles.

• Model Optimization: Experiment with different machine learning algorithms and model architectures to improve predictive performance.
• Feature Engineering: Explore additional features or transformations that may enhance the model's ability to capture relevant patterns in the data.
• Hyperparameter Tuning: Fine-tune model hyperparameters to optimize model performance and generalization.
• Data Augmentation: Consider techniques such as data augmentation to increase the diversity and size of the training dataset, potentially improving model robustness.

Bibliography

[1] Paulo Vieira Professor at IPG (thanks for the everything about the important informations)
[2] Python Software Foundation. Python Language Reference, version 3.x. Available online at: https://docs.python.org/3/
[3] Pillow (PIL Fork) Documentation. Python Imaging Library (Fork). Available online at: https://pillow.readthedocs.io/en/stable/
[4] NumPy Documentation. NumPy User Guide. Available online at: https://numpy.org/doc/stable/
[5] Pandas Documentation. Pandas User Guide. Available online at: https://pandas.pydata.org/docs/
[6] scikit-learn Documentation. scikit-learn: Machine Learning in Python. Available online at: https://scikit-learn.org/stable/documentation.html
[7] DecisionTreeRegressor Documentation. scikit-learn DecisionTreeRegressor API Reference. Available online at: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
[8] Anaconda Documentation. Anaconda Distribution User Guide. Available online at: https://docs.anaconda.com/
[9] Jupyter Notebook Documentation. Project Jupyter. Available online at: https://jupyter-notebook.readthedocs.io/en/stable/
[10] GitHub. Repository for split_image. Available online at: https://github.com/split_image
[11] GitHub. Repository for conversion_picture_to_numbers. Available online at: https://github.com/mtech00/conversion_picture_to_numbers
[12] Microsoft. Visual Studio Code Documentation. Visual Studio Code User Guide. Available online at: https://code.visualstudio.com/docs