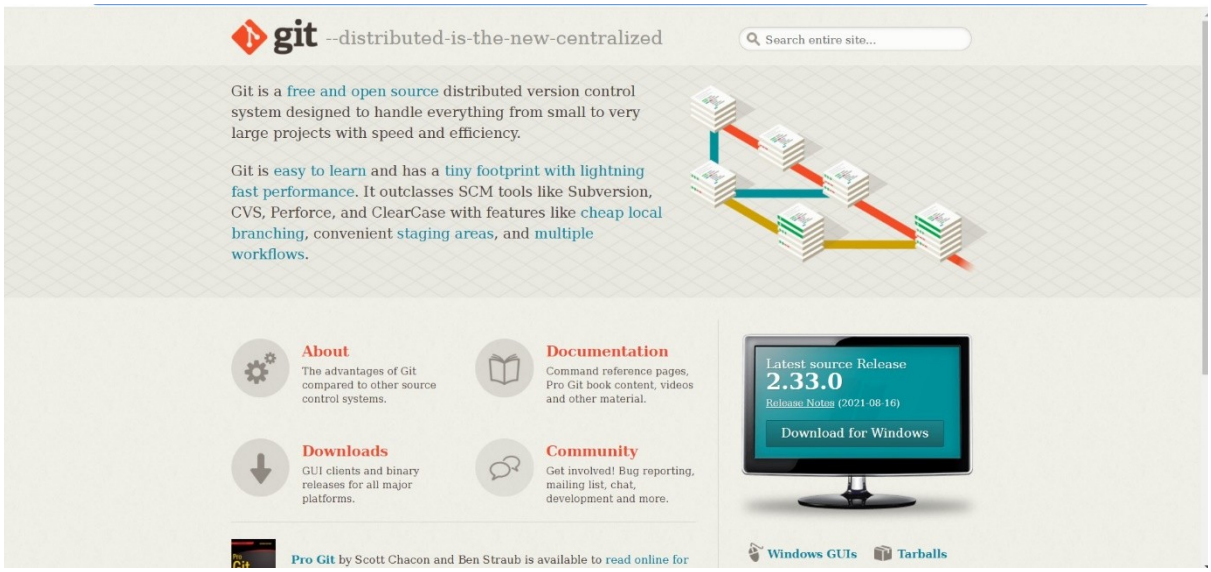
Installing Git in your system and setting a GitHub account

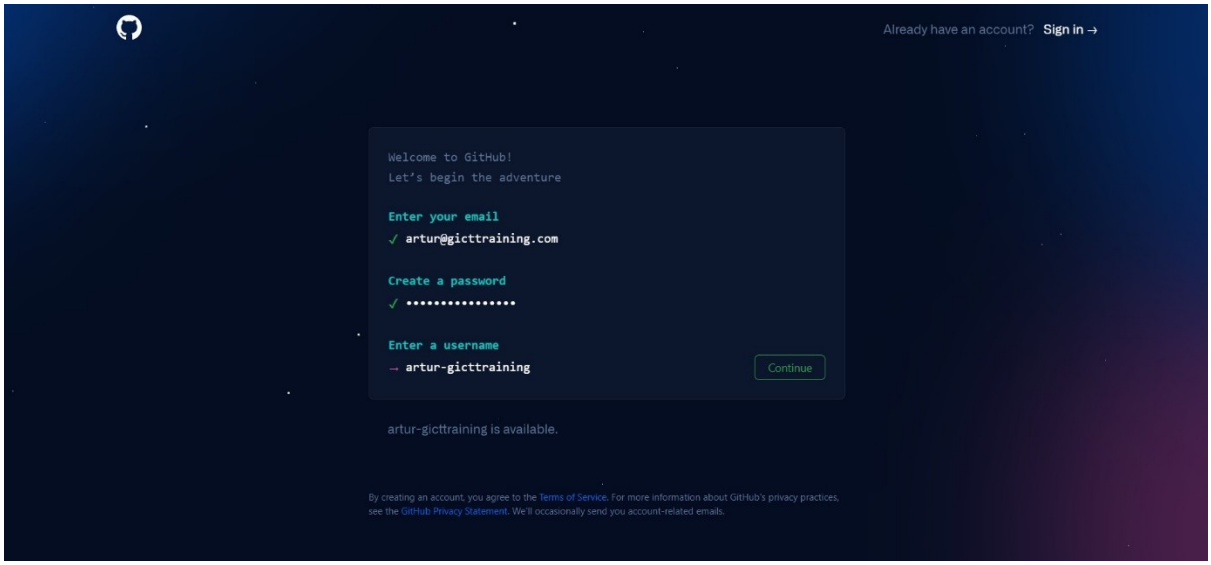
Step 1: Go to https://git-scm.com/ and click on Download for Windows

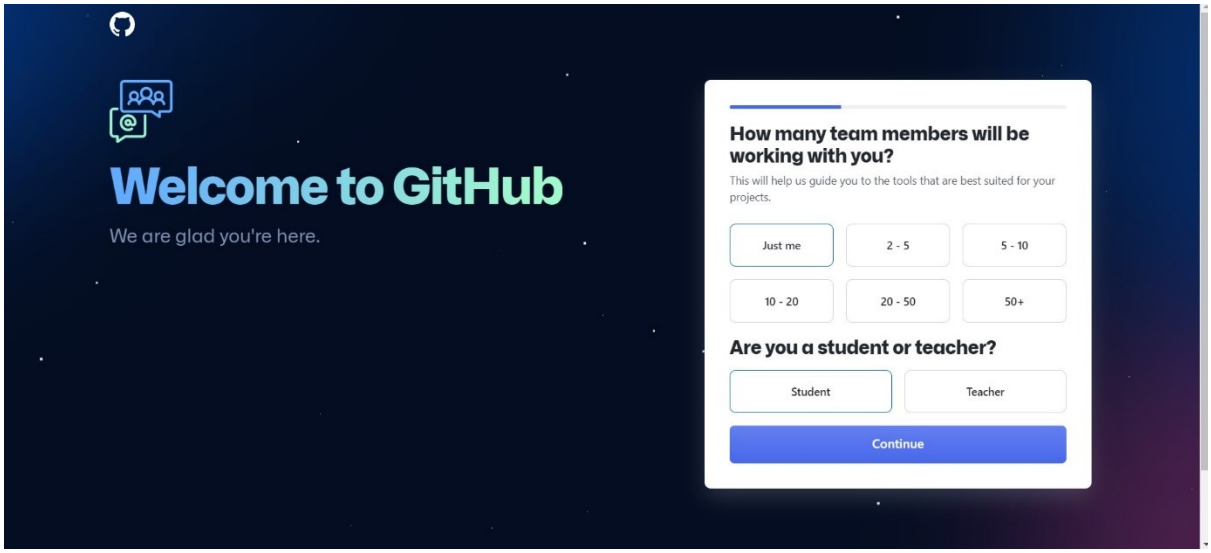


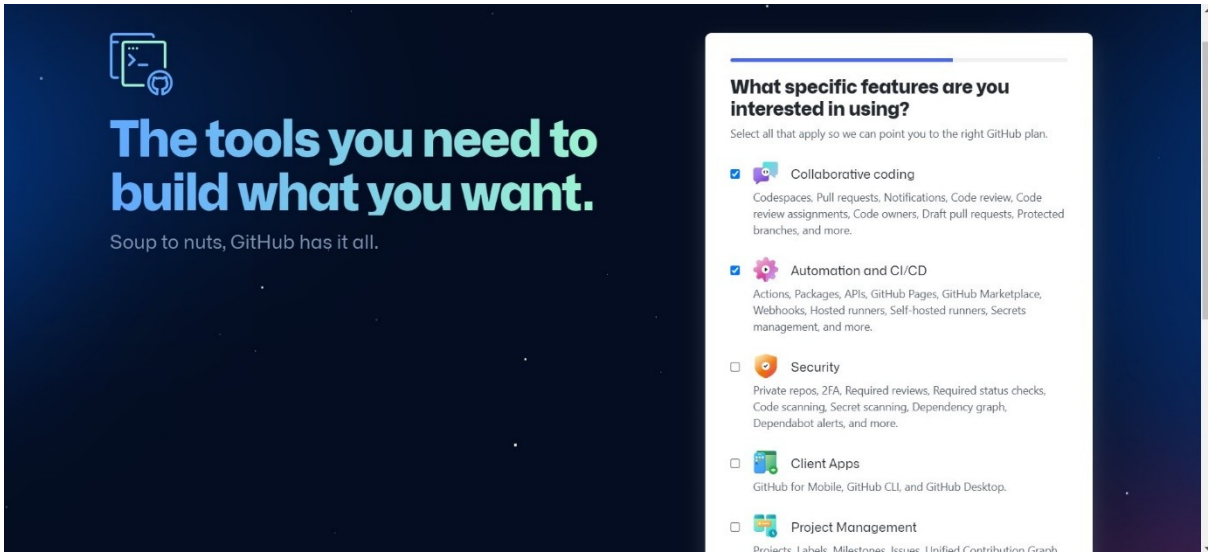
Step 2: Run the installer that you just downloaded and follow the instructions.

Step 3: Go to https://github.com/ and click on Sign Up.

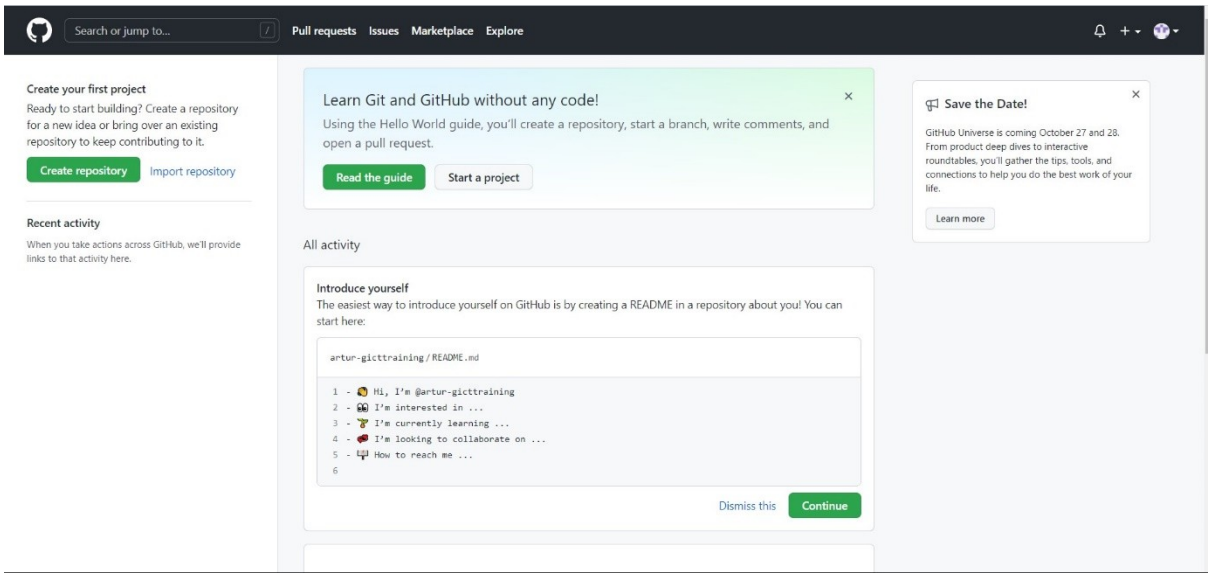
Step 4: Follow the instructions to set your GitHub account





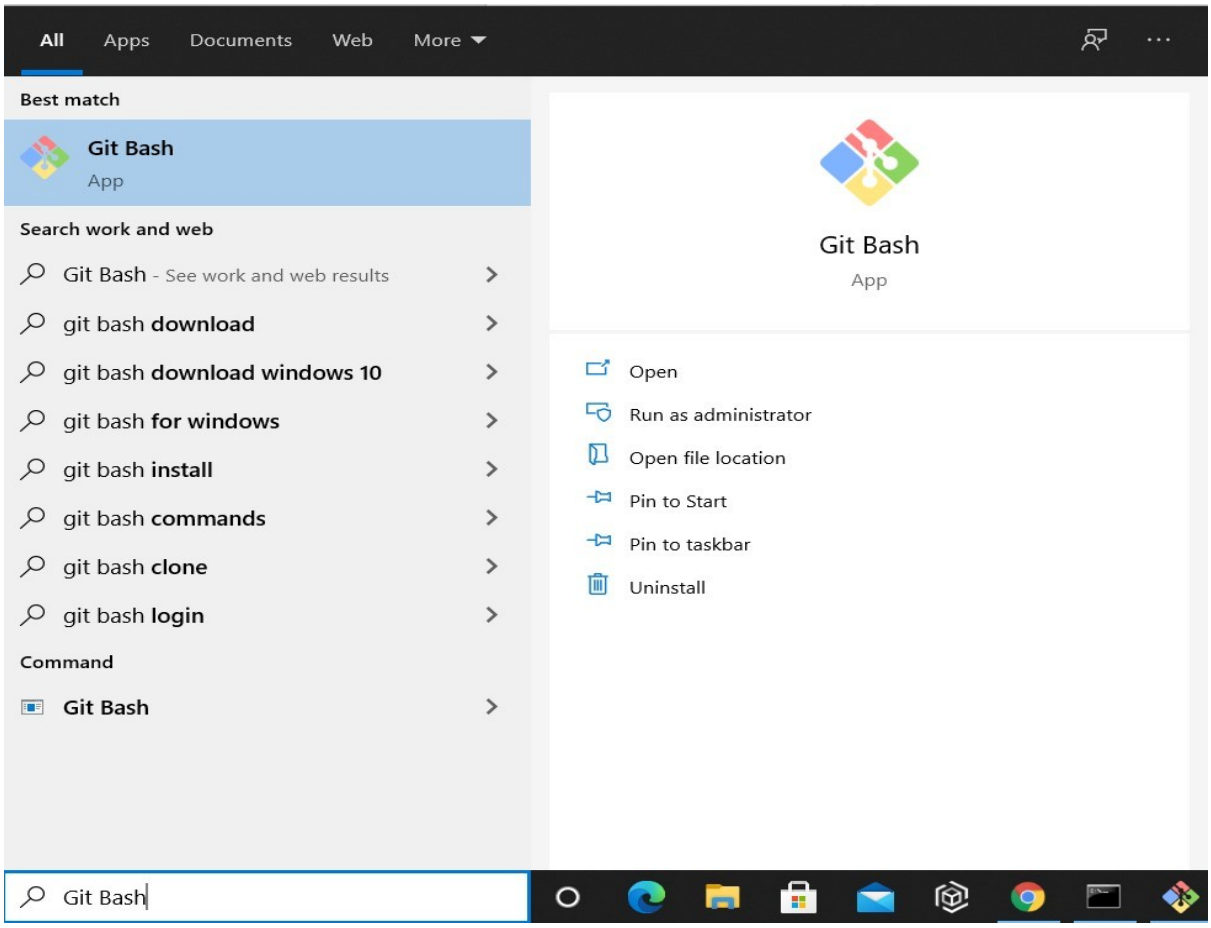


Step 5: After your account is created you should see something similar to the image below



Step 6: Adding SSH keys to your GitHub account. This will allow you to, for example, push/pull commits between your local code repositories with the ones you have in your GitHub account.

Step 7: Open Git Bash



Step 8: Type on Git Bash

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Step 9: When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

Step 10: At the prompt, type a secure passphrase. Remember this passphrase as you will need it sson.

Step 11: Type eval "$(ssh-agent -s)". Your output will look like

```
Artur@DESKTOP-1L4BSGI MINGW64 ~
$ eval "$(ssh-agent -s)"
Agent pid 286
```

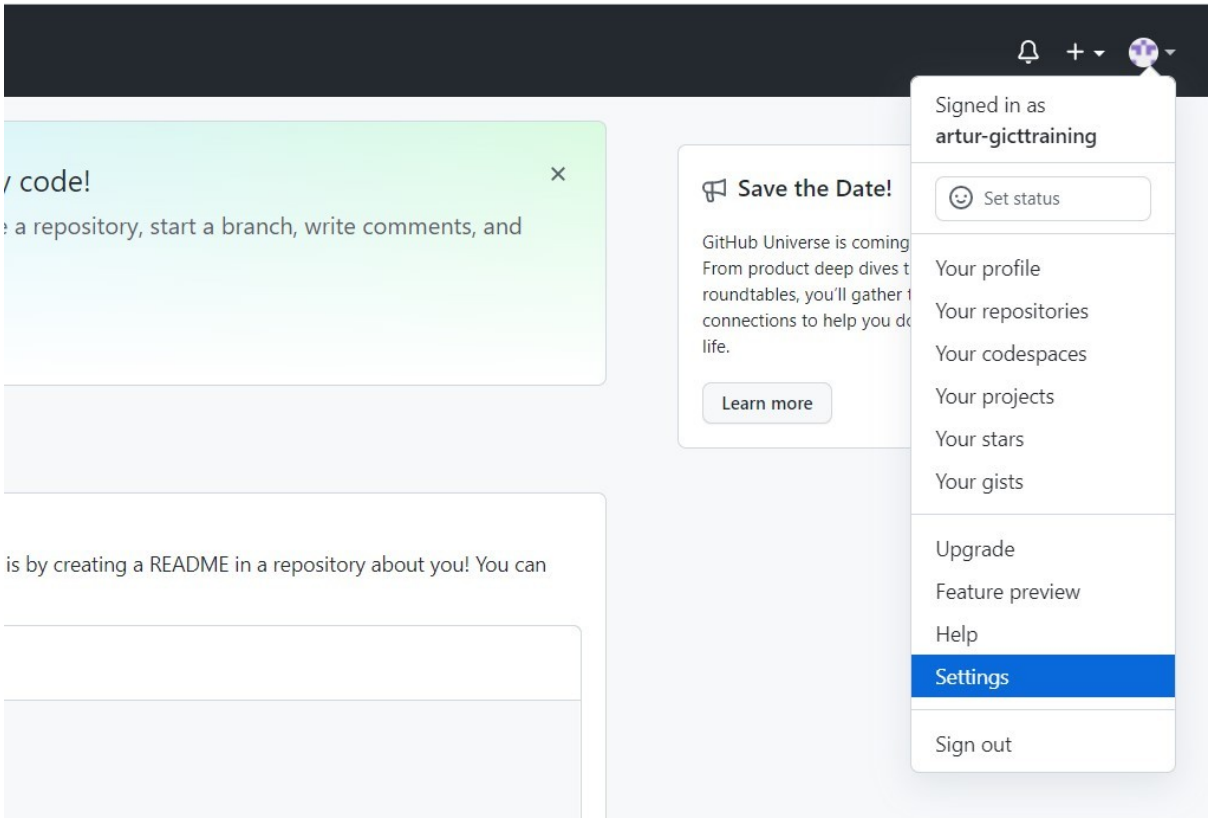
Step 12: To add your SSH key to the SSH agent type

```
ssh-add ~/.ssh/id_ed25519
```

Step 13: Copy the SSH public key to your clipboard with the command

```
clip < ~/.ssh/id_ed25519.pub
```

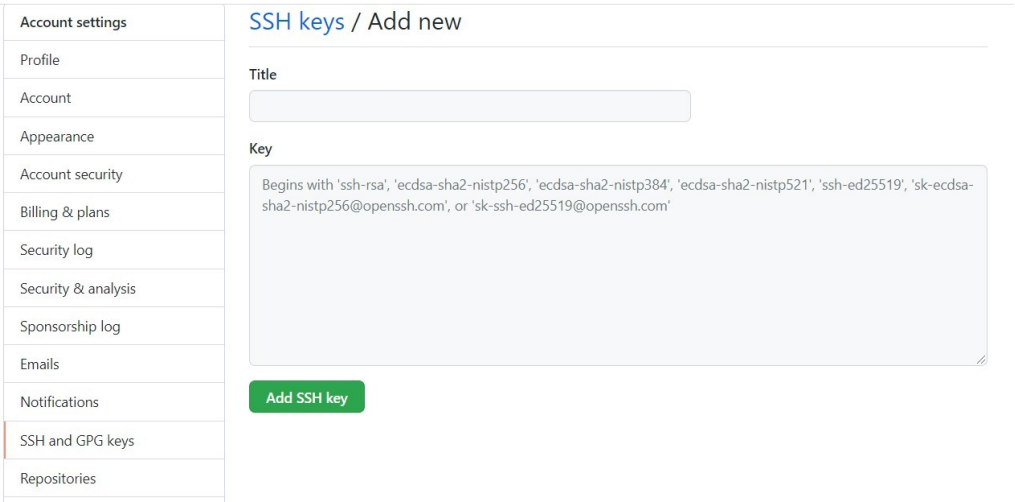
Step 14: Click on the top right corner button and then click “Settings”



Step 15: Then click on SSH and GPG keys. Click New SSH key or Add SSH key.

Step 16: In the title field put something like “Personal Windows PC”

Step 17: Paste your key into the “Key” field



Step 18: If prompted, type your GitHub password.

Step 19. Open a command prompt and type

git config --global user.name "Your Name"

git config --global.email “Your Email”

Setting up a Continuous Integration

Step 1: Create a folder named ml_ci.

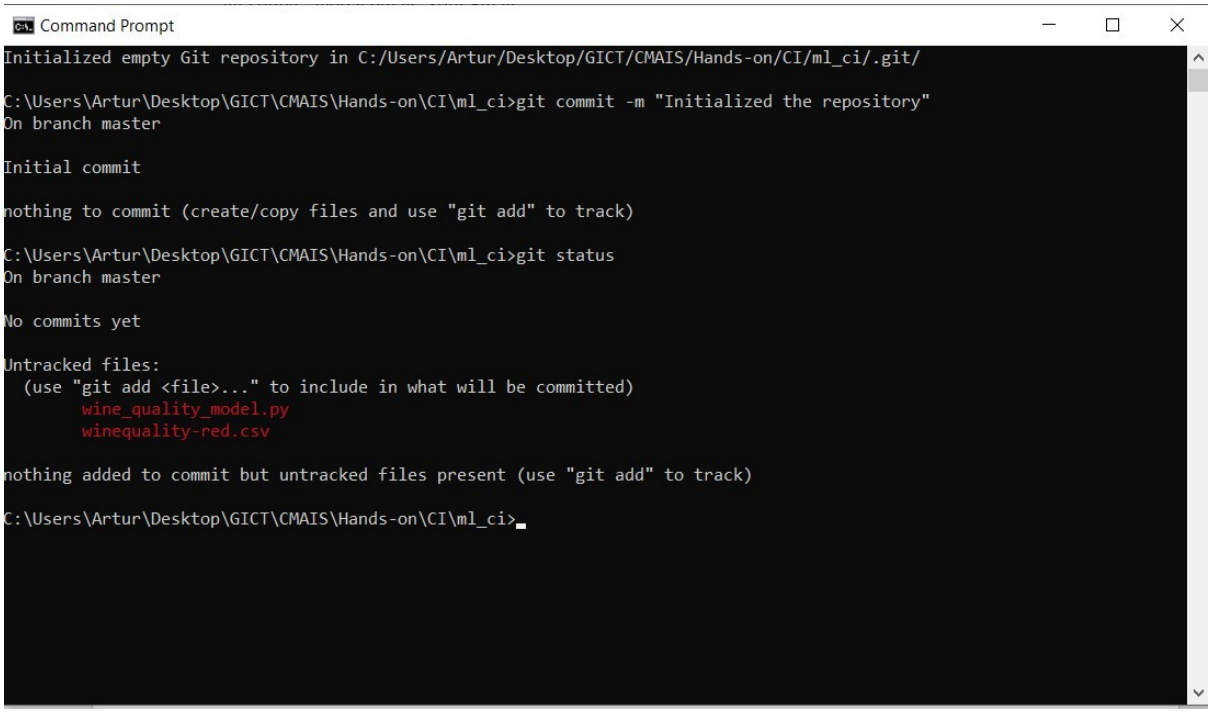
Step 2: Inside this directory run

git init .

This will setup a local git repository.

Step 3: Add the files wine_quality_model.py and winequality-red.csv files inside this directory.

Step 4: Type git status to check which files have been modified or untracked

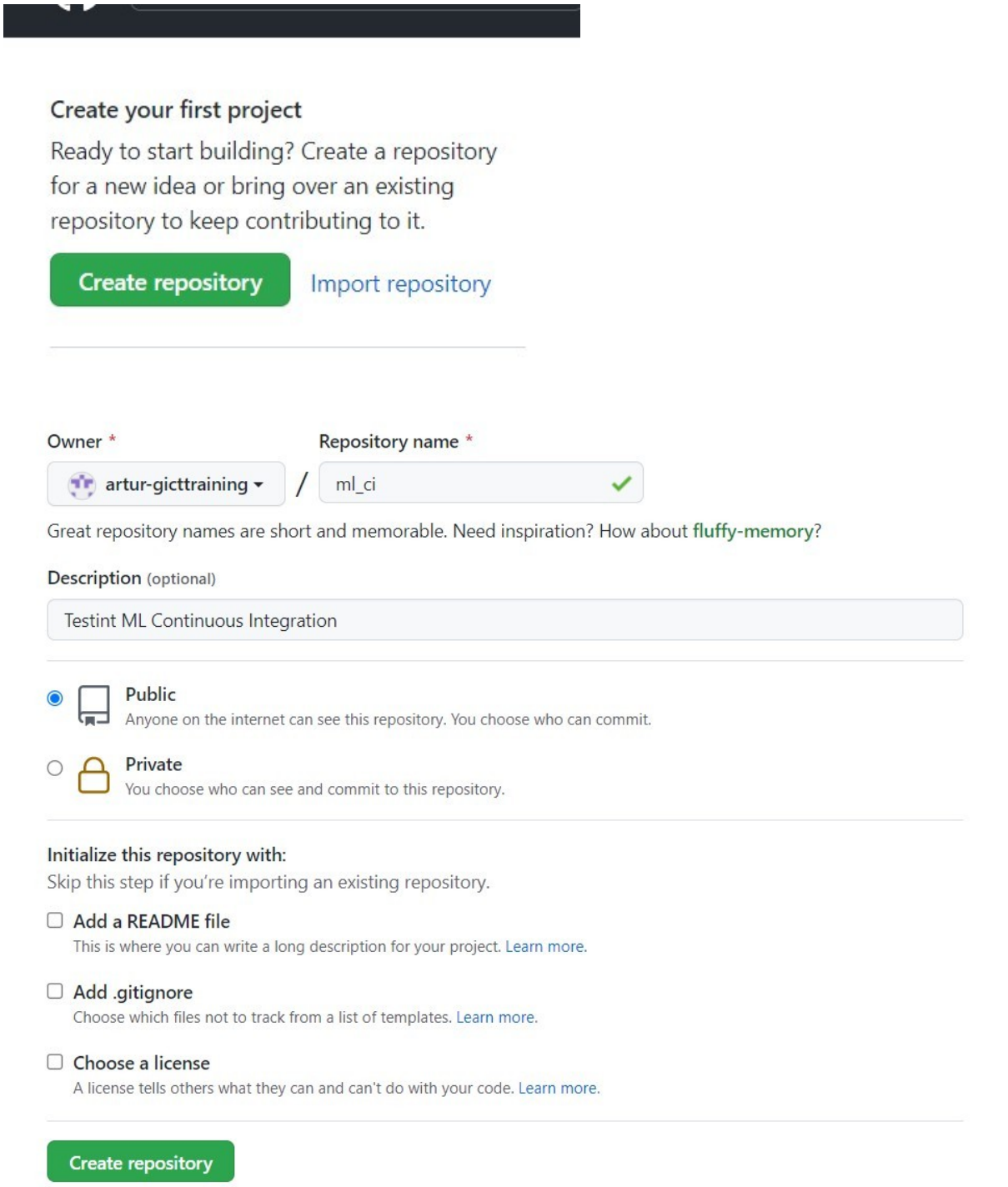


Step 5: Type git add . to track the added files.

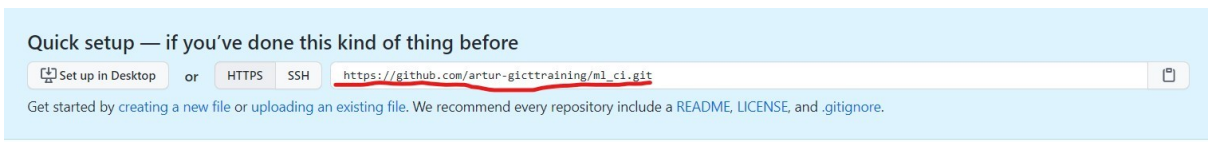
Step 6: Type git commit -m “First commit”

Step 7: Type git branch -M main

Step 8: In GitHub create a new repository



Step 9: Copy the link below



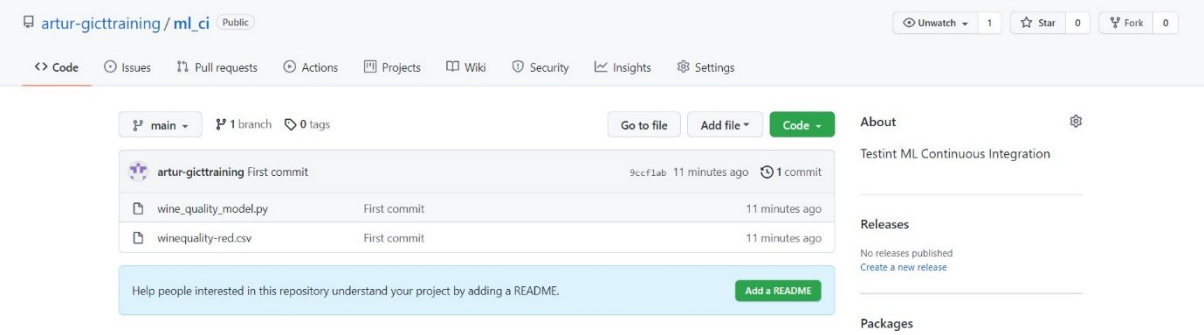
Step 10: Run the command

```
git remote add origin https://github.com/artur-gicttraining/ml_ci.git
```

Step 11: Run the command

```
git push -u origin main
```

Step 12: Check that your files are available on the GitHub repository



Step 13: In your local repository add a requirements.txt file with the following contents

```
pandas
sklearn
matplotlib
seaborn
numpy
```

Step 14. Run in the command prompt

git add .

git commit -m "Added requirements.txt"

Step 15: In you local repository create a file named README.md with the content

```
# My first ML CI
Modelling of Wine Quality
```

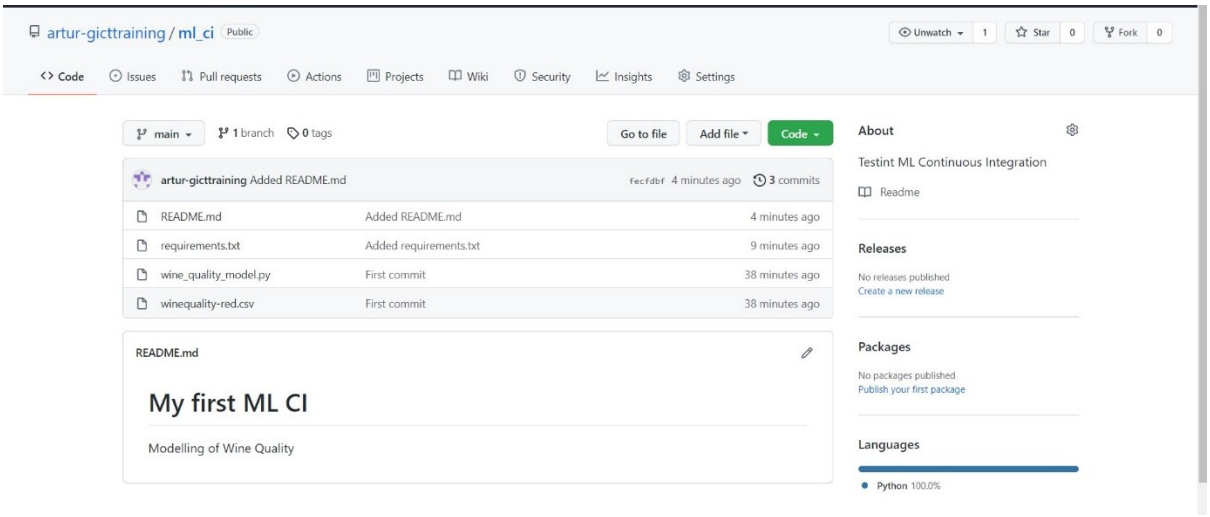
Step 16: Run the following commands

git add .

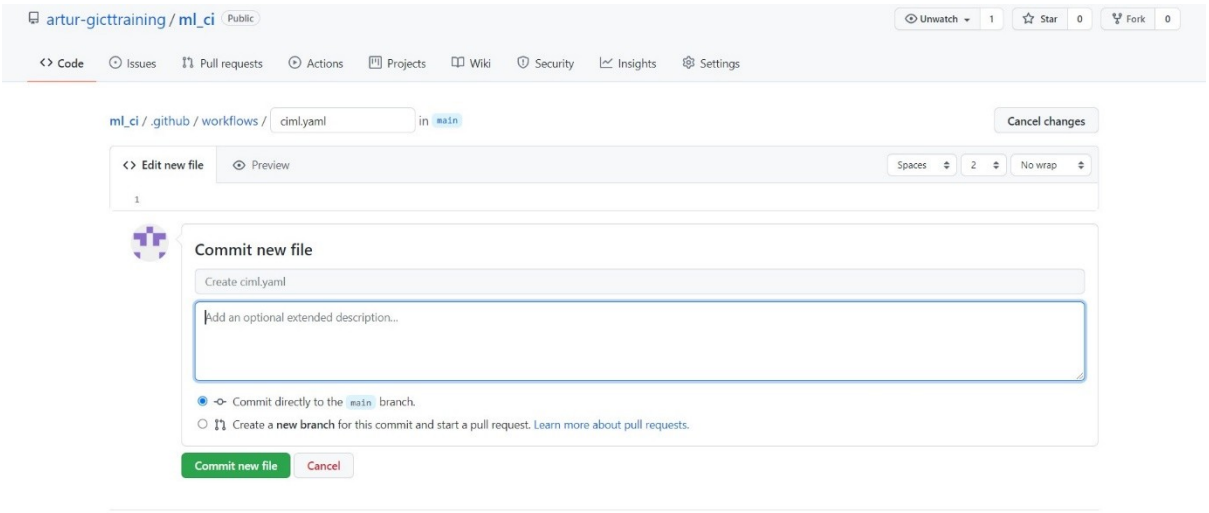
git commit -m "Added README.md"

git push

Step 17: Your repository on GitHub should look like the figure below



Step 18: We will now setup a Continuous Integration system such that everytime something happens in the repository it will execute some action. To use GitHub Actions crate the file ml_ci/.github/workflows/ciml.yaml by clicking in the “Add file” button.



Commit with message “Create ciml.yaml”.

Step 19: In your local repository run git pull and add the following code to the ciml.yaml file

```
name: your-workflow-name
on: [push]
jobs:
  run:
    runs-on: ubuntu-latest
    container: docker://dvcorg/cml-py3:latest
    steps:
      - uses: actions/checkout@v2
      - name: Train model
        env:
          REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
```

```
# Your ML workflow goes here
pip install -r requirements.txt
python wine_quality_model.py
```

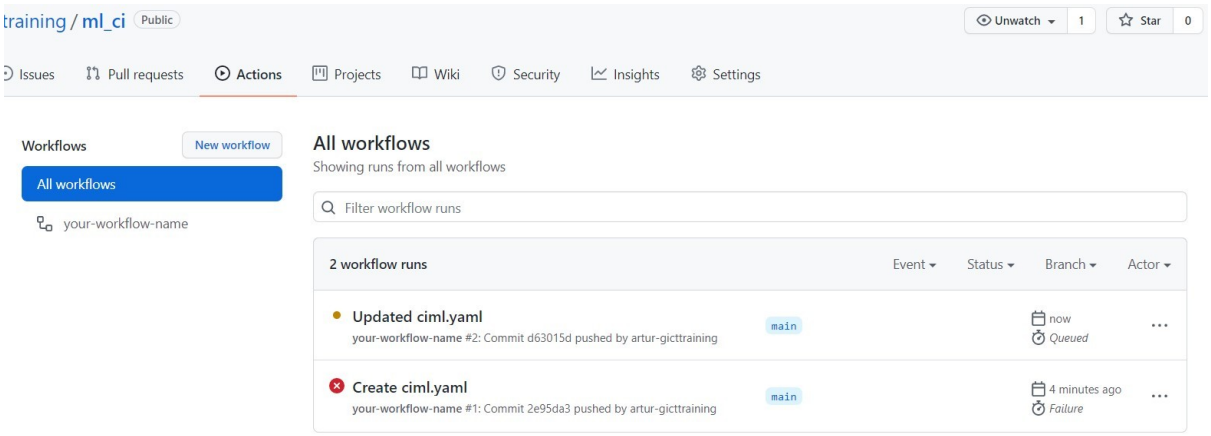
Step 20: Run the following commands

git add .

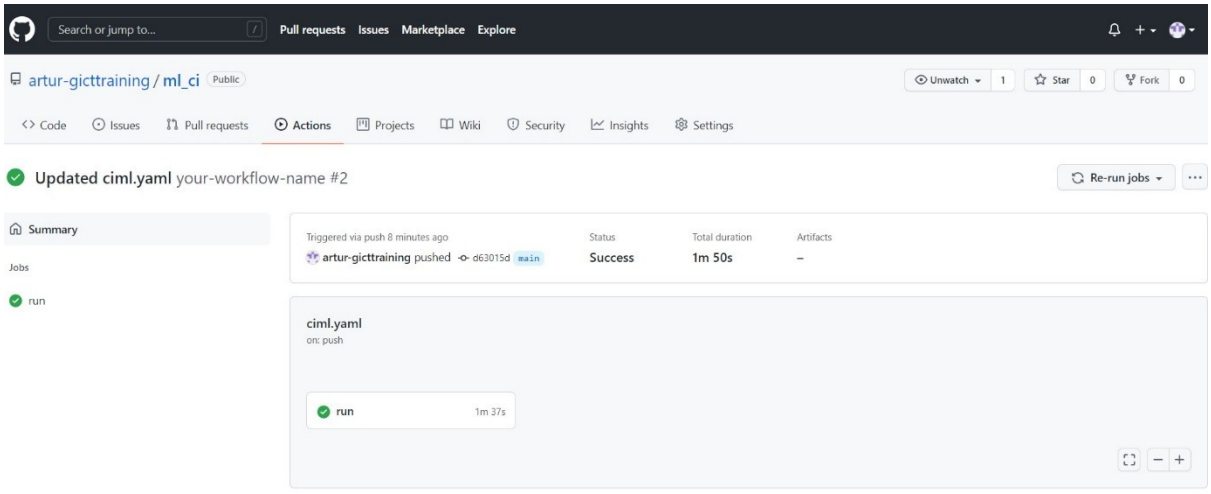
git commit -m “Updated ciml.yaml”

git push

Step 21: If you now visit your repository on GitHub and click on actions you will see something like the below figure



This means that your workflow as defined in the .yaml fine is running on GitHub. The yellow circle next to the commit “Updated ciml.yaml” means that the workflow is still running. If everything is executed successfully this ball will turn green, otherwise it will turn red.



Step 22: In the above workflow our python script was only training the model with all the data and no output was generated by the code. Now we are going to split the dataset into a train and test set, train the decision tree on the training set and compute the mean accuracy in the training and test

sets. We would like this two metrics to be saved in a file named results.txt. Modify the python metric to

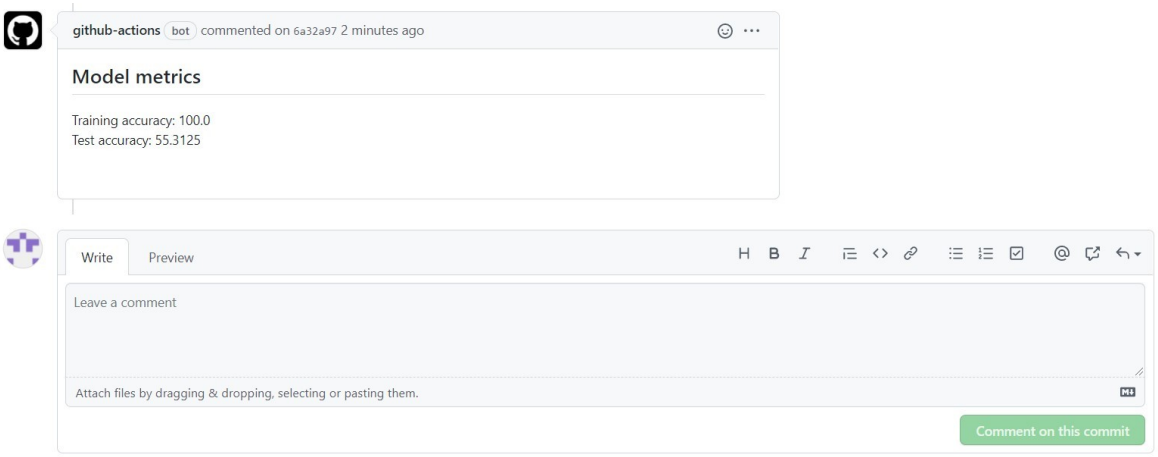
```
# Import wine dataset
import pandas as pd
wines = pd.read_csv('winequality-red.csv')
wines.columns = wines.columns.str.replace(" ", "_")

# Split dataset into features and target and intro training and test sets
from sklearn.model_selection import train_test_split
X = wines.loc[:, ['fixed_acidity', 'volatile_acidity', 'citric_acid', 'alcohol
']]
y = wines.loc[:, 'quality']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=42)

# Scale the variables to be within the range of -1 to 1.
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1,1))
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Train a Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Compute the training and test accuracy
training_acc = model.score(X_train, y_train) * 100
test_acc = model.score(X_test, y_test) * 100

# Output the results in a .txt file
with open("results.txt", "w") as f:
    f.write(f"Training accuracy: {training_acc}\n")
    f.write(f"Test accuracy: {test_acc}\n")

# Export the model using pickle
import pickle
file_name = "model.pkl"
open_file = open(file_name, "wb")
pickle.dump([scaler, model], open_file)
open_file.close()
```

Step 23: To output a report to the workflow add the following to the .yaml file

```
echo "## Model metrics" > report.md
cat results.txt >> report.md
```

Step 24: Commit the changes and run git push.

Step 25: When the workflow ends a report is generated. You can look by clicking in “Code” button in the GitHub repository, followed by a click in the most recent commit.



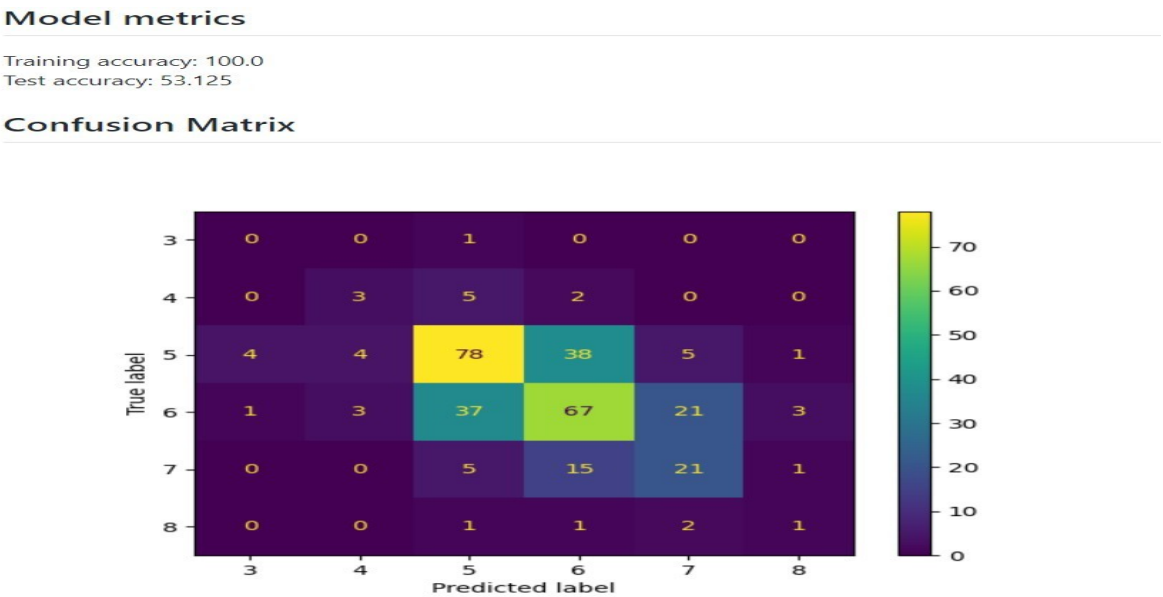
Step 26: Our report can also contain pictures. Let’s add a plot of the confusion matrix by adding to the python script

```
# Plot a confusion matrix
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
plot_confusion_matrix(model, X_test, y_test)
plt.savefig("confusion_matrix.png")
```

and add

```
echo "## Confusion Matrix" >> report.md
cml-publish feature_importance.png --md >> report.md
cml-publish residuals.png --md >> report.md
```

before cml-send-comment report.md in the ciml.yaml. Commit and push the changes and in GitHub you should find the result below



Step 27: Our model is clearly suffering from overfitting. We have chosen a Decision Tree Classifier and does not have a depth limit, causing overfitting. Let's modify our model to a maximum depth of 2. Start by creating a new branch named reduce-overfitting with the command

git checkout -b reduce-overfitting.

Modify our model to

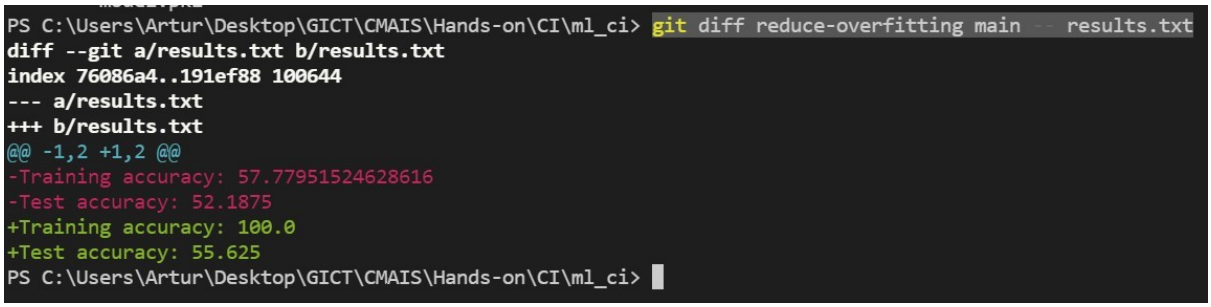
```
model = DecisionTreeClassifier(max_depth=2)
```

Step 28: Run the command python wine_quality_model.py, commit the changes and change back to the main branch.

Step 29: Run the command

git diff reduce-overfitting main -- results.txt

The output will look like



Step 30: We were successful in reducing overfitting, although at the expense of some accuracy. Let's merge and push the branches by running

git checkout main

git merge reduce-overfitting

git push

Step 31: There is a better way of comparing the metrics between two models and that is through a Data Version Control (DVC) pipeline. To install this library visit https://dvc.org/ to download and install dvc in your machine.

Step 32: Run dvc init in your local repository.

Step 33: Start to define a pipeline by adding a file named dvc.yaml with the following content

```
stages:
  train:
    cmd: python wine_quality_model.py
    deps:
    - wine_quality_model.py
    outs:
    - "confusion_matrix.png"
    metrics:
```

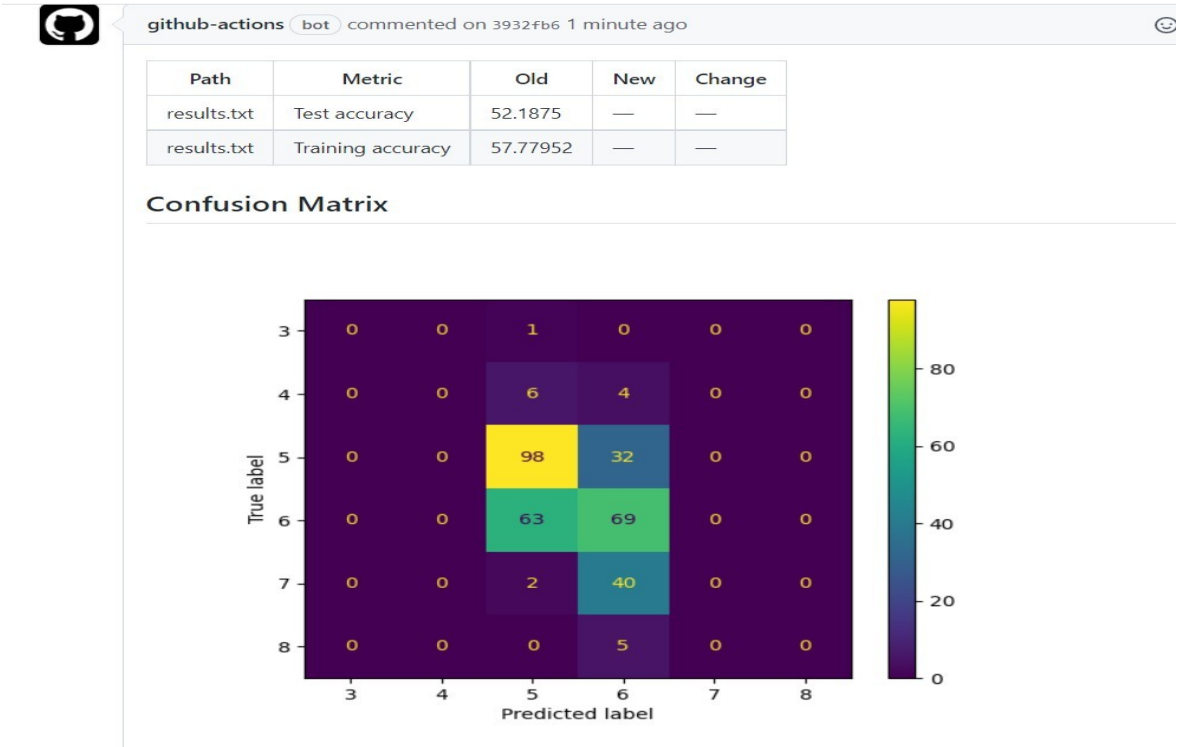
```
- results.txt:
    cache: false
```

Step 34: Run dvc repro. This command will execute python wine_quality_model.py as before and generate the confusion_matrix.png and results.txt files.

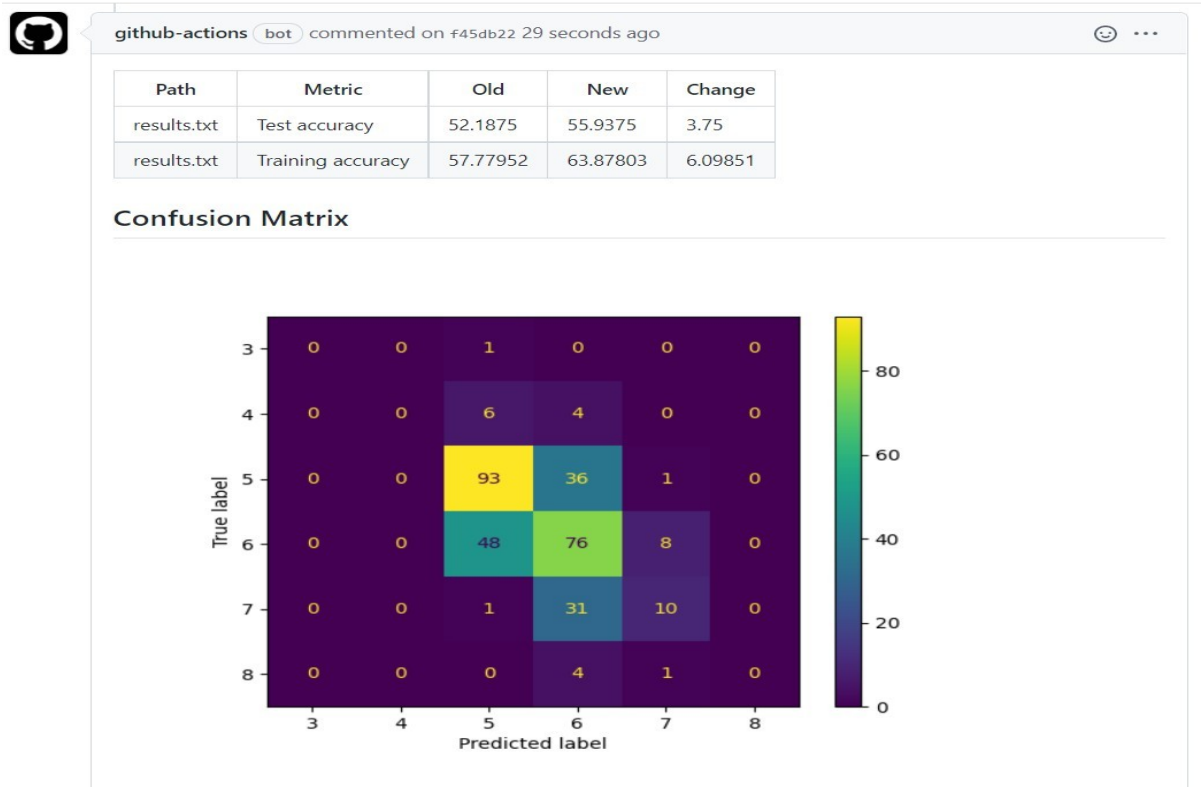
Step 35: Modify the end of your ciml.yaml file to

```
# Your ML workflow goes here
pip install -r requirements.txt
dvc repro

git fetch --prune
dvc metrics diff --show-md main > report.md

echo "## Confusion Matrix" >> report.md
cml-publish confusion_matrix.png --md >> report.md

cml-send-comment report.md
```

Add the commits and push. After the workflow ends you will find in GitHub



Step 36: Create a new branch with git checkout -b RandomForest. We will now build a Random Forest classifier with the extra feature residual sugar. Make the necessary modifications commit and push the changes. After the workflow ends in Git Hub you find the following output



Step 37: Go back to the main branch and merge with the RandomForest branch.

Step 38: We will now build a dvc pipeline that will import a whine quality dataset, merge it with the red wine dataset and finally train our final model with the whole dataset. Add the python script get_white_wine.py:

```
# Get the White Wine dataset
from urllib.request import urlretrieve
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-white.csv'
urlretrieve(url, "winequality-white-raw.csv")

# Load the white wine dataset as a data frame
import pandas as pd
white_wine = pd.read_csv("winequality-white-raw.csv", sep=";")

# Export as a csv file
white_wine.to_csv("winequality-white.csv", sep=",", index=False)
```

Step 39: Add the python script merge_data.py

```
import pandas as pd

# Load the white and red datasets
white = pd.read_csv("winequality-white.csv")
```

```
red = pd.read_csv("winequality-red.csv")

# Merge the data frames in a single file
result = pd.concat([white, red], ignore_index=True)

# Export as a csv file
result.to_csv("winequality.csv", sep=",", index=False)
```

Step 40: Modify the wine_quality_model.py file to load the dataset “winequality.csv”

Step 41: Modify the dvc.yaml file to

```
stages:
  get_white_wine_dataset:
    cmd: python get_white_wine.py
    deps:
      - get_white_wine.py
    outs:
      - winequality-white.csv
  merge_datasets:
    cmd: python merge_data.py
    deps:
      - merge_data.py
      - winequality-white.csv
    outs:
      - winequality.csv
  train:
    cmd: python wine_quality_model.py
    deps:
      - wine_quality_model.py
      - winequality.csv
    outs:
      - confusion_matrix.png
    metrics:
      - results.txt:
          cache: false
```

Step 42: Commit and push the commits. In Git Hub the following will be shown

github-actions bot commented on `f45db22` 29 seconds ago

Path	Metric	Old	New	Change
results.txt	Test accuracy	52.1875	55.9375	3.75
results.txt	Training accuracy	57.77952	63.87803	6.09851

Confusion Matrix