ACIVITY-10

DEEP LEARNING MODEL IMPLEMENTATION AND PERFORMANCE ANALYSIS

Problem Statement: In this project, we aim to improve the performance of existing regression and classification models by rebuilding them using deep learning techniques. The goal is to compare the performance of traditional machine learning (ML) approaches with deep learning (DL) methods, assessing metrics such as accuracy, precision, recall, and MSE, MAE, r2_score.

Regression model using deep learning algorithm

Python Code:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow import keras

# 1. Load and prepare data
iris_data = load_iris()
data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
data['target'] = iris_data.data[:, 2]  # Using petal width as the target


# 2. Display the first few rows of the dataset
print(data.head())

# 3. Visualize correlation with a heatmap
plt.figure(figsize=(10, 8))
```

```python
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm', center=0)
plt.title('Feature Correlation Heatmap')
plt.show()

# 4. Prepare features and target variable
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# 6. Build the deep learning model
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Output layer for regression
])

# 7. Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# 8. Train the model
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_val, y_val))

# 9. Evaluate the model
```

```
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Custom accuracy-like metric: Percentage of predictions within a threshold
threshold = 0.1  # Set your acceptable range
accuracy_like = np.mean(np.abs(y_pred.flatten() - y_test) <= threshold)
# Display results
print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared Score: {r2:.2f}')

# Optional: Display a few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred.flatten()})
print(results.head())
```

OUTPUT:

	Sepal length(cm)	Sepal width(cm)	Petal length(cm)	Petal width(cm)	target
0	5.1	3.5	1.4	0.2	1.4
1	4.9	3.0	1.4	0.2	1.4
2	4.7	3.2	1.3	0.2	1.3
3	4.6	3.1	1.5	0.2	1.5
4	5.0	3.6	1.4	0.2	1.4

```
[5 rows x 5 columns]
Epoch 1/100
4/4 ━━━━━━━━━━━━ 1s 65ms/step - loss: 18.0929 - val_loss: 19.1862
Epoch 2/100
4/4 ━━━━━━━━━━━━ 0s 12ms/step - loss: 16.7744 - val_loss: 18.3727
Epoch 3/100
4/4 ━━━━━━━━━━━━ 0s 12ms/step - loss: 16.3398 - val_loss: 17.6929
Epoch 4/100
```

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 15.1802 - val_loss: 17.0941

Epoch 5/100

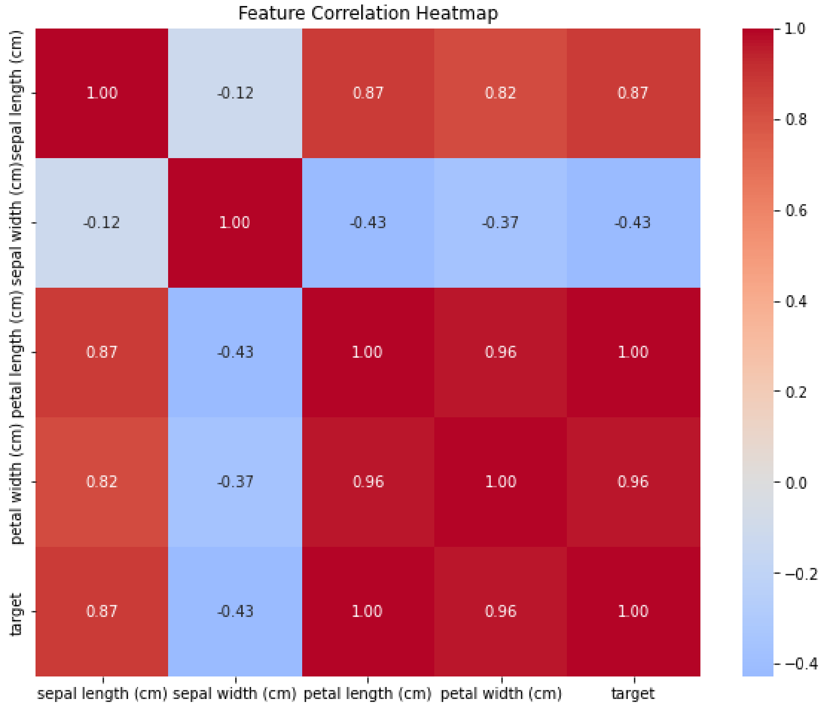
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 15.0142 - val_loss: 16.5433

Mean Squared Error: 0.04

R-squared Score: 0.99

	Actual	Predicted
143	5.9	5.920325
56	4.7	4.306509
128	5.6	5.640619
69	2.9	3.671755
68	4.4	4.786403

Graph:



Regression model using Machine Learning algorithm

Python Code:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# 1. Load and prepare data
iris_data = load_iris()
data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
data['target'] = iris_data.data[:, 2]  # Using petal width as the target

# 2. Display the first few rows of the dataset
print(data.head())

# 3. Visualize correlation with a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm', center=0)
plt.title('Feature Correlation Heatmap')
plt.show()

# 4. Prepare features and target variable
X = data.drop('target', axis=1)
y = data['target']
# Split the data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 6. Build the regression model
model = LinearRegression()

# 7. Train the model
model.fit(X_train, y_train)

# 8. Make predictions
y_pred = model.predict(X_test)

# 9. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Custom accuracy-like metric: Percentage of predictions within a threshold
threshold = 0.1  # Set your acceptable range
accuracy_like = np.mean(np.abs(y_pred - y_test) <= threshold) * 100

print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared Score: {r2:.2f}')
print(f'Custom Accuracy-like Metric (within ±{threshold}): {accuracy_like:.2f}%')

# Optional: Display a few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

OUTPUT:

	Sepal length(cm)	Sepal width(cm)	Petal length(cm)	Petal width(cm)	target
0	5.1	3.5	1.4	0.2	1.4
1	4.9	3.0	1.4	0.2	1.4
2	4.7	3.2	1.3	0.2	1.3
3	4.6	3.1	1.5	0.2	1.5
4	5.0	3.6	1.4	0.2	1.4

[5 rows x 5 columns]

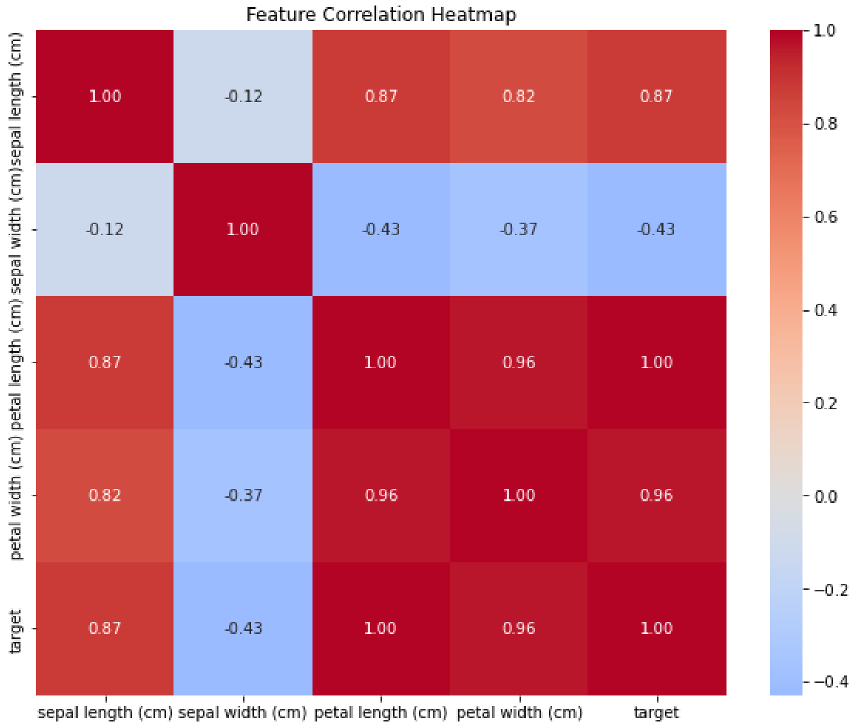
Mean Squared Error: 0.00

R-squared Score: 1.00

Custom Accuracy-like Metric (within ±0.1): 100.00%

	Actual	Predicted
73	4.7	4.7
18	1.7	1.7
118	6.9	6.9
75	4.5	4.5
76	4.8	4.8

Graph:



Classification model using Deep Learning algorithm

Python Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load_wine

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to_categorical

# Load the Wine dataset

data = load_wine()

X = data.data  # Features

y = data.target  # Target variable (wine classes)

# One-hot encoding the target variable for multi-class classification

y = to_categorical(y)

# Split the data into training and testing sets (80% training, 20% testing)

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

# Standardize the feature data (Deep learning models work better with normalized data)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

```
# Build the deep learning model
model = Sequential()

# Input layer with 13 features, and two hidden layers
model.add(Dense(128, input_shape=(X_train.shape[1],), activation='relu'))  # First hidden layer
model.add(Dense(64, activation='relu'))  # Second hidden layer

# Output layer with 3 neurons for the 3 wine classes and softmax activation for multi-class classification
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'\nTest Accuracy: {test_acc:.2f}')

# Predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Calculate accuracy
accuracy = accuracy_score(y_test_classes, y_pred_classes)
print(f'Accuracy: {accuracy:.2f}')

# Confusion matrix and classification report
```

```python
print("\nConfusion Matrix:")

print(confusion_matrix(y_test_classes, y_pred_classes))

print("\nClassification Report:")

print(classification_report(y_test_classes, y_pred_classes, target_names=data.target_names))

# Plotting training and validation accuracy over epochs

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

OUTPUT:

runfile('E:/untitled8.py', wdir='E:')

Epoch 1/50

5/5 ━━━━━━2s 119ms/step - accuracy: 0.4734 - loss: 1.0220 - val_accuracy: 0.8889 - val_loss: 0.7236

Epoch 2/50

5/5 ━━━━━━ 0s 20ms/step - accuracy: 0.7983 - loss: 0.7303 - val_accuracy: 0.9722 - val_loss: 0.5221

Epoch 3/50

5/5 ━━━━━━0s 21ms/step - accuracy: 0.9471 - loss: 0.5433 - val_accuracy: 1.0000 - val_loss: 0.3761

Epoch 4/50

5/5 ━━━━━━ 0s 16ms/step - accuracy: 0.9832 - loss: 0.3769 - val_accuracy: 1.0000 - val_loss: 0.2728

Epoch 5/50

Test Accuracy: 1.00

2/2 ━━━━━━━━ 0s 36ms/step

Accuracy: 1.00

Confusion Matrix:

[[14 0 0]

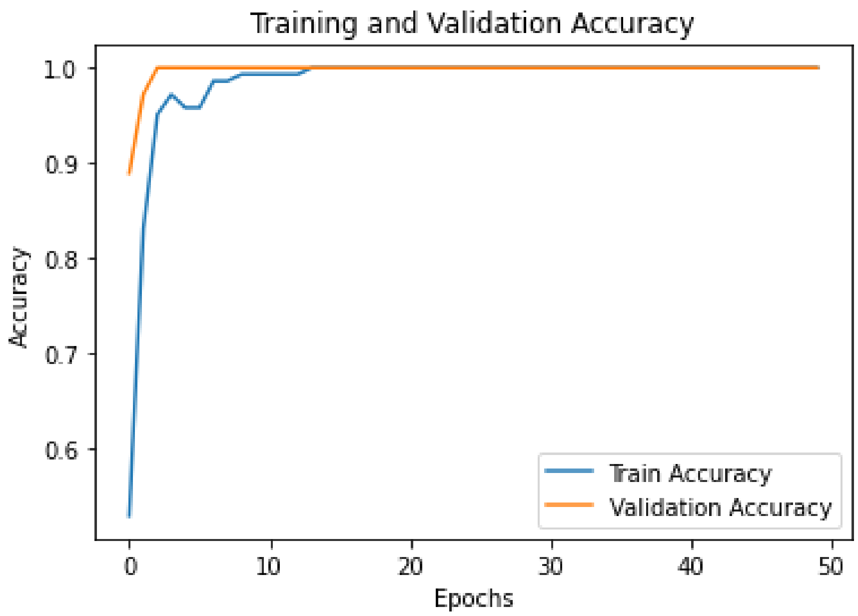
 [0 14 0]

 [0 0 8]]

Classification Report:

	precision	recall	f1-score	support
Class_0	1.00	1.00	1.00	14
Class_1	1.00	1.00	1.00	14
Class_2	1.00	1.00	1.00	8
			1.00	
accuracy			1.00	36
Macro avg	1.00	1.00	1.00	36
Weighted avg	1.00	1.00	1.00	36

Graph:



Classification model using Machine Learning algorithm

Python Code:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_wine

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import warnings

warnings.filterwarnings('ignore')

# Load the Wine dataset

data = load_wine()

X = data.data  # Features (chemical composition of wines)

y = data.target  # Target variable (wine classes)

# Split the data into training and testing sets (80% training, 20% testing)

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

# Create and fit the Logistic Regression model

model = LogisticRegression(max_iter=200)

model.fit(X_train, y_train)

# Predict on the test set

y_pred = model.predict(X_test)

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
```

```
# Print confusion matrix and classification report
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=data.target_names))

# Plotting (Optional)
# Visualize the first two features only
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', s=20)
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
plt.title('Wine Dataset (First Two Features)')
plt.colorbar(label='Wine Class (0, 1, 2)')
plt.show()
```

OUTPUT:

runfile('E:/untitled4.py', wdir='E:')

Accuracy: 0.97

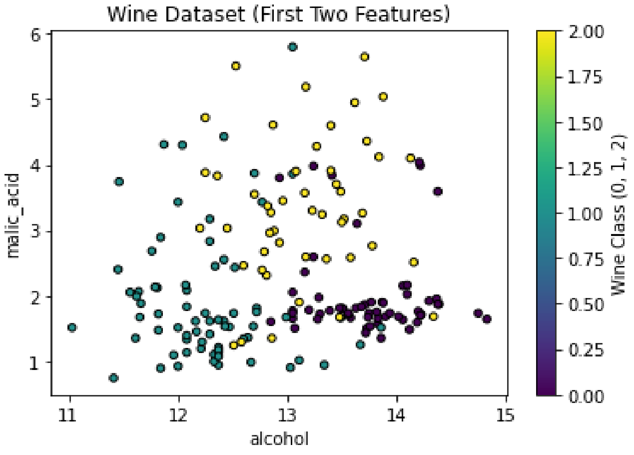
Confusion Matrix:

```
[[13  1  0]
 [ 0 14  0]
 [ 0  0  8]]
```

Classification Report:

	precision	recall	f1-score	support
Class_0	1.00	0.93	0.96	14
Class_1	0.93	1.00	0.97	14
Class_2	1.00	1.00	1.00	8
accuracy			0.97	36
Macro avg	0.98	0.98	0.98	36
Weighted avg	0.97	0.97	0.97	36

Graph:



Analyze the performance of ML and DL

1. Accuracy

- **ML:** Often sufficient for simpler tasks where data features are well-understood.
- **DL:** Generally achieves higher accuracy for complex tasks (e.g., image, speech recognition), but may require much more data.

2. Data Requirements

- **ML:** Performs well on small to medium-sized datasets, often relying on feature engineering.
- **DL:** Requires large amounts of data to train deep networks effectively and avoid overfitting.

3. Feature Engineering

- **ML:** Relies heavily on manual feature extraction and selection by domain experts.
- **DL:** Automatically learns features from raw data, reducing the need for manual feature engineering.

4. Model Complexity

- **ML:** Simpler models like decision trees, logistic regression, and support vector machines are easier to interpret and tune.
- **DL:** Complex neural networks with multiple layers (e.g., CNN, RNN) require advanced tuning and have higher complexity.

5. Training Time

- **ML:** Generally faster to train on small to medium-sized datasets due to simpler models.
- **DL:** Requires much more computational power and longer training times due to deep architectures and large datasets.

6. Computational Resources

- **ML:** Can be trained on standard CPUs and smaller-scale hardware.
- **DL:** Requires GPUs/TPUs and high-performance computing resources for efficient training, especially on large models.

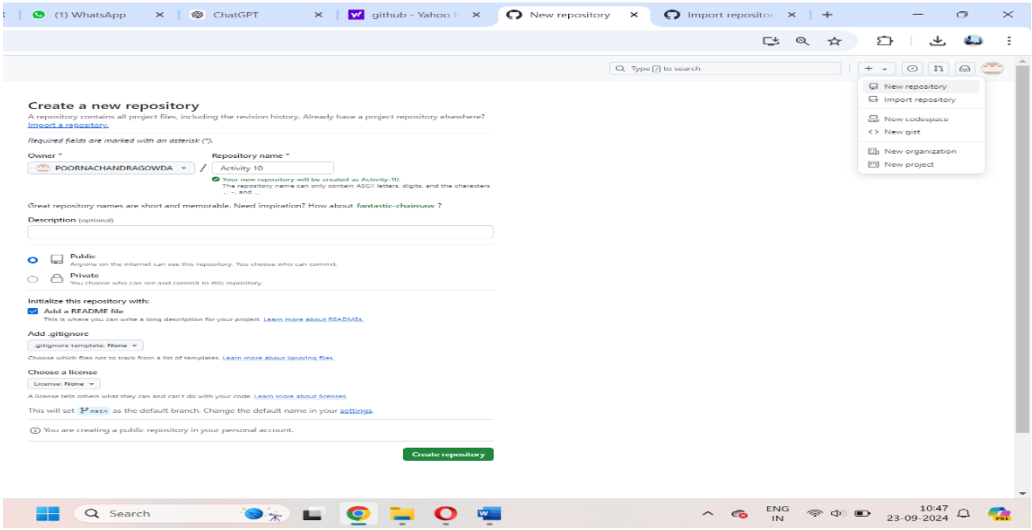
Uploading Files to GitHub

1. Sign in to GitHub

- **Open GitHub and sign in with your credentials.**

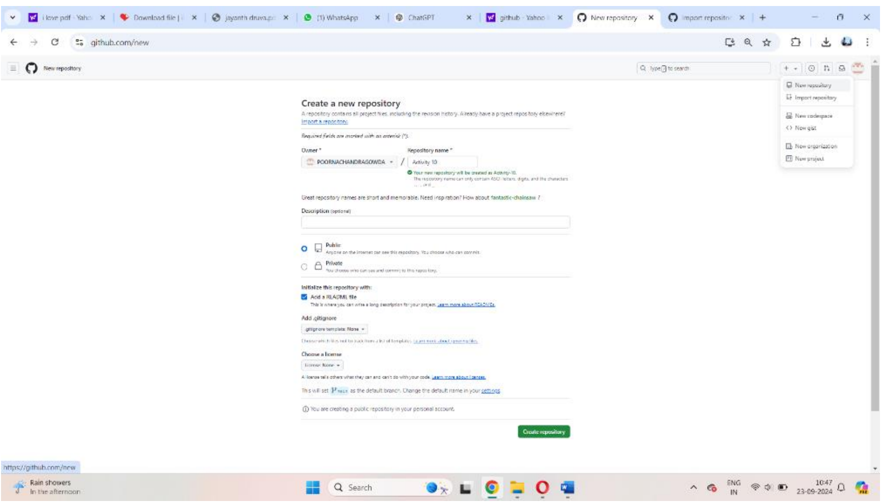
2. Create a New Repository

- Click on the "+" icon in the top-right corner and select "New repository".
- Give your repository a name and description (optional), and choose whether it should be public or private.
- Click "Create repository".



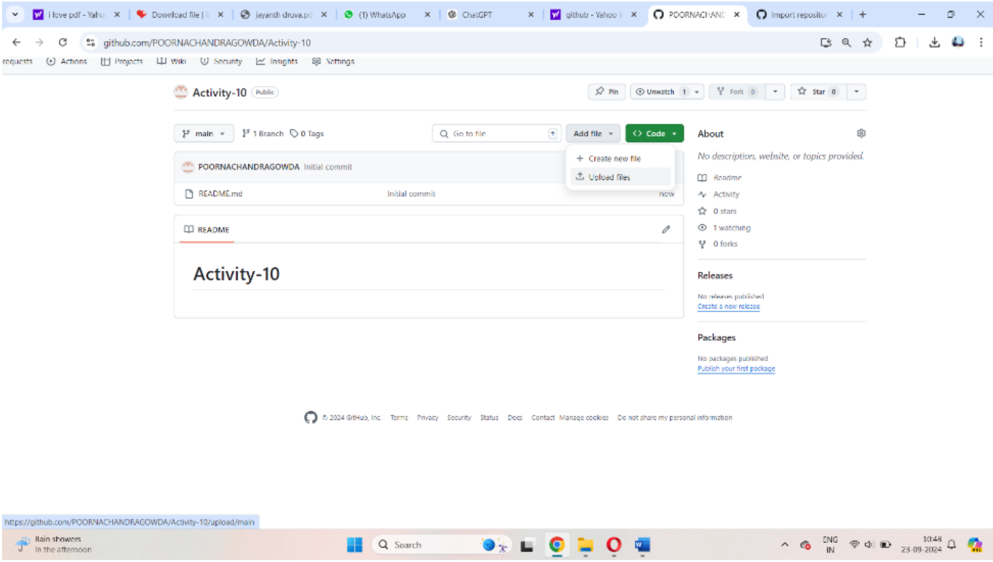
3. Navigate to Your Repository

- After creating the repository, you'll be directed to the repository's page.You'll see options like "Quick setup" if it's a new repository.



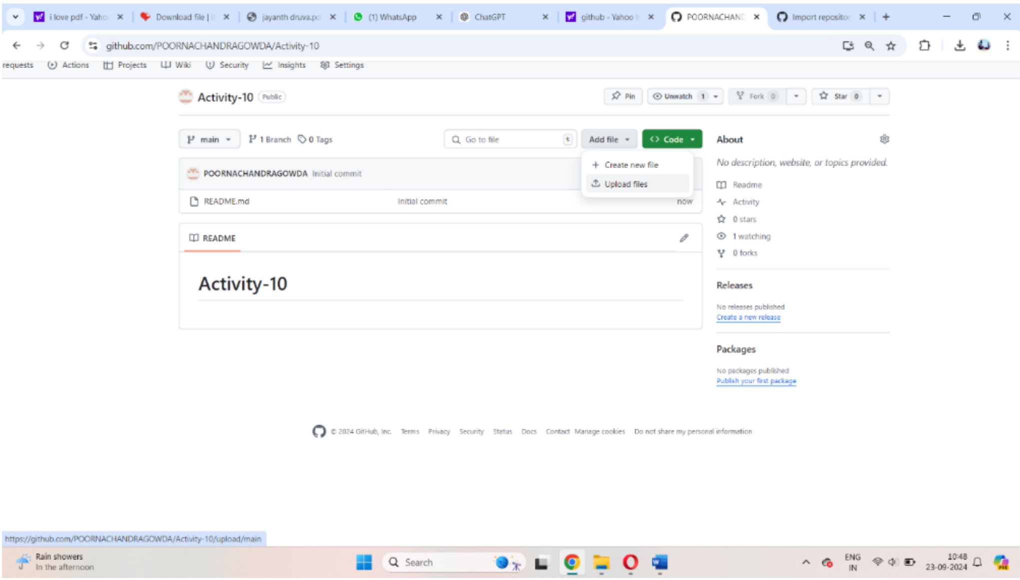
4. Click "Add File"

- On the repository page, click the "Add file" dropdown.
- Choose "Upload files" from the dropdown.



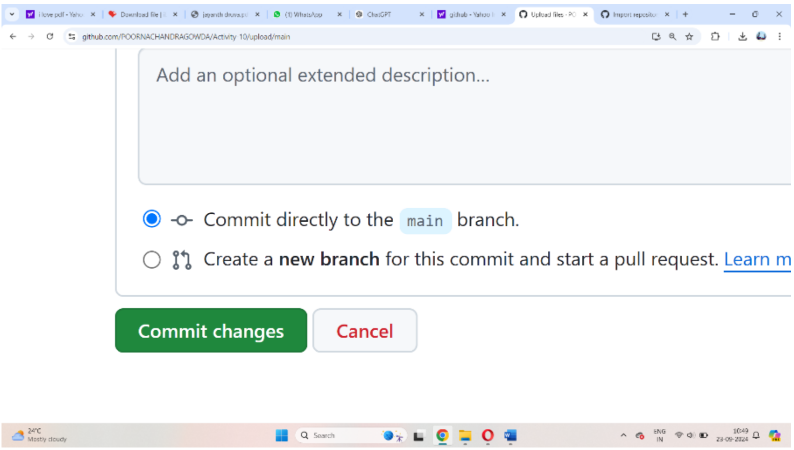
5. Select the File to Upload

- Drag and drop the file you want to upload or click "choose your files" to select a file from your computer.



6. Commit the File

- Once the file is uploaded, add a commit message in the "Commit changes" box (this is optional but recommended to describe the change).
- Click "Commit changes" to upload the file to GitHub.



Click https://github.com/POORNACHANDRAGOWDA/Activity -10.gitto view the uploaded file