ACTIVITY - 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 for Machine Learning Algorithm

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

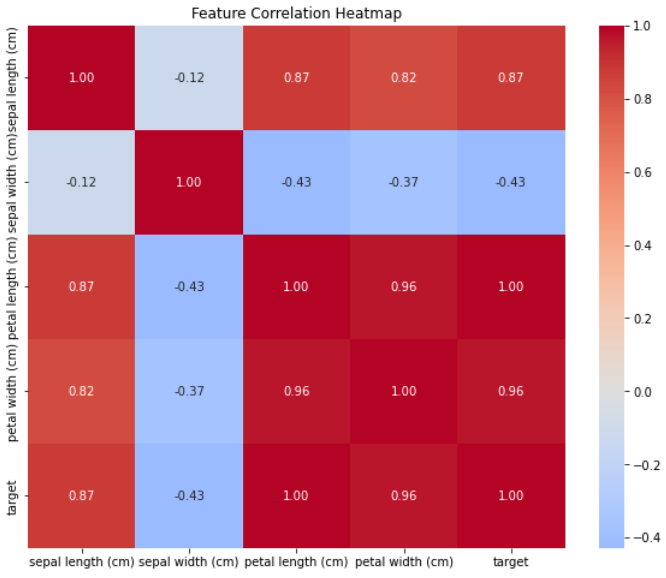
[5 rows x 5 columns]

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
78	4.5	4.5
76	4.8	4.8



Regression for Deep Learning Algorithm

```
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))
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
```

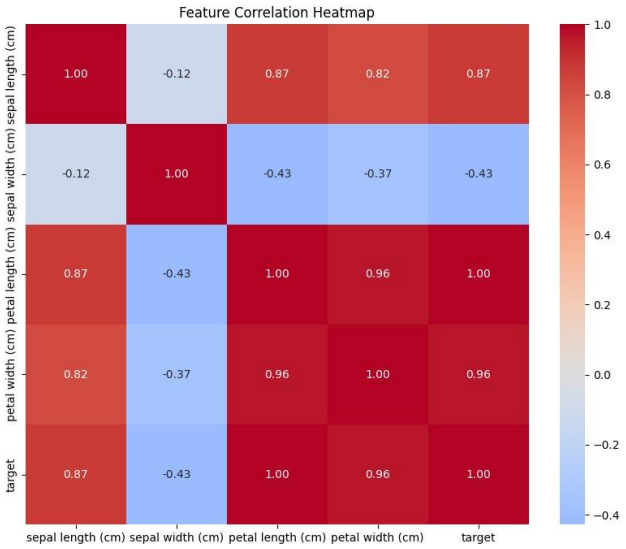
```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# 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)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

	target
0	1.4
1	1.4
2	1.3
3	1.5
4	1.4



Epoch 1/100

4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 103ms/step - loss: 19.3726 - val_loss: 18.6297

Epoch 2/100

4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 16.1130 - val_loss: 17.2258

Epoch 3/100

4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 15.5343 - val_loss: 15.8627

Epoch 4/100

4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 13.6319 - val_loss: 14.5409

Epoch 5/100

4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 13.1883 - val_loss: 13.2545

Classification for Machine Learning Algorithm

```
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 LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns

# 1. Load and prepare data
iris_data = load_iris()
data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
data['target'] = iris_data.target
# 2. Display the first few rows of the dataset
print(data.head())

# 3. Visualize the data distribution using pairplot
plt.figure(figsize=(10, 6))
sns.pairplot(data, hue='target', palette='Set2')
plt.title('Iris Dataset Pairplot')
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 Logistic Regression model
model = LogisticRegression()

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

# 8. Make predictions
y_pred = model.predict(X_test)
# 9. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')
print('Classification Report:\n', class_report)

# 10. Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=iris_data.target_names, yticklabels=iris_data.target_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
# 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 width (cm)	target
0	5.1	3.5	...	0.2	0
1	4.9	3.0	...	0.2	0
2	4.7	3.2	...	0.2	0
3	4.6	3.1	...	0.2	0
4	5.0	3.6	...	0.2	0

[5 rows x 5 columns]

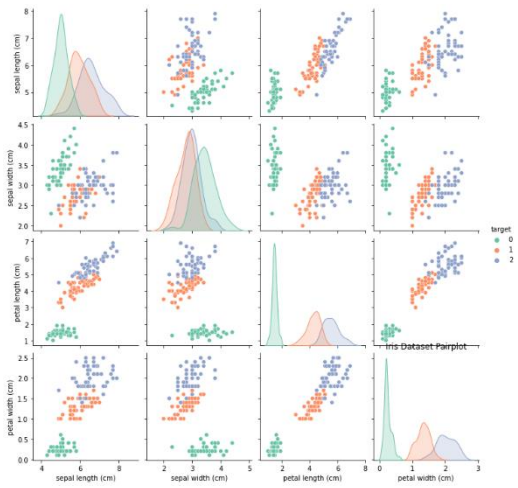
<Figure size 720x432 with 0 Axes>

Accuracy: 100.00%

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

	Actual	Predicted
73	1	1
18	0	0
118	2	2
78	1	1
76	1	1



Classification for Deep Learning Algorithm

```
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 accuracy_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.target

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

# 3. Visualize the data distribution
sns.pairplot(data, hue='target', palette='Set2')
plt.title('Iris Dataset Pairplot')
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)
```

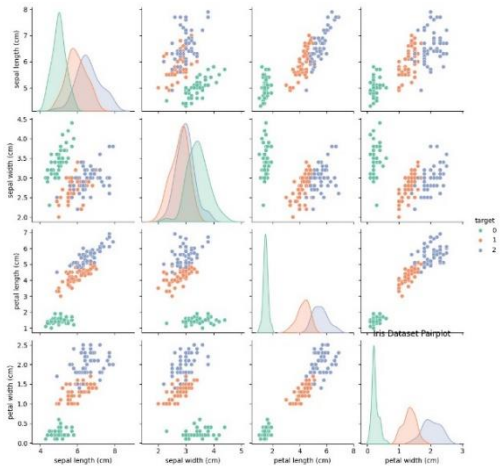
```
# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 6. Build the 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(3, activation='softmax')  # 3 classes for iris species
])

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

# 8. Train the model
history = model.fit(X_train, y_train, epochs=100, validation_split=0.2)

# 9. Evaluate the model
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Convert probabilities to class labels

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_classes)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Optional: Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
```

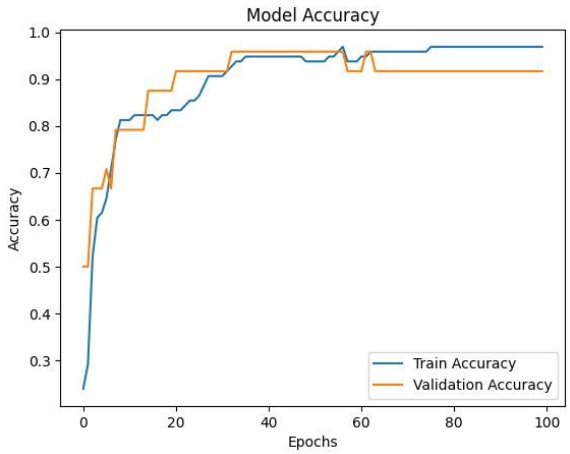
```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

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

OUTPUT:

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

	target
0	0
1	0
2	0
3	0
4	0





3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 221ms/step - accuracy: 0.2448 - loss: 1.1763 - val_accuracy: 0.5000 - val_loss: 1.0829

Epoch 2/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.2826 - loss: 1.1083 - val_accuracy: 0.5000 - val_loss: 1.0171

Epoch 3/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.4792 - loss: 1.0535 - val_accuracy: 0.6667 - val_loss: 0.9556

Epoch 4/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.5404 - loss: 1.0086 - val_accuracy: 0.6667 - val_loss: 0.9020

Epoch 5/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.6042 - loss: 0.9552 - val_accuracy: 0.6667 - val_loss: 0.8538

Analyze the performance of ML and DL

1. Data Requirements

ML: Generally requires less data to achieve good performance. It can work well with small to medium-sized datasets, especially with effective feature engineering.

DL: Typically requires large amounts of data to perform optimally. It excels in scenarios with extensive datasets, such as images or text.

2. Model Complexity

ML: Models can be simpler and more interpretable (e.g., linear regression, decision trees). This simplicity often leads to faster training times.

DL: Models are usually more complex (e.g., neural networks with multiple layers) and may require significant computational resources and time to train.

3. Training Time

ML: Training times are generally shorter due to less complexity, allowing for quicker iterations and model tuning.

DL: Training can take hours or even days, particularly for large datasets and deep networks. GPUs are often needed to expedite the process.

4. Overfitting

ML: More prone to overfitting if models are too complex relative to the amount of data. Regularization techniques can mitigate this.

DL: Also prone to overfitting, especially in deep networks. Techniques such as dropout, early stopping, and data augmentation are commonly used to combat this.

5. Feature Engineering

ML: Often requires substantial manual feature engineering and selection. Success heavily relies on the quality of features extracted from the data.

DL: Can automatically learn relevant features from raw data, particularly beneficial for unstructured data (e.g., images, audio).

6. Performance Metrics

ML: Common metrics include accuracy, precision, recall, F1 score, MSE, and R^2, depending on the problem (classification or regression).

DL: Uses similar metrics but can also leverage more specialized ones, such as IoU (Intersection over Union) for object detection tasks.

7. Interpretability

ML: Models like decision trees and linear regression are generally easier to interpret, making it simpler to understand feature contributions.

DL: Often described as "black boxes," where understanding the decision-making process is more challenging, though techniques like SHAP and LIME can help.

8. Deployment and Scalability

ML: Generally easier and quicker to deploy in production environments due to lighter model complexity.

DL: Deployment can be more complex, especially if GPU resources are required. However, frameworks and libraries have improved deployment ease.

9. Use Cases

ML: Suited for structured data tasks such as fraud detection, customer segmentation, and time series forecasting.

DL: Ideal for unstructured data tasks like image classification, natural language processing, and speech recognition.

10. Community and Ecosystem

ML: A long-established field with a wealth of resources, libraries, and documentation (e.g., scikit-learn).

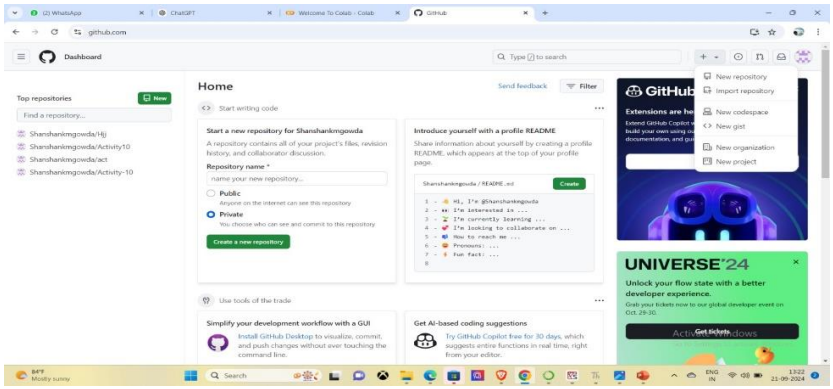
DL: Rapidly evolving with a strong community and extensive frameworks (e.g., TensorFlow, PyTorch) that support complex model development.

Creating Git Repository:

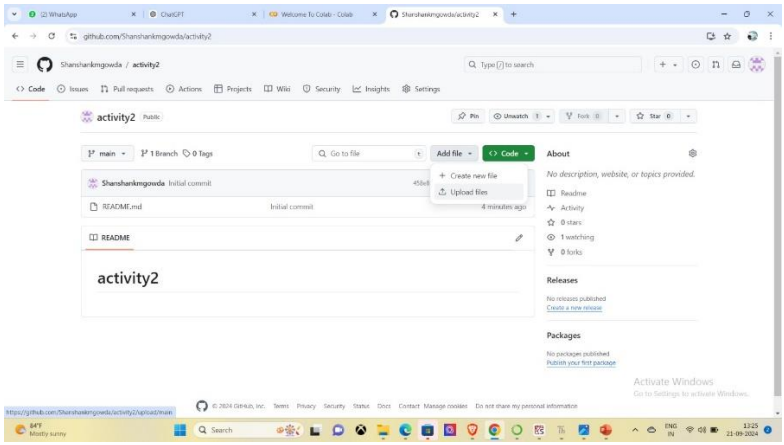
Using the GitHub Web Interface

1. Sign In: Go to GitHub and log in to your account.

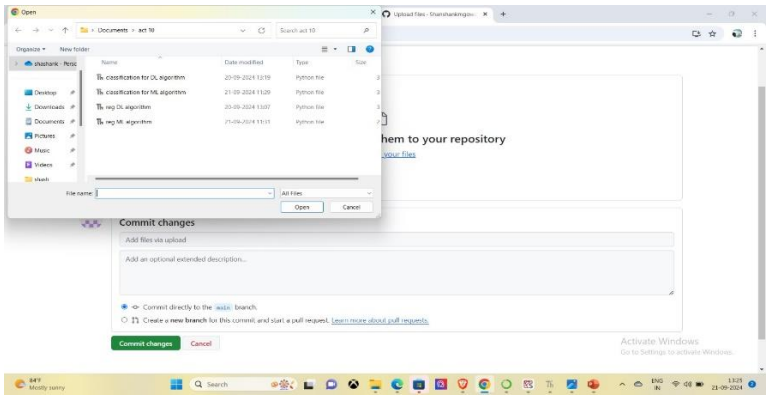
2. Navigate to Repository: Click on the repository where you want to upload the file. If you don’t have one, you can create a new repository by clicking the "+" icon in the top right corner and selecting "New repository."



3. Go to Upload Files: In the repository, click on the "Add file" button (usually found near the top right) and select "Upload files."



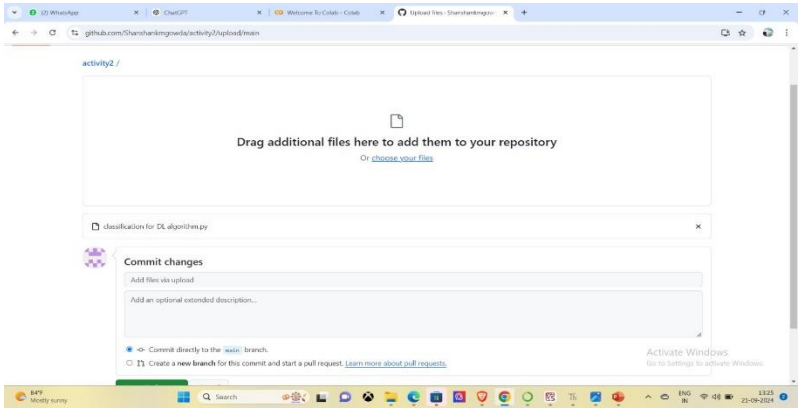
4. Drag and Drop or Choose Files: You can either drag and drop your file(s) into the upload area or click "choose your files" to select files from your computer.



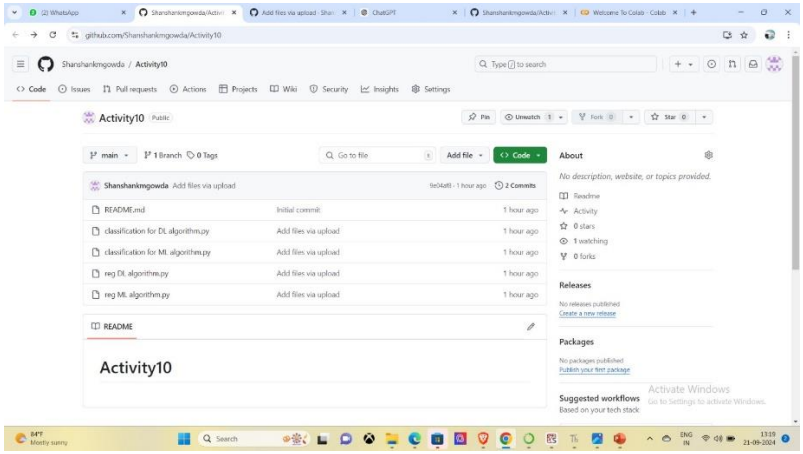
5.Review Your Changes: Once the files are uploaded, you'll see a list of them. Review to ensure everything is correct.

6. Commit Changes: Scroll down to the "Commit changes" section. Add a commit message describing what you're uploading (e.g., "Add my_file.txt").

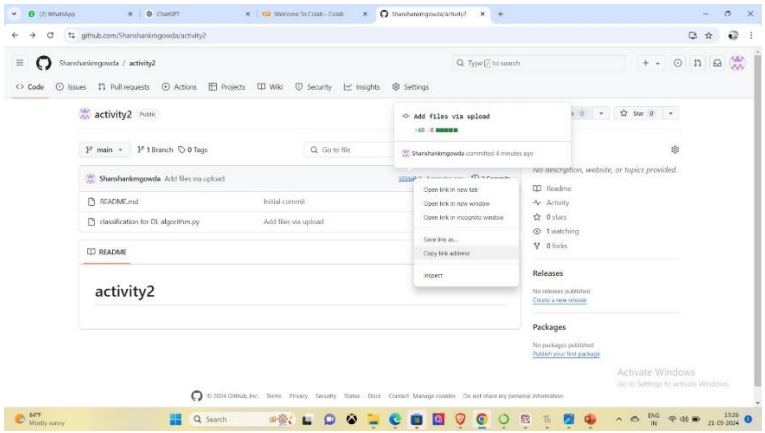
7. Click Commit: Click the green "Commit changes" button to upload the file.



8.Verify Upload: Check your repository to confirm that the file has been uploaded successfully.



9. Share the Link : You can now share the link to the file or the repository with others.



10.Repository Link: Click here https://github.com/Shanshankmgowda/Activity10 to verify the uploaded files.