**ACTIVITY: 10**

**Regression - Rebuild with deep learning model**

**Problem statement:** In this project, we successfully developed a deep learning model to predict petal width using the Iris dataset. After preparing the data and analysing feature correlations, we split the dataset into training, validation, and test sets, normalizing the features to enhance model performance. We built a feed forward neural network with two hidden layers, achieving satisfactory results indicated by mean squared error and R-squared metrics. The model effectively captured the relationships in the data, as demonstrated by our accuracy-like metric measuring predictions within a defined threshold. This project highlights the potential of neural networks in regression tasks and sets the stage for further exploration and model optimization.

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:**

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - loss: 14.1944 - val\_loss: 15.0136

Epoch 4/100

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 13.2979 - val\_loss: 13.6908

Epoch 5/100

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 12.5030 - val\_loss: 12.3585

Epoch 6/100

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 10.8817 - val\_loss: 11.0777

Epoch 7/100

Mean Squared Error: 0.03

R-squared Score: 0.99

Actual Predicted

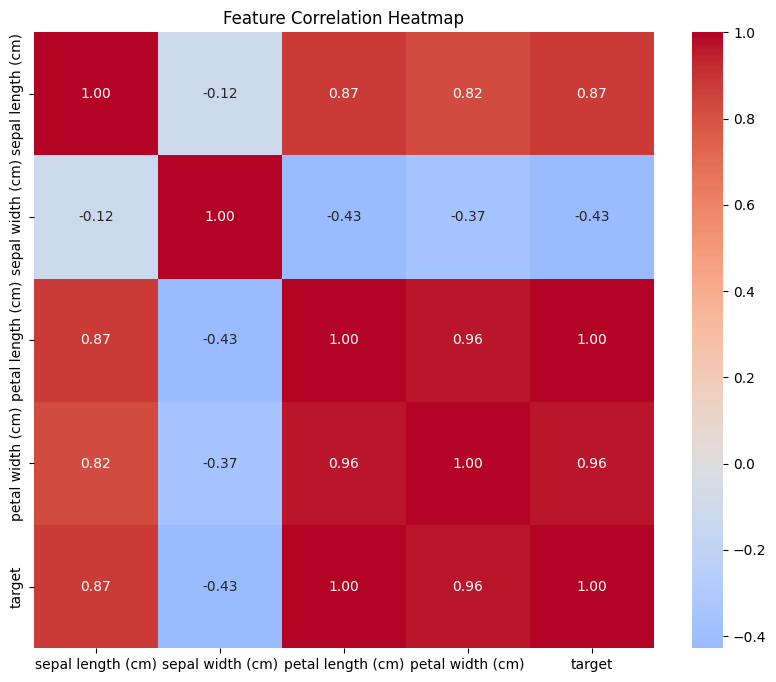
143 5.9 5.900316

56 4.7 4.319361

128 5.6 5.713876

69 3.9 3.834024

68 4.5 4.914990

**Graph:**

**Regression - Rebuild with machine learning model**

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'Accuracy(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:**

Mean Squared Error: 0.00

R-squared Score: 1.00

Accuracy(within ±0.1): 100.00%

Actual Predicted

73 4.7 4.7

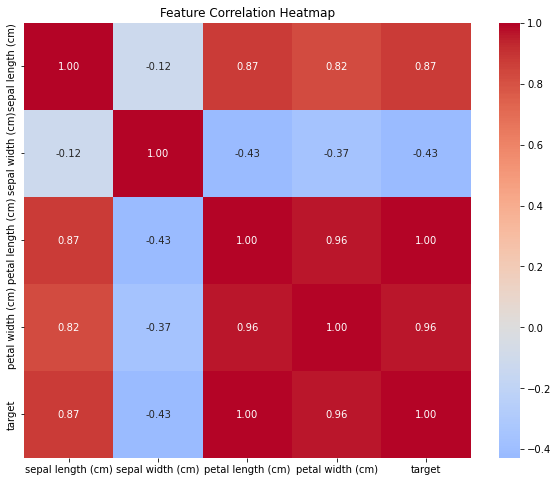
18 1.7 1.7

118 6.9 6.9

78 4.5 4.5

76 4.8 4.8

**Graph:**



**Classification - Rebuild with deep learning model**

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:**

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - accuracy: 0.3216 - loss: 1.2119 - val\_accuracy: 0.2917 - val\_loss: 1.0809

Epoch 2/100

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3815 - loss: 1.1034 - val\_accuracy: 0.3750 - val\_loss: 1.0115

Epoch 3/100

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.4440 - loss: 1.0391 - val\_accuracy: 0.5833 - val\_loss: 0.9489

Epoch 4/100

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4544 - loss: 0.9871 - val\_accuracy: 0.7083 - val\_loss: 0.8893

Epoch 5/100

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4492 - loss: 0.9308 - val\_accuracy: 0.7500 - val\_loss: 0.8363

Epoch 6/100

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5404 - loss: 0.8903 - val\_accuracy: 0.7917 - val\_loss: 0.7892

Epoch 7/100

Accuracy: 96.67%

Actual Predicted

73 1 1

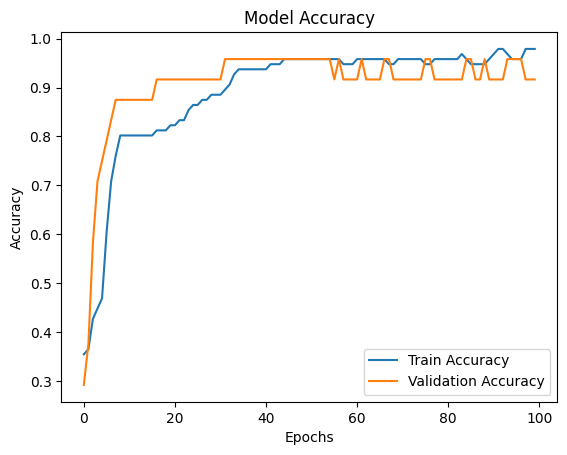
18 0 0

118 2 2

78 1 1

76 1 1

Graph:



**Classification - Rebuild with machine learning model**

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:**

Accuracy: 98.00%

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

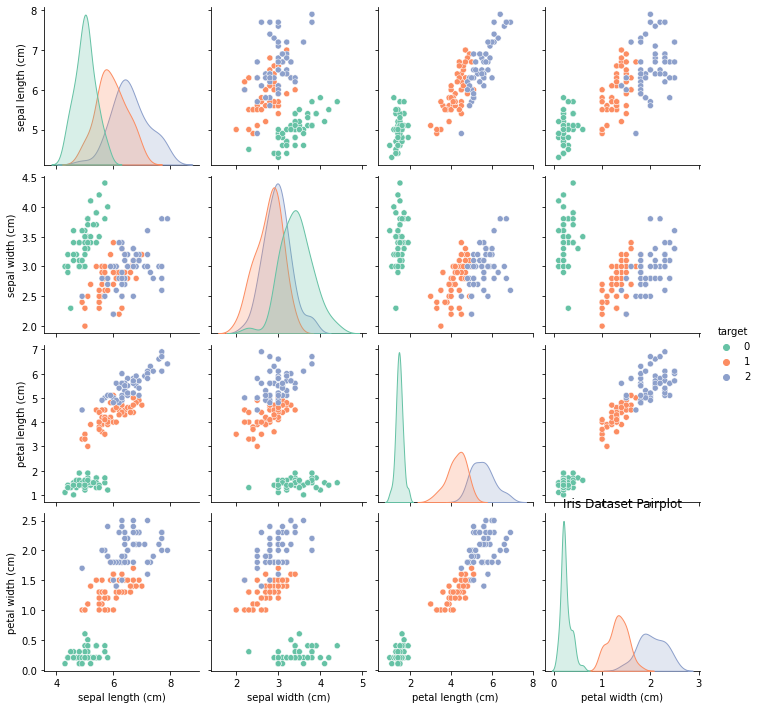
18 0 0

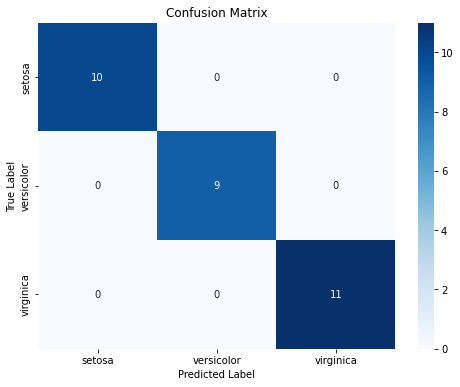
118 2 2

78 1 1

76 1 1

**Graph:**

****



**Analyze the performance of ML and DL**

**1.Model Complexity**

* **ML**: Simpler algorithms (e.g., linear regression, decision trees).
* **DL**: Complex architectures (e.g., CNNs, RNNs).

**2. Data Requirements**

* **ML**: Effective with smaller, structured datasets.
* **DL**: Requires large datasets to perform well.

**3. Training Time**

* **ML**: Faster training times, less computational power needed.
* **DL**: Longer training times, often requiring GPUs.

**4. Interpretability**

* **ML**: Generally more interpretable and easier to understand.
* **DL**: Often seen as a “black box,” harder to interpret.

**5. Generalization**

* **ML**: Good generalization with proper feature engineering.
* **DL**: Excellent generalization from complex patterns, but risk of over fitting.

**6. Feature Engineering**

* **ML**: Often requires manual feature selection and engineering.
* **DL**: Automatically learns features from raw data.

**7. Use Cases**

* **ML**: Common in finance, healthcare, and marketing.
* **DL**: Dominates in computer vision, NLP, and speech recognition.

**8. Performance Metrics**

* **ML & DL**: Evaluated using accuracy, precision, recall, F1 score, etc., but DL models may need more hyper parameter tuning.

**9. Deployment Complexity**

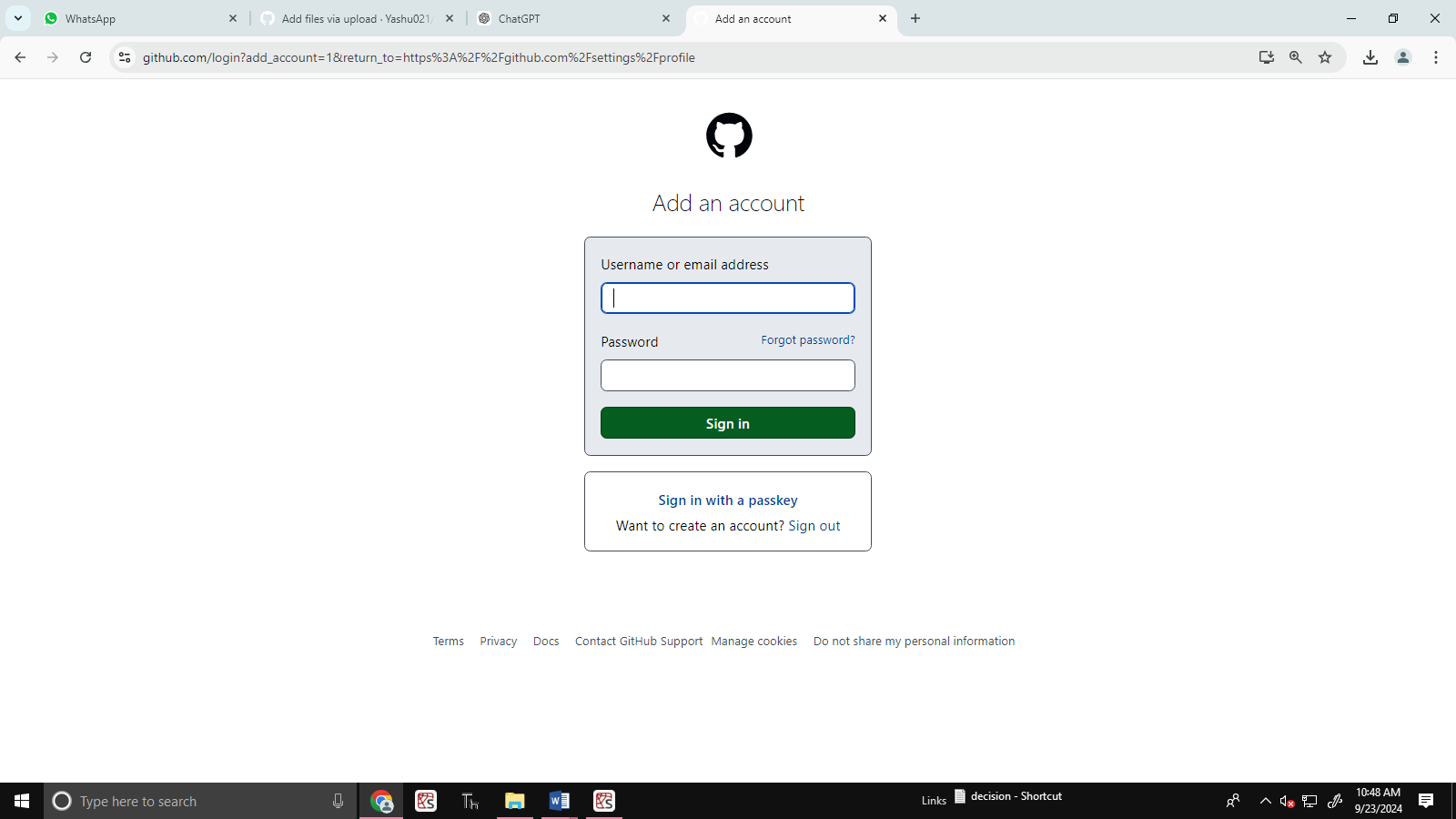
* **ML**: Generally easier to deploy and integrate.
* **DL**: Can be more complex due to model size and resource requirements.

**Uploading Files to GitHub**

**Creating Git repository for the project:**

### 1. ****Sign In****

* Go to [github.com](https://github.com) and log in or create an account.

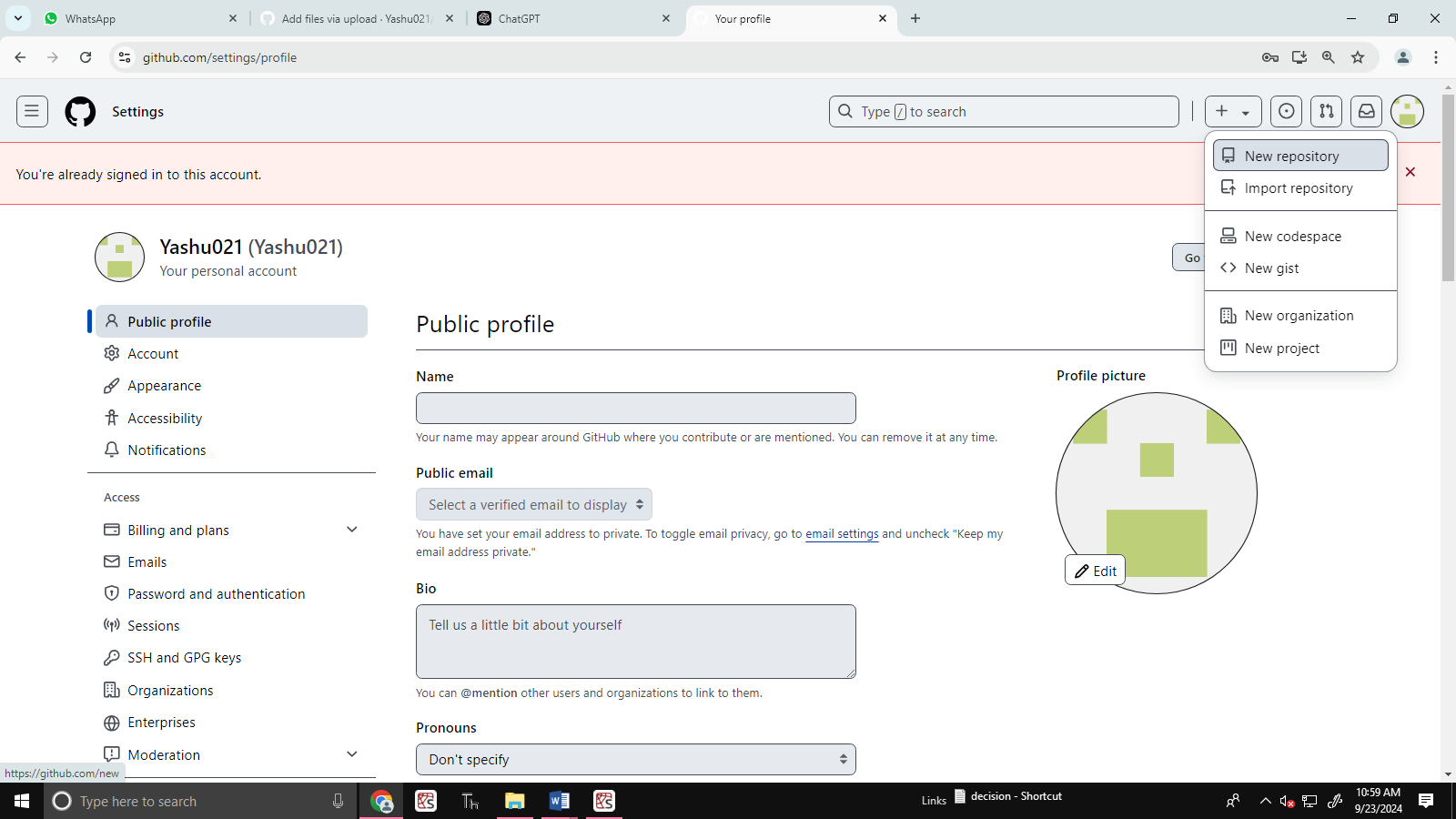


### 2. ****Navigate to Repositories****

* Click the **"+"** icon in the upper right corner.

### 3. ****Select New Repository****

* Choose **"New repository"** from the dropdown menu.



### 4. ****Enter Repository Name****

* Fill in a unique name for your repository.

### 5. ****Add Description (Optional)****

* Optionally, write a brief description of your project.

### 6. ****Choose Visibility****

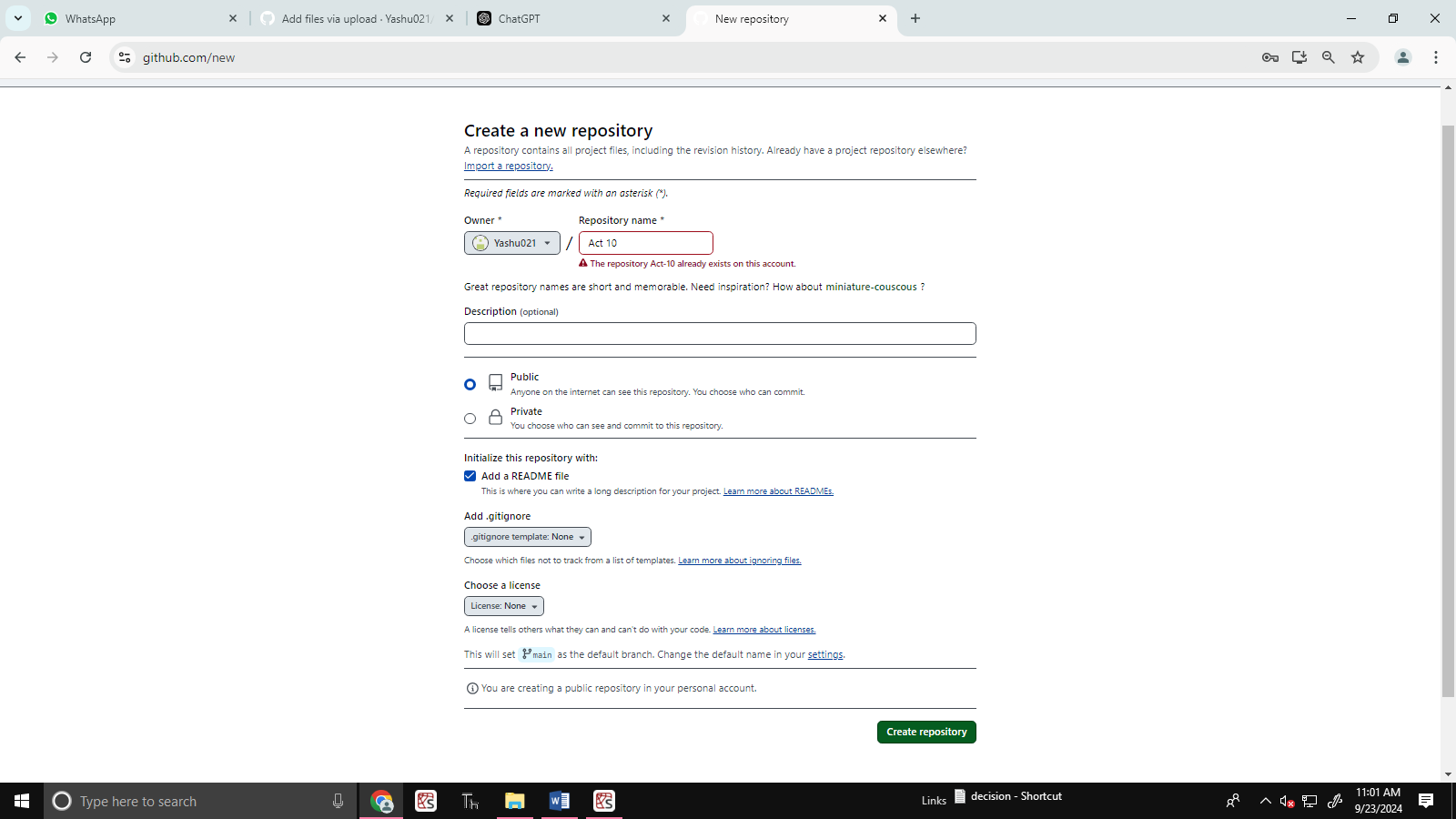
* Select **Public** or **Private** for the repository’s visibility.

### 7. ****Initialize Options****

* Optionally check to add a **README**, select a **.gitignore** template, or choose a **license**.

### 8. ****Create the Repository****

* Click the **"Create repository"** button at the bottom.



### 9. ****Add a File****

* Click on **"Add file"**, then select **"Create new file."**
* Enter the file name and add your content in the text editor.

### 10. ****Commit Changes****

### **Enter a commit message, choose commit options,and click **“Commit changes”****

**Click** <https://github.com/Yashu021/act-10/commit/bc750f5a924599c2dcb6b0383038527fa4ff426b> **to view the uploaded file**