Experiment No.4
AI for disease Prognosis
Date of Performance: 16/08/2024
Date of Submission: 11/09/2024

Aim: To perform AI for prognosis of a diseases

Objective: The objective of this experiment is to develop an AI-powered disease prognosis system that employs advanced machine learning algorithms, specifically convolutional neural networks (CNNs) for medical imaging and recurrent neural networks (RNNs) for time series data.

Theory:

The objective of using AI for the prognosis of diseases is to improve the accuracy and effectiveness of disease prediction, allowing for earlier and more precise diagnoses and treatment. AI can be a valuable tool in healthcare for a variety of purposes related to disease prognosis:

Early Detection: Detect diseases or medical conditions at an earlier stage when treatment is more effective, potentially saving lives and reducing the severity of the disease's impact.

Risk Assessment: Assess an individual's risk of developing a particular disease based on a range of factors, such as genetics, lifestyle, and medical history.

Personalized Medicine: Tailor treatment plans to individual patients based on their unique characteristics and needs, improving the effectiveness of treatments and reducing side effects.

Predictive Analytics: Use historical patient data to predict disease outcomes and recommend appropriate interventions or treatments.

Resource Allocation: Optimize healthcare resource allocation by identifying patients at higher risk and allocating resources accordingly. This can be especially important in resource-constrained healthcare systems.

Patient Engagement: Engage patients in their own healthcare by providing them with personalized health insights and recommendations.

Reducing Healthcare Costs: Improve the efficiency of healthcare systems by reducing unnecessary tests and treatments through more accurate prognosis and diagnosis.

Research and Drug Development: Assist researchers in identifying potential drug candidates or treatment strategies based on AI analysis of disease pathways and patient data.

Public Health Planning: Help public health officials and organizations plan for disease outbreaks, resource allocation, and prevention strategies.

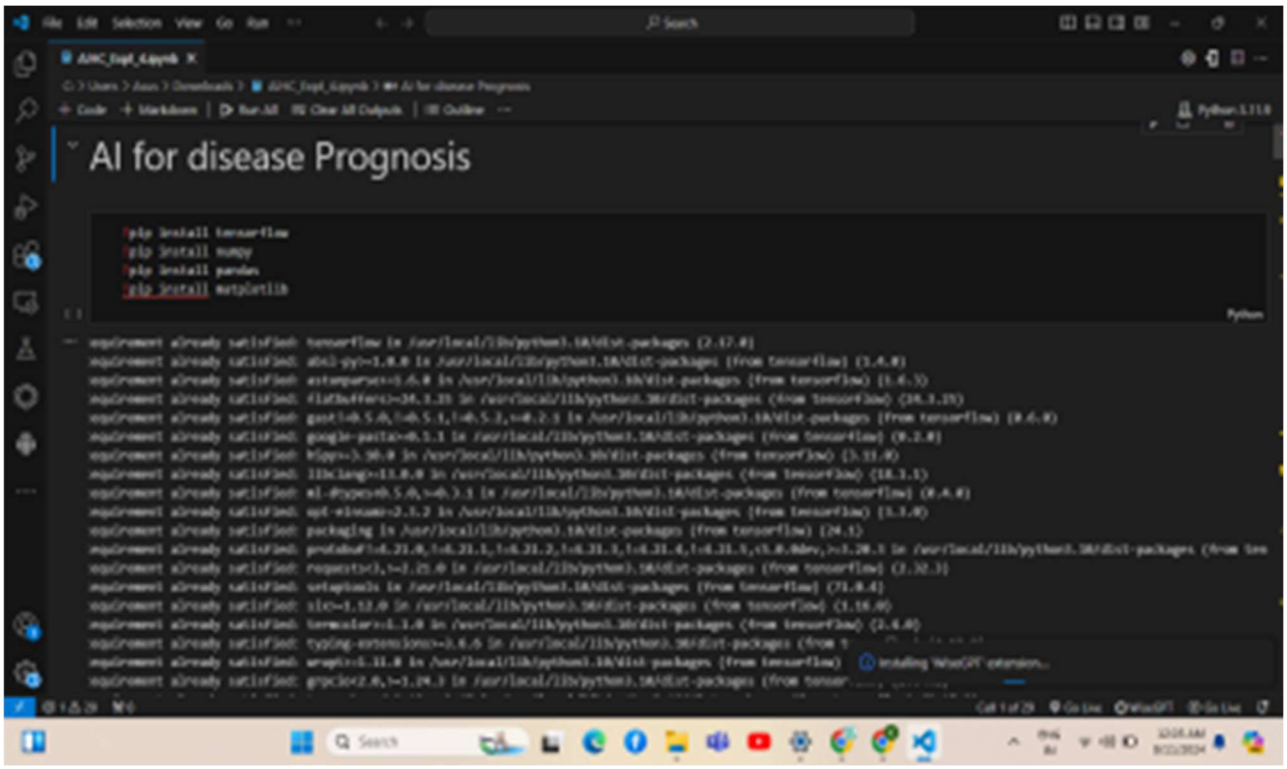
Chronic Disease Management: Aid in the management of chronic diseases by providing ongoing monitoring and early warnings of potential complications.

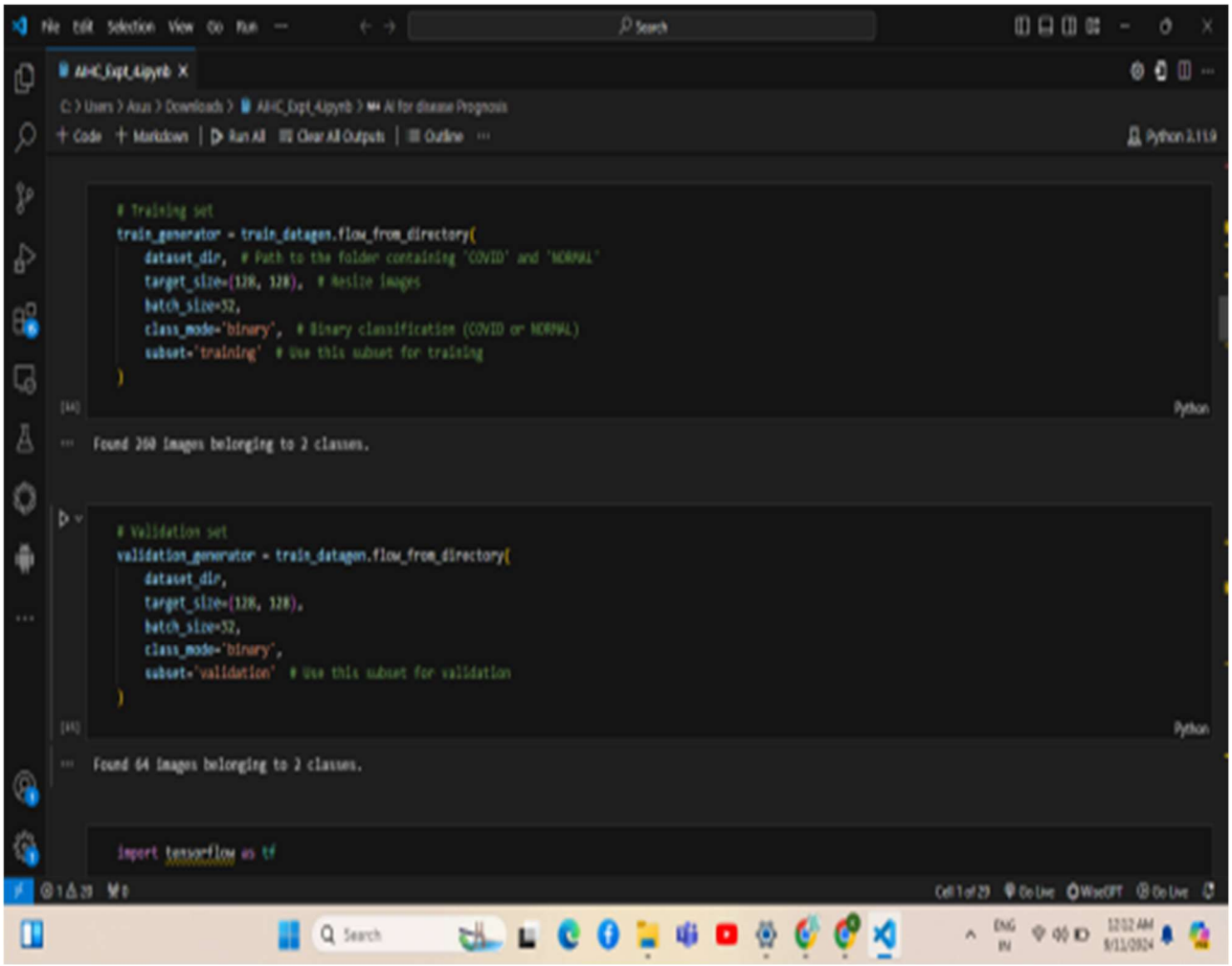
Telemedicine: Enable remote monitoring and diagnosis, particularly in underserved or remote areas.

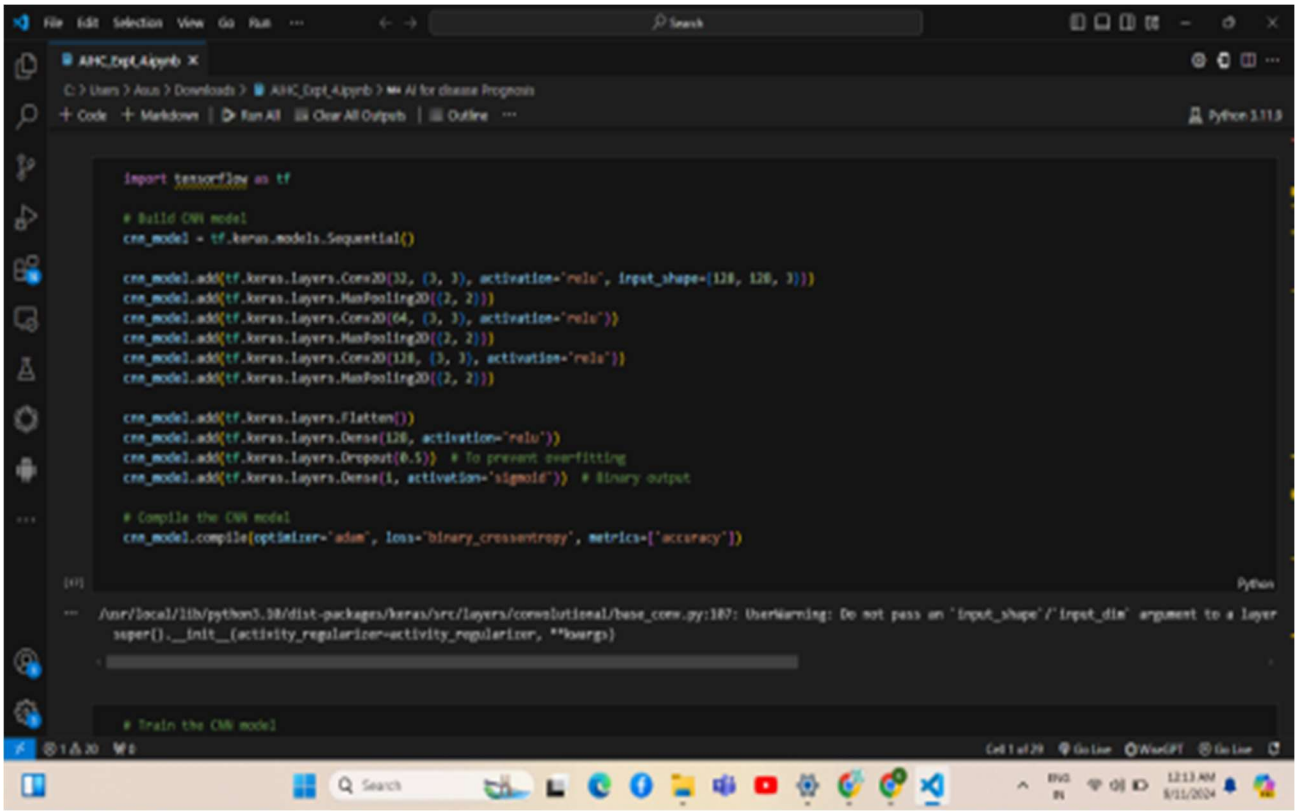
Quality of Life Improvement: Enhance the quality of life for patients by enabling proactive and preventive healthcare rather than just reactive treatment.

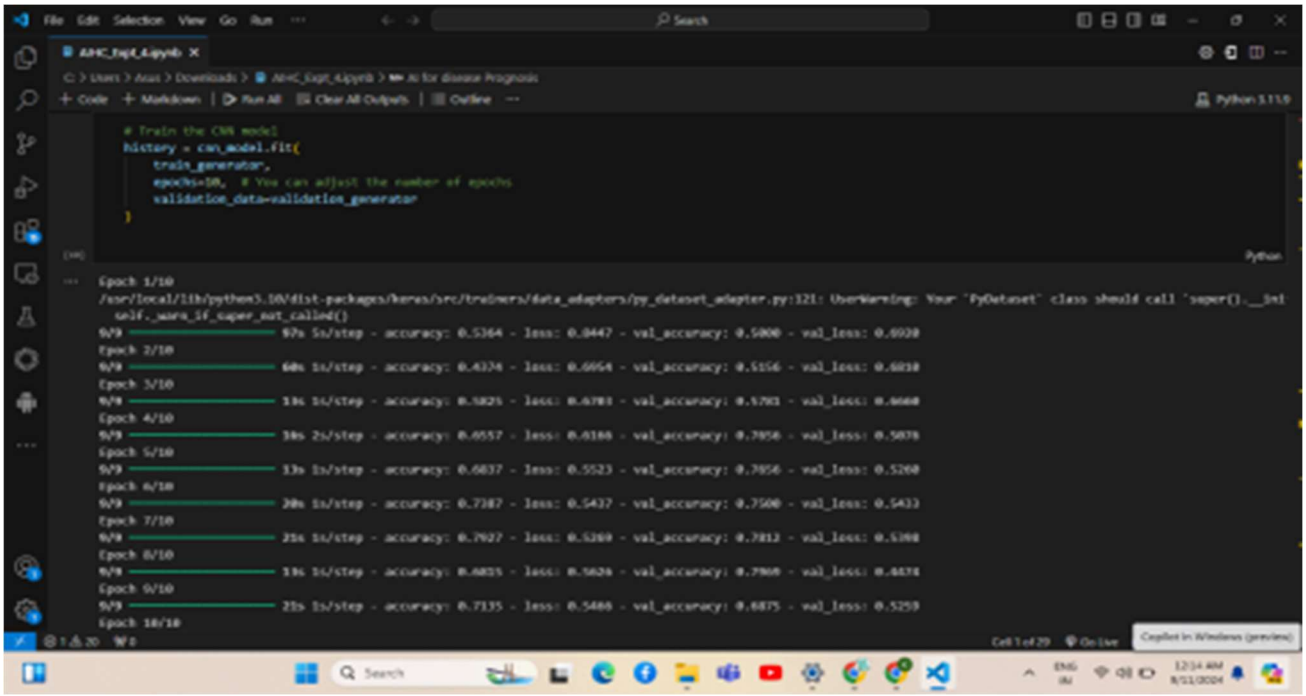
In summary, the primary objective of using AI for disease prognosis is to improve healthcare outcomes, reduce costs, and enhance the overall quality of care by providing more accurate, personalized, and timely information to healthcare providers, researchers, and patients. AI has the potential to transform the healthcare industry by making disease prognosis and management more data-driven and patient-centric.

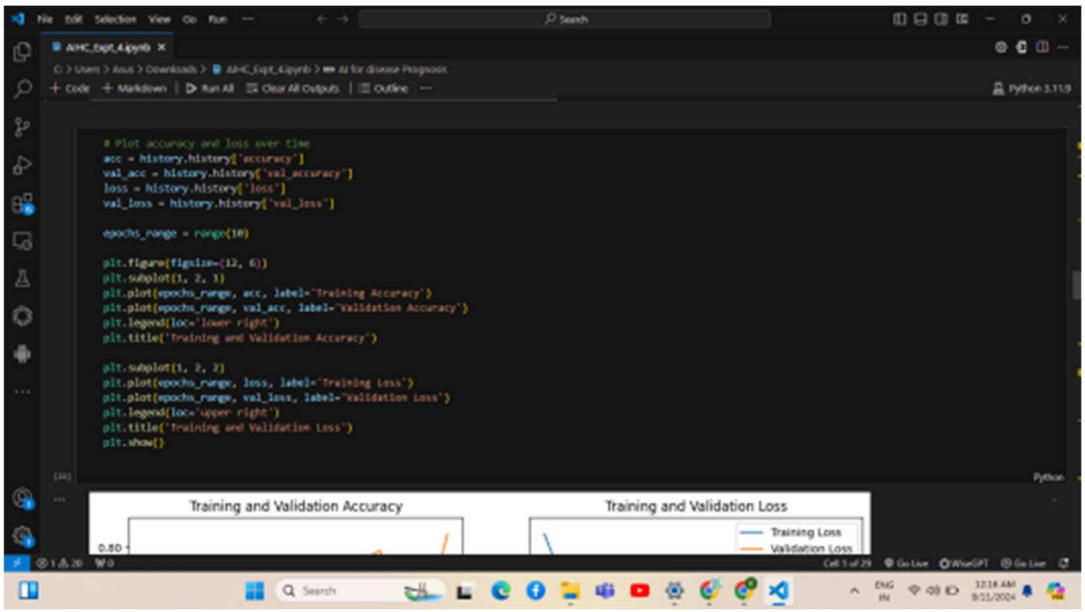
Code: -

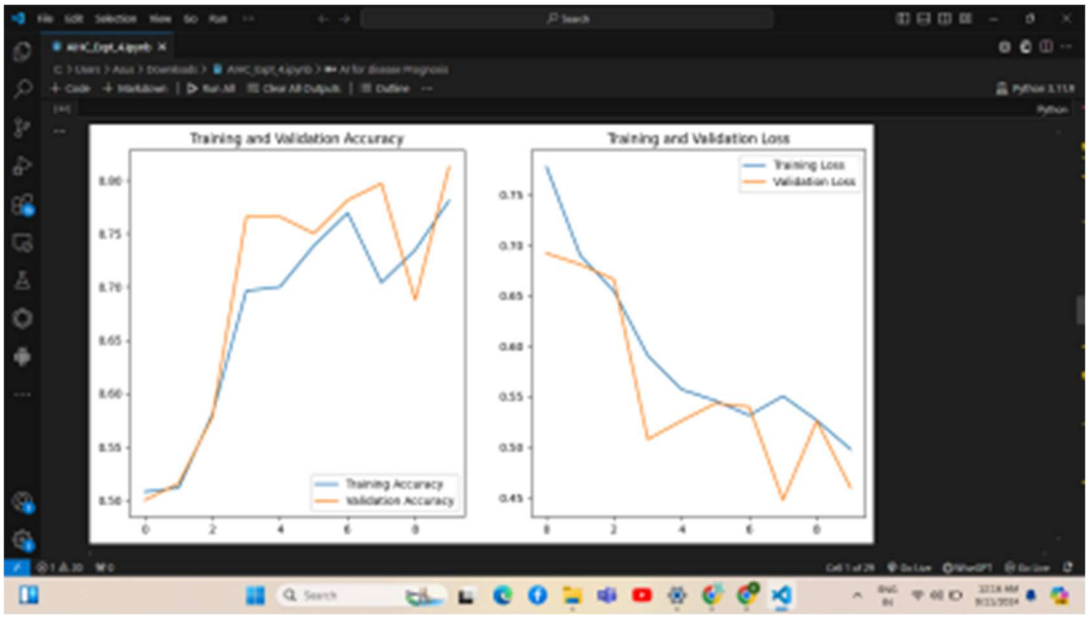

AI for disease Prognosis
!pip install tensorflow
!pip install numpy
!pip install pandas
!pip install matplotlib


AI for disease Prognosis
# Training set
train_generator = train_datagen.flow_from_directory(
dataset_dir, # Path to the folder containing 'COVID' and 'NORMAL'
target_size=(128, 128), # Resize images
batch_size=32,
class_mode='binary', # Binary classification (COVID or NORMAL)
subset='training' # Use this subset for training
)
Found 260 images belonging to 2 classes.
# Validation set
validation_generator = train_datagen.flow_from_directory(
dataset_dir,
target_size=(128, 128),
batch_size=32,
class_mode='binary',
subset='validation' # Use this subset for validation
)
Found 64 images belonging to 2 classes.
import tensorflow as tf


AIHC_Expt_4.ipynb
import tensorflow as tf
# Build CNN model
cnn_model = tf.keras.models.Sequential()
cnn_model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_model.add(tf.keras.layers.Flatten())
cnn_model.add(tf.keras.layers.Dense(128, activation='relu'))
cnn_model.add(tf.keras.layers.Dropout(0.5)) # To prevent overfitting
cnn_model.add(tf.keras.layers.Dense(1, activation='sigmoid')) # Binary output
# Compile the CNN model
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
# Train the CNN model


AIHC_Expt_4.ipynb
# Train the CNN model
history = cnn_model.fit(
train_generator,
epochs=10, # You can adjust the number of epochs
validation_data=validation_generator
)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__ini
self._warn_if_super_not_called()
9/9 ━━━━━━━━━━━━━━━━━━━━ 97s 5s/step - accuracy: 0.5364 - loss: 0.0447 - val_accuracy: 0.5000 - val_loss: 0.6930
Epoch 2/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 60s 1s/step - accuracy: 0.4374 - loss: 0.6954 - val_accuracy: 0.5156 - val_loss: 0.6830
Epoch 3/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.5825 - loss: 0.6703 - val_accuracy: 0.5703 - val_loss: 0.6660
Epoch 4/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 34s 2s/step - accuracy: 0.6557 - loss: 0.6166 - val_accuracy: 0.7056 - val_loss: 0.5876
Epoch 5/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.6837 - loss: 0.5523 - val_accuracy: 0.7056 - val_loss: 0.5260
Epoch 6/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.7187 - loss: 0.5437 - val_accuracy: 0.7500 - val_loss: 0.5433
Epoch 7/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.7027 - loss: 0.5269 - val_accuracy: 0.7812 - val_loss: 0.5198
Epoch 8/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.6835 - loss: 0.5626 - val_accuracy: 0.7969 - val_loss: 0.4874
Epoch 9/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.7135 - loss: 0.5486 - val_accuracy: 0.6875 - val_loss: 0.5259
Epoch 10/10


AHC_Expt_4.ipynb
# Plot accuracy and loss over time
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(10)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
Training and Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss


Training and Validation Accuracy
Training and Validation Loss
Training Accuracy
Validation Accuracy
Training Loss
Validation Loss

```python
from sklearn.preprocessing import LabelEncoder

# Step 2: Identify non-numeric columns and encode them
# Select only numeric columns and convert categorical columns into numeric values
for column in data.columns:
    if data[column].dtype == 'object':  # If the column is categorical
        print(f"Converting {column} column")
        le = LabelEncoder()  # Initialize LabelEncoder
        data[column] = le.fit_transform(data[column].astype(str))  # Convert to numerical values
```

```
Converting Patient ID column
Converting Sex column
Converting Blood Pressure column
Converting Diet column
Converting Country column
Converting Continent column
Converting Hemisphere column
```

```python
# Step 3: Separate features and target
X = data.iloc[:, :-1].values  # Features
y = data.iloc[:, -1].values  # Target (e.g., heart attack risk)
```

```python
# Step 4: Normalize the features
scaler = MinMaxScaler()
```

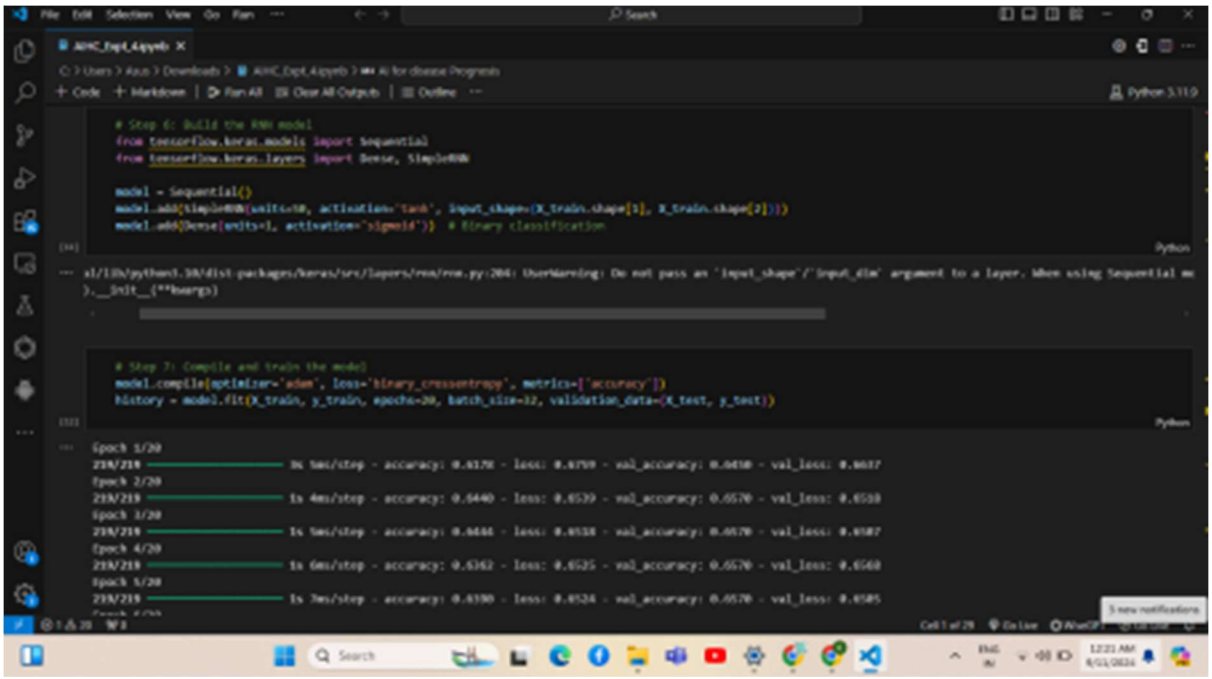
```python
# Step 4: Normalize the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

```python
# Reshape the data for RNN
timesteps = 5  # You can adjust this based on your data
X_rnn = []
y_rnn = []
```

```python
# Create sequences (for RNN input)
for i in range(timesteps, len(X_scaled)):
    X_rnn.append(X_scaled[i-timesteps:i, :])  # Create sequences of 'timesteps' length
    y_rnn.append(y[i])

X_rnn = np.array(X_rnn)
y_rnn = np.array(y_rnn)
```

```python
# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_rnn, y_rnn, test_size=0.2, random_state=42)
```



Google Collaboratory Link: -
https://colab.research.google.com/drive/1ulgCXcFxMf4hXX7PIZphH61qYi2gRtWH?usp=sharing

Conclusion: -
Comment on how useful it is to use AI for prognosis of a disease

Using AI for disease prognosis is highly beneficial due to its ability to improve diagnostic accuracy, offer early detection, and support personalized treatment plans. The application of AI models like CNNs and RNNs helps streamline healthcare, making it more data-driven, cost-effective, and proactive. While challenges like data privacy and model explainability exist, the advantages of AI in medical prognosis outweigh these issues, promising better patient outcomes and more efficient healthcare systems.