Create a simple chatbot using Python and natural language processing libraries to engage in conversation with users. (pip install nltk)

a. Import all necessary libraries
b. Generate some patterns/prompts
c. Generate some Response
d. Make conversion
e. End conversion

```
# run code on command prompt
# use following command:
# python chatbot.py
# type qouestion will get answer automatically
# question : what is your name?
# Answer:My name is Chatty!"

# to exit from command prompt type ctrl+C

import nltk
import re
import random

# Download necessary Nltk resources
nltk.download('punkt')

# Define greetings and responses
reflections = {
   "what is your name?": "My name is Chatty!",
   "how are you?": "I'm doing well, thanks for asking! How about you?",
}

# Define patterns as a dictionary
pairs = [
   (r"my name is (.*)", ["Hello %1, nice to meet you!"]),
   (r"what is your purpose?", ["I am a simple chatbot to practice natural language
processing. How can I help you today?"]),
   ("quit", ["Bye for now! See you next time."]),
   ("bye", ["Bye for now! See you next time."]),
]

def chat():
   print("Hello! I'm Chatty. How can I help you?")
   while True:
      user_input = input("> ")
      if user_input.lower() in reflections:
```

```
            print(reflections[user_input.lower()])
        else:
            matched = False
            for pattern, response in pairs:
                if re.match(pattern, user_input.lower()):
                    print(random.choice(response))
                    matched = True
                    # Check for stop word "bye" after a match
                    if user_input.lower() == "bye":
                        print("Chatbot stopping...")
                        break  # Exit the loop
            if not matched:
                print("Sorry, I don't understand. Try asking something else.")

if __name__ == "__main__":
    chat()
```

Build a single layer Perceptron,Train the network on a simple dataset, such as the XOR problem, and analyse the model's performance.

a. Import all necessary libraries
b. Define Dataset x= ([[0, 0], [0, 1], [1, 0], [1, 1]]) y= ([[0], [1], [1], [0]])
c. Preprocessing Dataset and Splitting Dataset into train and test
d. Build a Model and Compile model
e. Display model summary and accuracy.
f. Make Predictions

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split

X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

y = np.array([[0],
              [1],
              [1],
              [0]])

model = Sequential()
```

```
model.add(Dense(4, input_dim=2, activation='sigmoid'))  # Hidden
    layer with 4 neurons
model.add(Dense(1, activation='sigmoid'))  # Output layer with 1 output

model.compile(optimizer=Adam(),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X, y, epochs=1000, verbose=2)  # Training for 1000 epochs

loss, accuracy = model.evaluate(X, y)
print(f"Model Loss: {loss:.4f}")
print(f"Model Accuracy: {accuracy * 100:.2f}%")

predictions = model.predict(X)
print("\nPredictions:")
for i, prediction in enumerate(predictions):
   print(f"Input: {X[i]} -> Predicted Output: {prediction[0]:.4f}, Expected
    Output: {y[i][0]}")
```

Build a Multi Layer Perceptron , Train the network on a simple dataset,
Such as the XOR problem, and analyse the model's performance.

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset and Splitting Dataset into train and test
d. Build a Model and Compile model
e. Display model summary and accuracy.
f. Make Predictions

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input
Y = np.array([[0], [1], [1], [0]])              # Output

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=42)

model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))  # Hidden layer with 4 neurons
model.add(Dense(1, activation='sigmoid'))           # Output layer
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()

model.fit(X_train, Y_train, epochs=1000, verbose=0)

loss, accuracy = model.evaluate(X_test, Y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

predictions = model.predict(X_test)
predictions = (predictions > 0.5).astype(int)  # Converting probabilities to binary
outputs

for i, (pred, actual) in enumerate(zip(predictions, Y_test)):
   print(f"Sample {i+1}: Predicted = {pred[0]}, Actual = {actual[0]}")
```

Build a simple LSTM model using TensorFlow and the IMDB Movie Reviews dataset.

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset
d. Splitting Dataset into train and test
e. Build a Model and Compile model
f. Display model summary and accuracy.

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

num_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)

max_length = 500
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)

model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=32,
input_length=max_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
model.summary()

model.fit(X_train, y_train, epochs=3, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Implement Sentiment analysis using LSTM model and IMDB Movie Reviews dataset. Predict whether sentiment is positive or negative based on given text review.

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset and Splitting Dataset into train and test
d. Build a Model and Compile model
e. Display model summary and accuracy.
f. Make Predictions

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

num_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)

max_length = 500
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)

model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=64,
input_length=max_length))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))                    # LSTM
layer with dropout
model.add(Dense(1, activation='sigmoid'))                                   # Output layer

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()

model.fit(X_train, y_train, epochs=3, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
```

```
print(f"Test Accuracy: {accuracy * 100:.2f}%")

def predict_sentiment(text):

    word_index = imdb.get_word_index()

    words = text.lower().split()
    sequence = [word_index.get(word, 2) for word in words]  # 2 is the index for
unknown words
    padded_sequence = pad_sequences([sequence], maxlen=max_length)

    prediction = model.predict(padded_sequence)[0][0]
    sentiment = "Positive" if prediction > 0.5 else "Negative"
    print(f"Review: {text}\nPredicted Sentiment: {sentiment} ({prediction * 100:.2f}%
confidence)")

predict_sentiment("The movie was fantastic and had an amazing storyline.")
predict_sentiment("It was the worst film I have ever seen. Terrible experience.")
```

Build a simple CNN model using MNIST dataset and display model summary and test model Accuracy.

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset
d. Splitting Dataset into train and test
e. Build a Model and Compile model
f. Display model summary and accuracy.

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical

 (X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))  # Output layer for 10 classes

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

model.summary()

model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Build a CNN model for handwritten digit recognition using MNIST dataset and predict the digit based on given image..

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset and Splitting Dataset into train and test
d. Build a Model and Compile model
e. Display model summary and accuracy.
f. Make Predictions

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

 (X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))  # Output layer for 10 classes

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

model.summary()

model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

def predict_digit(image):
      image = image.reshape(1, 28, 28, 1).astype('float32') / 255.0

   prediction = model.predict(image)
   predicted_digit = np.argmax(prediction)


   plt.imshow(image.reshape(28, 28), cmap='gray')
   plt.title(f"Predicted Digit: {predicted_digit}")
   plt.show()

predict_digit(X_test[0])
```

Build a CNN model for fashion Dress recognition using Fashion MNIST dataset and predict theclass based on given image.

class_names = [‘T-shirt/top’, ‘Trouser’, ‘Pullover’, ‘Dress’, ‘Coat’, ‘Sandal’, ‘Shirt’, ‘Sneaker’, ‘Bag’,‘Ankle boot’].

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset and Splitting Dataset into train and test
d. Build a Model and Compile model
e. Display model summary and accuracy.
f. Make Predictions

```
import tensorflow as tf
```

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import numpy as np
import matplotlib.pyplot as plt

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

 (X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])

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

model.summary()

model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

def predict_class(image):
    image = image.reshape(1, 28, 28, 1).astype('float32') / 255.0

    prediction = model.predict(image)
    predicted_class = np.argmax(prediction)

    plt.imshow(image.reshape(28, 28), cmap='gray')
    plt.title(f"Predicted Class: {class_names[predicted_class]}")
    plt.show()

predict_class(X_test[0])
```

Build a simple CNN model using CIFAR -10 dataset and display model summary and test model Accuracy.

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing
d. Dataset
e. Splitting Dataset into train and test Build
f. a Model and Compile model Display model summary and accuracy.

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import numpy as np

class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
           'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

 (X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential([
   Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
   MaxPooling2D((2, 2)),
   Conv2D(64, (3, 3), activation='relu'),
   MaxPooling2D((2, 2)),
   Flatten(),
   Dense(128, activation='relu'),
```

```
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])

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

model.summary()

model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Implement a deep learning model CNN to classify images from the CIFAR-10 dataset and predict the class based on given image.

class_names = [‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’, ‘dog’, ‘frog’, ‘horse’, ‘ship’, ‘truck’]

a. Import all necessary libraries
b. Loading Dataset
c. Preprocessing Dataset and Splitting Dataset into train and test
d. Build a Model and Compile model
e. Display model summary and accuracy.
f. Make Predictions

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']

 (X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])

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

model.summary()

model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

def predict_image(image):
    image = image.reshape(1, 32, 32, 3).astype('float32') / 255.0

    prediction = model.predict(image)
    predicted_class = np.argmax(prediction)

    plt.imshow(image.reshape(32, 32, 3))
    plt.title(f"Predicted Class: {class_names[predicted_class]}")
    plt.show()

predict_image(X_test[0])
```

Build a Single Layer Neural Network Model and find Net input of Network using different input and weight vectors including bias value.[Use bias =0.45]

Ans: 0.94

a. Import all necessary libraries
b. Build a input vector & Accept values from user. X = [0.2 0.6 0.1]^T
c. Build a weight vector and Accept values from user .W=[0.3 0.7 0.1]^T
d. Build a Model and Compile model.

e. Calculate Net input of Network
f. Print result.

```
import numpy as np

X = np.array([float(input("Enter input X1: ")),
              float(input("Enter input X2: ")),
              float(input("Enter input X3: "))])

W = np.array([float(input("Enter weight W1: ")),
              float(input("Enter weight W2: ")),
              float(input("Enter weight W3: "))])

bias = 0.45

# e. Calculate Net Input of Network
# Net input = (X • W) + bias
net_input = np.dot(X, W) + bias

print(f"Net Input of the Network: {net_input}")
```

Build a Single Layer Neural Network Model and find output of different input and weight variables including bias value using Activation function.

[Use data: bias=0.53 ,binary=1/(1+np.exp(-y)) , bipolar = (np.exp(y)-1)/(np.exp(y)+1)]

Ans: 0.53 ,0.629 & 0.259

a. Import all necessary libraries
b. Build a input vector & Accept values from user. X=[0.8, 0.6 , 0.4]
c. Build a weight vector and Accept values from user.W= [0.1, 0.3 -0.2]
d. Build a Model and Compile model.
e. Calculate Net output of Network
f. Print result.

```
import numpy as np

X = np.array([float(input("Enter input X1: ")),
              float(input("Enter input X2: ")),
              float(input("Enter input X3: "))])

W = np.array([float(input("Enter weight W1: ")),
              float(input("Enter weight W2: ")),
              float(input("Enter weight W3: "))])

bias = 0.53
```

```
def binary_activation(y):
   return 1 / (1 + np.exp(-y))

def bipolar_activation(y):
   return (np.exp(y) - 1) / (np.exp(y) + 1)

net_input = np.dot(X, W) + bias

binary_output = binary_activation(net_input)
bipolar_output = bipolar_activation(net_input)

print(f"Net Input of the Network: {net_input}")
print(f"Binary Activation Output: {binary_output}")
print(f"Bipolar Activation Output: {bipolar_output}")
```

Build a Single Layer Neural Network Model and find net input of different input and weight vectors.

W=[0.2 0.1 -0.3]^T & X = [0.3 0.5 0.6]^T Ans:-0.07

a. Import all necessary libraries
b. Build a input vector & Accept values from user.
c. Build a weight vector and Accept values from user
d. Build a Model and Compile model.
e. Calculate Net input of Network

Print result

```
import numpy as np

X = np.array([float(input("Enter input X1: ")),
        float(input("Enter input X2: ")),
        float(input("Enter input X3: "))])

W = np.array([float(input("Enter weight W1: ")),
        float(input("Enter weight W2: ")),
        float(input("Enter weight W3: "))])

net_input = np.dot(X, W)

print(f"Net Input of the Network: {net_input}")
```

Build a simple Autoencoder model for Image Compression using given image and display model summary and test model Accuracy.

a. Import all necessary libraries
b. Loading & Pre-processing Image

c. Define the autoencoder architecture (Encoder &Decoder Layer)
d. Define model &Compile model
e. Train the autoencoder & Create the autoencoder model
f. Display the compressed image/Save image.

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.datasets import mnist
from tensorflow.keras.optimizers import Adam

 (X_train, _), (X_test, _) = mnist.load_data()
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

X_train = X_train.reshape((len(X_train), -1))
X_test = X_test.reshape((len(X_test), -1))

input_dim = X_train.shape[1]
encoding_dim = 64  # Compression size

input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_img)


decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')

autoencoder.fit(X_train, X_train, epochs=20, batch_size=256, shuffle=True,
validation_data=(X_test, X_test))

encoded_imgs = Model(input_img, encoded).predict(X_test)
decoded_imgs = autoencoder.predict(X_test)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis('off')

        ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
```