Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)
University with Graded Autonomy Status

RECORD NOTEBOOK

ARTIFICIAL INTELLIGENCE (BCS19I07)

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING/DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

NAME :

REGISTER NO :

COURSE : B.TECH CSE(DS&AI)

YEAR/SEM/SEC :

2024-2025(ODD SEMESTER)

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)
University with Graded Autonomy Status

BONAFIDE CERTIFICATE

REGISTER NO:

NAME OF LAB : ARTIFICIAL INTELLIGENCE (BCS19I07

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING/DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

Certified that this is the bonafide record of work done by of II Year B.Tech-CSE(DS&AI), Sec- in the **ARTIFICIAL INTELLIGENCE (BCS19I07)** during the year 2024-2025.

Signature of Lab-in-Charge **Signature of Head of Dept**

Submitted for the Practical Examination held on _____

Internal Examiner **External Examiner**

INDEX

Ex.No	TELECOMMUNICATION CHRUN PREDICTION USING RANDOM FOREST	Date
1		

AIM:

To predict customer churn in a telecommunications dataset using the Random Forest algorithm and evaluate the model's performance.

ALGORITHM:

STEP 1: Import essential libraries such as pandas, numpy, scikit-learn, and matplotlib

STEP 2: Load the dataset using pandas.

STEP 3: Handle missing values by filling or dropping them.

STEP 4: Encode categorical features using techniques like one-hot encoding.

STEP 5: Split the dataset into features (X) and the target variable (y).

STEP 6: Split the Data into Training and Testing Sets:

STEP 7: Initialize the RandomForestClassifier from scikit-learn.

STEP 8: Train the model using the training data.

STEP 9: Use the trained model to make predictions on the test data.

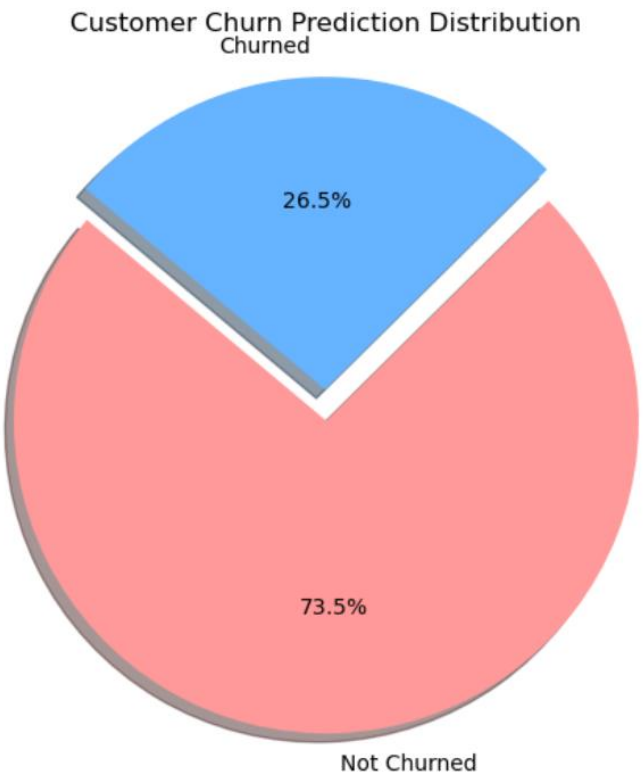
PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
employee = pd.read_csv(r"M:\AI\Data set\Telco-Customer-Churn.csv")
employee.head()
employee = employee.drop(['customerID'],axis=1)
employee.columns
employee['TotalCharges'] =employee["TotalCharges"].replace(" ",np.nan).astype(float)
employee.isna().sum()
employee.TotalCharges.fillna(employee.TotalCharges.mean(),inplace=True)
employee.isna().sum()
employee.head()
employee['Churn'] = employee['Churn'].map({'No': 0, 'Yes': 1})
employee_encoded = pd.get_dummies(employee, drop_first=True)

employee_encoded.head()
X = employee_encoded.drop('Churn', axis=1)
y = employee_encoded['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)
Model= RandomForestClassifier(n_estimators=100, random_state=42)
Model.fit(X_train, y_train)
y_pred = Model.predict(X_test)
results_df = pd.DataFrame({
   'Actual value': y_test,
   'Predicted value': y_pred
})
results_df

conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

labels = ['Not Churned', 'Churned']
sizes = [conf_matrix[0, 0] + conf_matrix[0, 1], conf_matrix[1, 0] + conf_matrix[1, 1]]
colors = ['#ff9999','#66b3ff']
explode = (0.1, 0)
plt.figure(figsize=(8, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=True, startangle=140)
plt.title('Customer Churn Prediction Distribution')
plt.axis('equal')
plt.show()
```

OUTPUT:

	Actual value	Predicted value
185	1	1
2715	0	0
3825	0	0
1807	1	1
132	0	0
...	...	...
6366	0	0
315	0	0
2439	0	0
5002	0	0
1161	1	0

1409 rows × 2 columns



RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	SUPPORT VECTOR MACHINE TO CLASSIFY EMAILS	Date
2		

AIM

To write a Python program to classify emails as spam or not spam using a Support Vector Machine (SVM).

ALGORITHM

STEP 1: Collect a dataset containing emails labeled as spam or not spam.

STEP 2: Load the dataset. Clean the email text data (remove HTML tags, punctuation, etc.).

STEP 3: Convert the text data into numerical features using techniques.

STEP 4: Split the dataset into training and testing sets.

STEP 5: Train the SVM model using the training data.

STEP 6: Test the model using the testing data.

STEP 7: Evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

STEP 8: Use the trained model to classify new emails as spam or not spam.

STEP 9: Visualize the classification results using suitable charts or graphs.

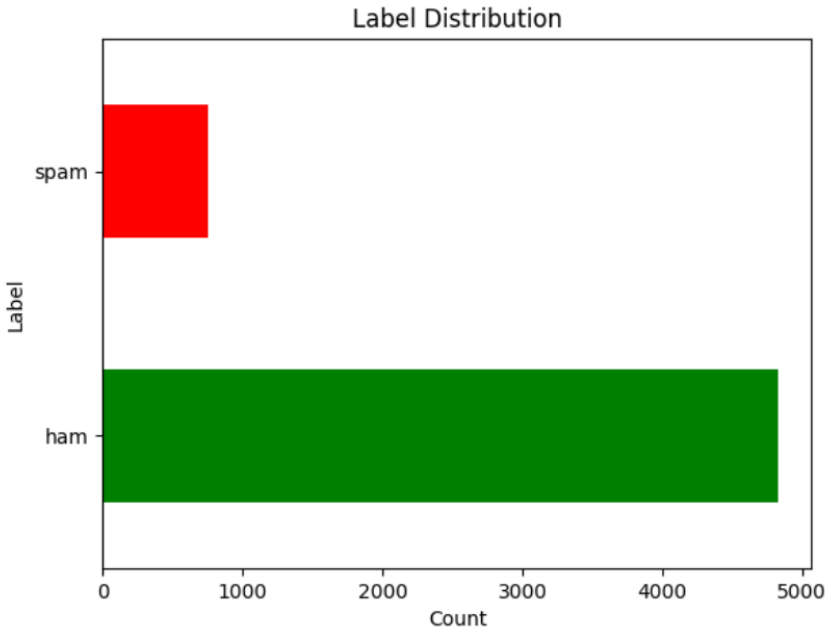
PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data = pd.read_csv(r'M:\AI\Data set\spam.csv')
data.head()
data.info()
data.describe()
X = data['EmailText'].values
y = data['Label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
cv = CountVectorizer()
X_train = cv.fit_transform(X_train)
X_test = cv.transform(X_test)
classifier = SVC(kernel='rbf', random_state=10)  # rbf -> Radial Basis Function
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_df
print("Model Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
n = pd.value_counts(data["Label"], sort=True)
n.plot(kind='barh', color=["green", "red"])
plt.title('Label Distribution')
plt.xlabel('Count')
plt.ylabel('Label')
plt.show()
```

OUTPUT:

	Actual	Predicted
0	ham	ham
1	ham	ham
2	ham	ham
3	ham	ham
4	ham	ham
...	...	...
2781	ham	ham
2782	ham	ham
2783	spam	spam
2784	ham	ham
2785	ham	ham

2786 rows × 2 columns



RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	CLUSTERING COUNTRIES BASED ON GEOGRAPHICAL COORDINATES.	Date
3		

AIM

To group countries into clusters based on their longitude and latitude for geographical segmentation

ALGORITHM

STEP 1: Load the dataset containing countries' names, longitude, and latitude from a CSV file.

STEP 2: Visualize the countries on a scatter plot using their geographical coordinates (longitude and latitude).

STEP 3: Select longitude and latitude columns for clustering.

STEP 4: Apply the K-Means algorithm to group countries into 3 clusters based on their geographical coordinates.

STEP 5: Select relevant features for clustering (e.g., total purchase amount, frequency of visits, age, location).

STEP 6: Determine the optimal number of clusters (k)

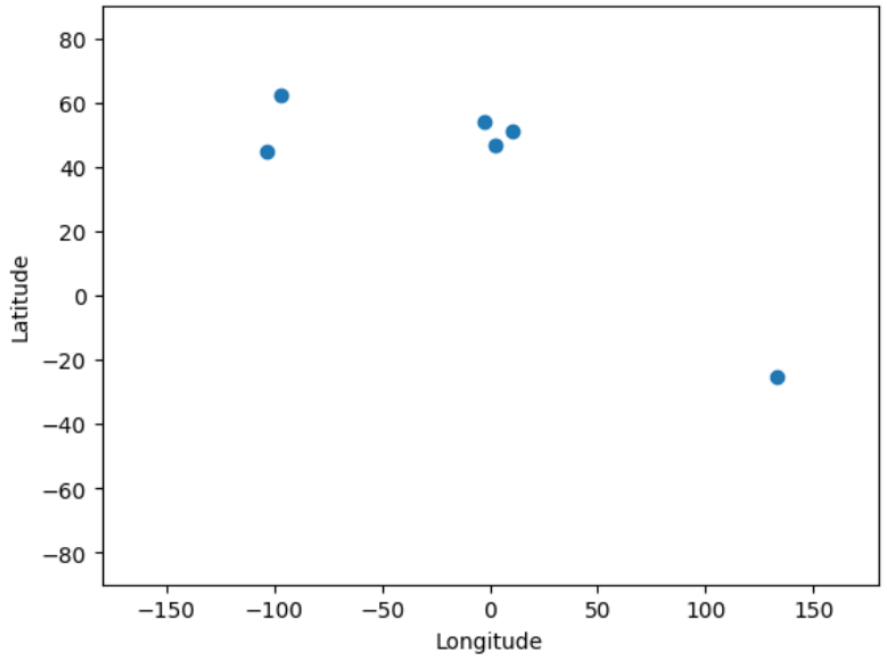
STEP 7: Identify the clusters and add the cluster labels to the original dataset.

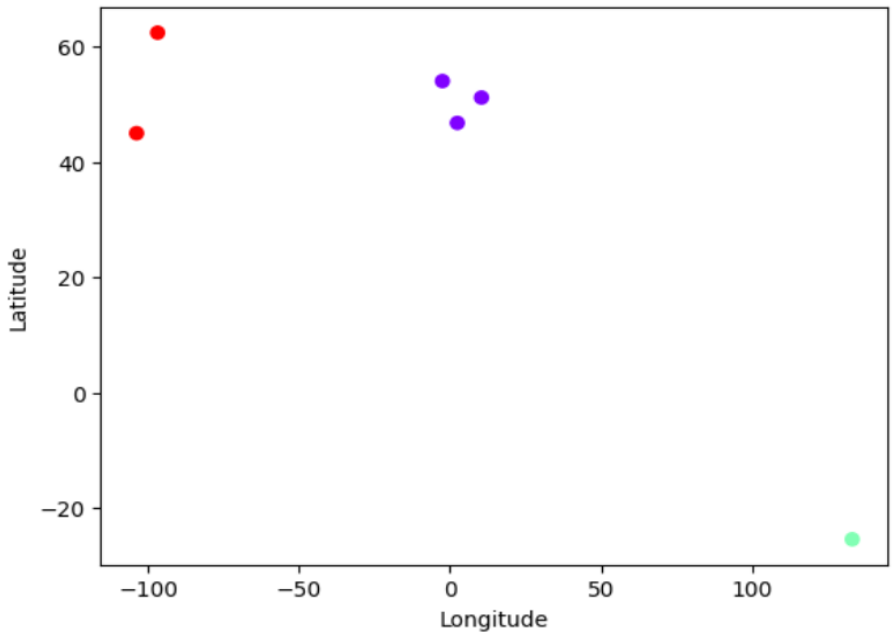
STEP 8: Visualize the clustered countries on a scatter plot with different colors representing different clusters

PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
data = pd.read_csv("D:\Technologies \ML Datasets\Country_clusters.csv")
plt.scatter(data['Longitude'], data['Latitude'])
plt.xlim(-180, 180)
plt.ylim(-90, 90)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
x = data.iloc[:, 1:3]
kmeans = KMeans(3)
kmeans.fit(x)
identified_clusters = kmeans.fit_predict(x)
data_with_clusters = data.copy()
data_with_clusters['Clusters'] = identified_clusters
plt.scatter(data_with_clusters['Longitude'], data_with_clusters['Latitude'],
c=data_with_clusters['Clusters'], cmap='rainbow')
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

OUTPUT:





RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	TEXT TO SPEECH USING IBM WATSON API KEYS	Date
4		

AIM

To write a python program to convert text into speech using the IBM Watson Text to Speech API.

ALGORITHM

STEP 1: Create an IBM Cloud account if you do not already have one.

STEP 2: Log in to your IBM Cloud account and create an instance of the IBM Watson Text to Speech service.

STEP 3: Obtain the API key and service URL from the credentials of the IBM Watson Text to Speech service instance.

STEP 4: Install the IBM Watson SDK for your programming language

STEP 5: Authenticate with the IBM Watson Text to Speech service using the API key and service URL.

STEP 6: Prepare the text you want to convert into speech.

STEP 7: Use the IBM Watson Text to Speech API to synthesize the text into speech.

STEP 8: Save the synthesized speech audio to a file (e.g., WAV or MP3 format).

STEP 9: Verify the audio file by playing it to ensure the text has been accurately converted to speech.

PROGRAM:

```
import requests
from requests.auth import HTTPBasicAuth
url ='https://api.au-syd.text-to-speech.watson.cloud.ibm.com/instances/94bc3937-1991-4aca-9cbf-ab3411375524/v1/synthesize'
api_key ='FZbxQ98T1MLU94hBXng9cC7u7KS3KuaHzLhpPe_PjbXP'
text = "How many times do I have to tell you the same thing? You never listen."
headers = {
    'Content-Type': 'application/json',
    'Accept': 'audio/mp3'  # Request MP3 format
}
data = {
    'text': text,
    'voice': 'en-US_AllisonV3Voice'
}
response = requests.post(
    url,
    headers=headers,
    json=data,  # Use json=data for JSON payload
    auth=HTTPBasicAuth('apikey', api_key)
)
if response.status_code != 200:
    print(f"Error: {response.status_code} - {response.text}")
else:
    with open('output.mp3', 'wb') as audio_file:
        audio_file.write(response.content)
    print("Audio content written to file 'output.mp3'")
```

OUTPUT:

```
Audio content written to file 'output.mp3'
```

RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	SPEECH TO TEXT USING IBM WATSON API KEYS	Date
5		

AIM

To write a python program to convert speech into text using the IBM Watson Speech to Text API.

ALGORITHM

STEP 1: Create an IBM Cloud account if you do not already have one.

STEP 2: Log in to your IBM Cloud account and create an instance of the IBM Watson Speech to Text service.

STEP 3: Obtain the API key and service URL from the credentials of the IBM Watson Speech to Text service instance.

STEP 4: Install the IBM Watson SDK for your programming language (e.g., Python) using the appropriate package manager.

STEP 5: Authenticate with the IBM Watson Speech to Text service using the API key and service URL.

STEP 6: Prepare the audio file you want to convert into text (e.g., WAV or MP3 format).

STEP 7: Use the IBM Watson Speech to Text API to transcribe the audio file into text.

STEP 8: Retrieve and review the transcribed text from the API response.

STEP 9: Optionally, save the transcribed text to a file for further use.

PROGRAM:

```
pip install ibm-watson
import json
from ibm_watson import SpeechToTextV1
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
apikey = "Nr4Lhzm28MShtRdIITCC5swS_U877tp0nBQEUiEjh7Ub"
url = "https://api.eu-gb.speech-to-text.watson.cloud.ibm.com/instances/65dce26f-30b0-4768-8f03-ef5f6110422c"
authenticator = IAMAuthenticator(apikey)
speech_to_text = SpeechToTextV1(authenticator=authenticator)
speech_to_text.set_service_url(url)
audio_file_path = "C:\\Users\\santh\\LAB program\\output.mp3"
try:
    with open(audio_file_path, 'rb') as audio_file:
        result = speech_to_text.recognize(
            audio=audio_file,
            content_type='audio/mp3',
            model='en-US_BroadbandModel'
        ).get_result()
    text = result['results'][0]['alternatives'][0]['transcript']
    print(f"Transcribed Text: {text}")

except Exception as e:
    print(f"An error occurred: {e}")
```

OUTPUT:

```
Transcribed Text: how many times do I have to tell you the same thing you never listen
```

RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	SENTIMENT ANALYSIS TECHNIQUES	Date
6		

AIM

To analyze and determine the sentiment (positive, negative, neutral) of a given text using sentiment analysis techniques.

ALGORITHM

STEP 1: Gather or load the text data you want to analyze. This could be from social media posts, reviews, surveys, etc.

STEP 2: Preprocess the text data by removing any irrelevant information

STEP 3: Choose or build a sentiment analysis model.

STEP 4: Apply the sentiment analysis model to the preprocessed text data to determine the sentiment.

STEP 5: Interpret the sentiment results (e.g., classify text as positive, negative, or neutral).

STEP 6: Optionally, visualize the sentiment analysis results using charts or graphs to summarize the overall sentiment distribution.

STEP 7: Use the sentiment insights for decision-making or further analysis, such as improving customer experience or monitoring brand reputation.

PROGRAM:

```
from transformers import pipeline
model_name = "distilbert-base-uncased-finetuned-sst-2-english"
sentiment_pipeline = pipeline('sentiment-analysis', model=model_name)

def analyze_sentiment(text):
    result = sentiment_pipeline(text)
    return result

sample_texts = [
    "I love this product! It's absolutely amazing.",
    "This is the worst experience I've ever had.",
    "It's okay, not too bad, but could be better."
]

for text in sample_texts:
    print(f"Text: {text}")
    print(f"Sentiment Analysis: {analyze_sentiment(text)}")
    print()
```

OUTPUT:

```
Text: I love this product! It's absolutely amazing.
Sentiment Analysis: [{'label': 'POSITIVE', 'score': 0.999885082244873}]

Text: This is the worst experience I've ever had.
Sentiment Analysis: [{'label': 'NEGATIVE', 'score': 0.9997679591178894}]

Text: It's okay, not too bad, but could be better.
Sentiment Analysis: [{'label': 'POSITIVE', 'score': 0.9789144992828369}]
```

RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	DEVELOPING CHATBOT USING IBM WATSON ASSISTANT	Date
7		

AIM

To create a chatbot for banking application and deploy a chatbot using IBM Watson Assistant.

ALGORITHM

STEP 1: Create an IBM Cloud account if you do not already have one.

STEP 2: Log in to your IBM Cloud account and create an instance of the IBM Watson Assistant service.

STEP 3: Obtain the API key and service URL from the credentials of the IBM Watson Assistant service instance.

STEP 4: Access the Watson Assistant dashboard and create a new workspace or skill for your chatbot.

STEP 5: Design the chatbot's conversational flow

STEP 6: Train the chatbot by providing example phrases for each intent and testing the chatbot's ability to recognize these intents accurately.

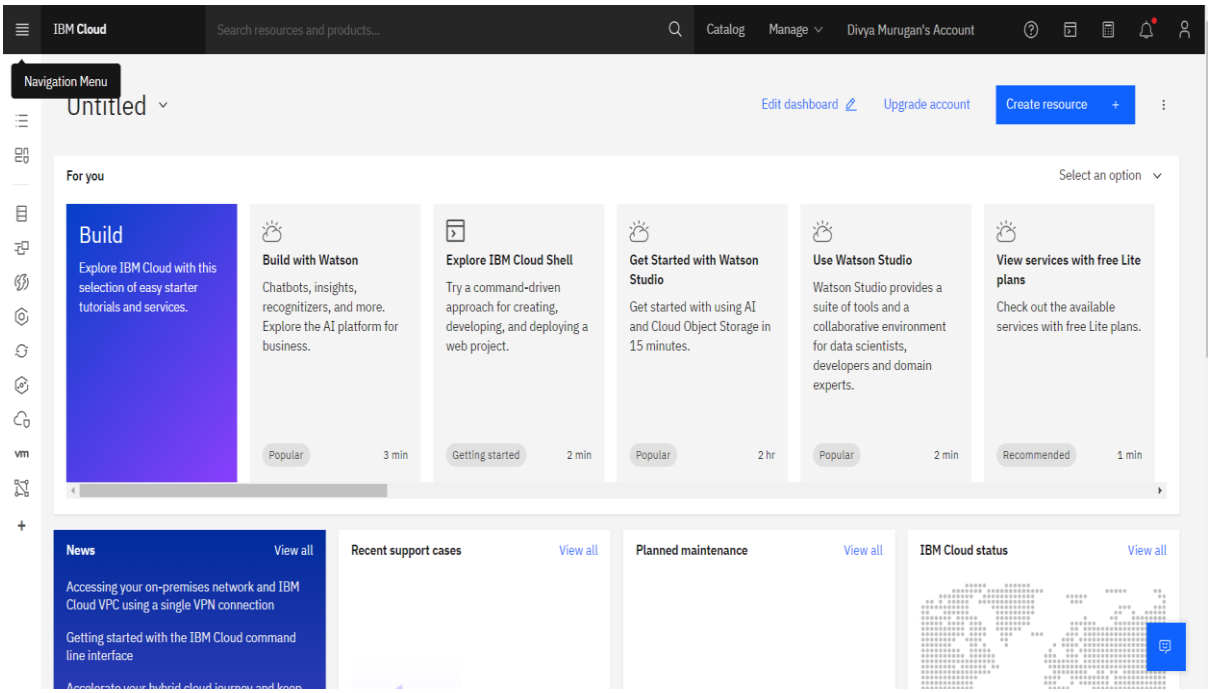
STEP 7: Integrate the chatbot with a messaging platform or application by using the appropriate APIs or SDKs provided by IBM Watson.

STEP 8: Test the chatbot in a real-world scenario to ensure it functions as expected, and refine its responses based on feedback and performance.

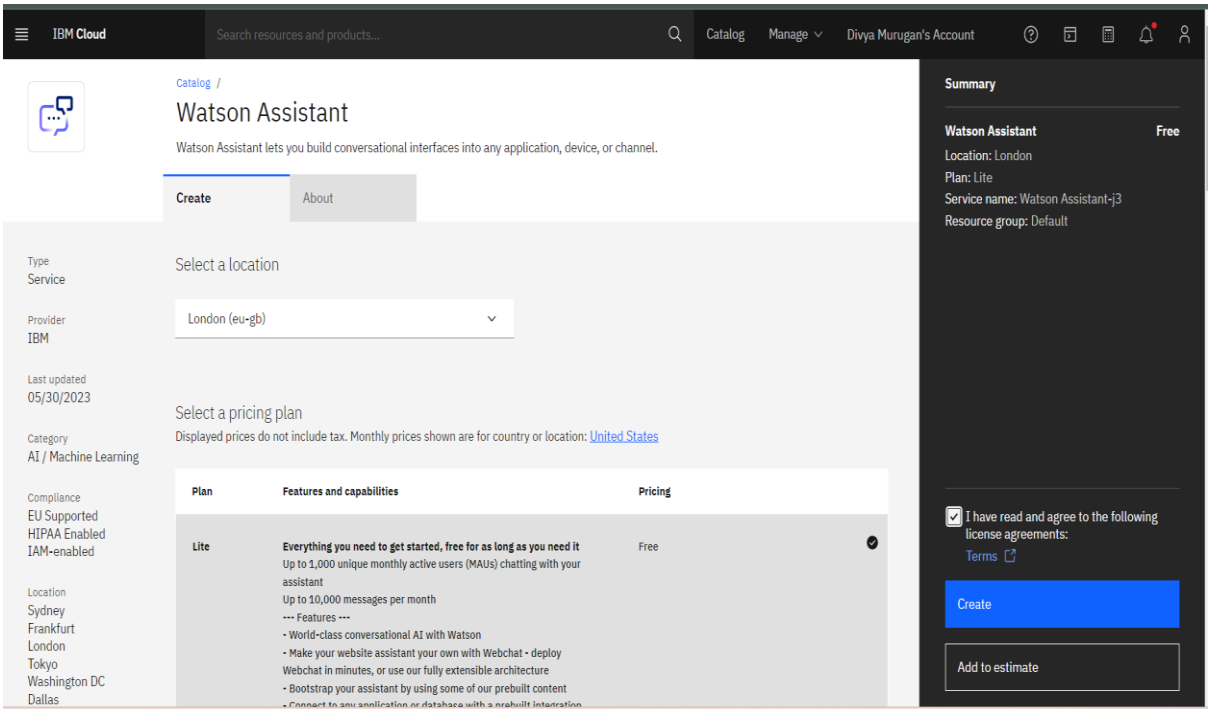
STEP 9: Deploy the chatbot and monitor its interactions to gather insights and make continuous improvements.

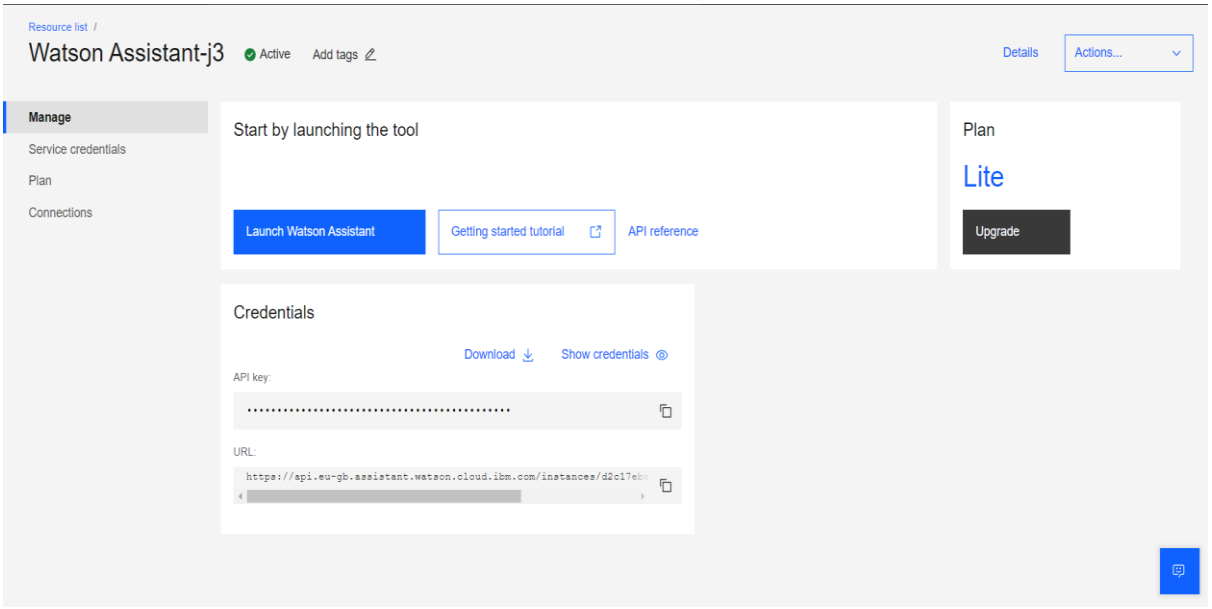
PROCEDURE TO IMPLEMENTATION:

STEP 1: Login to IBM Cloud using your credentials**.**

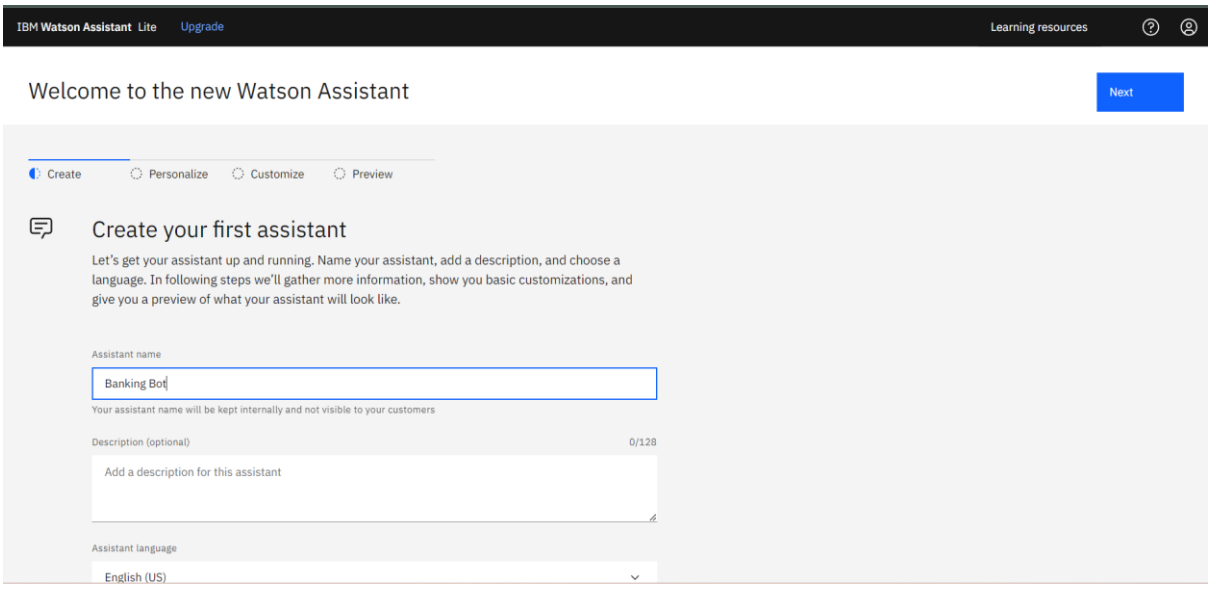


STEP 2: In the ***catalog*** search, type "***Watson Assistant***" and open the service. Afterward, click on "***Launch Watson Assistant***" to access the Watson Assistant platform.

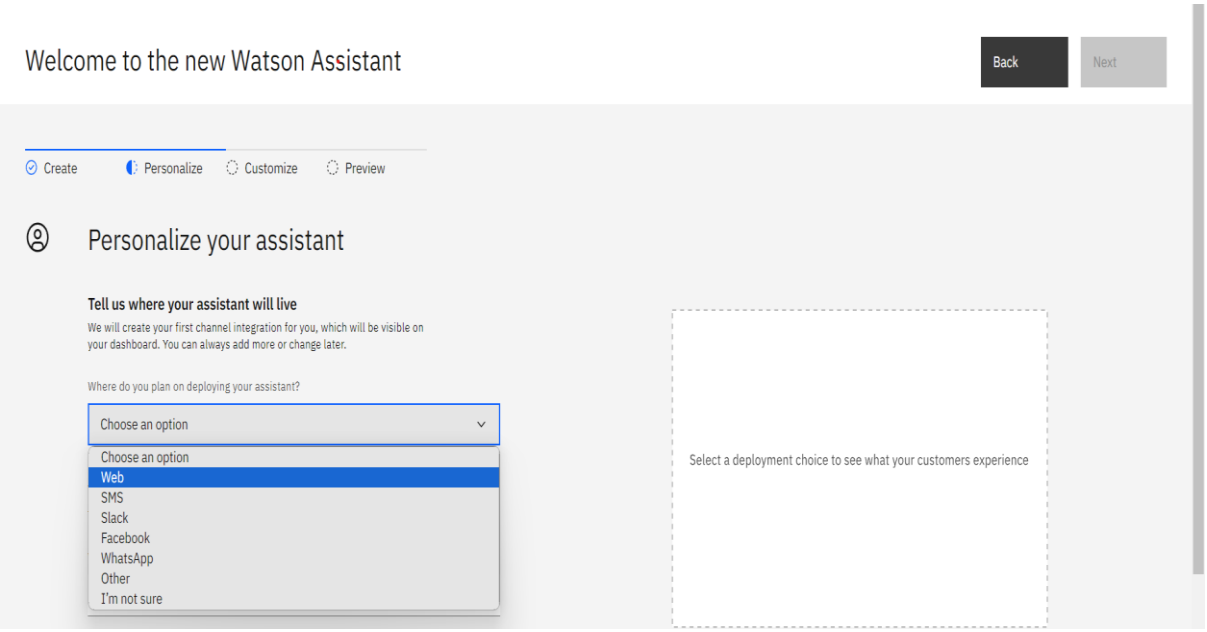




STEP 3: At the top of the page in a horizontal view, you'll find several options: *Create, Personalize, Customize, and Preview*. Within the "***Create"*** page, you can select your ***Assistant's name.***



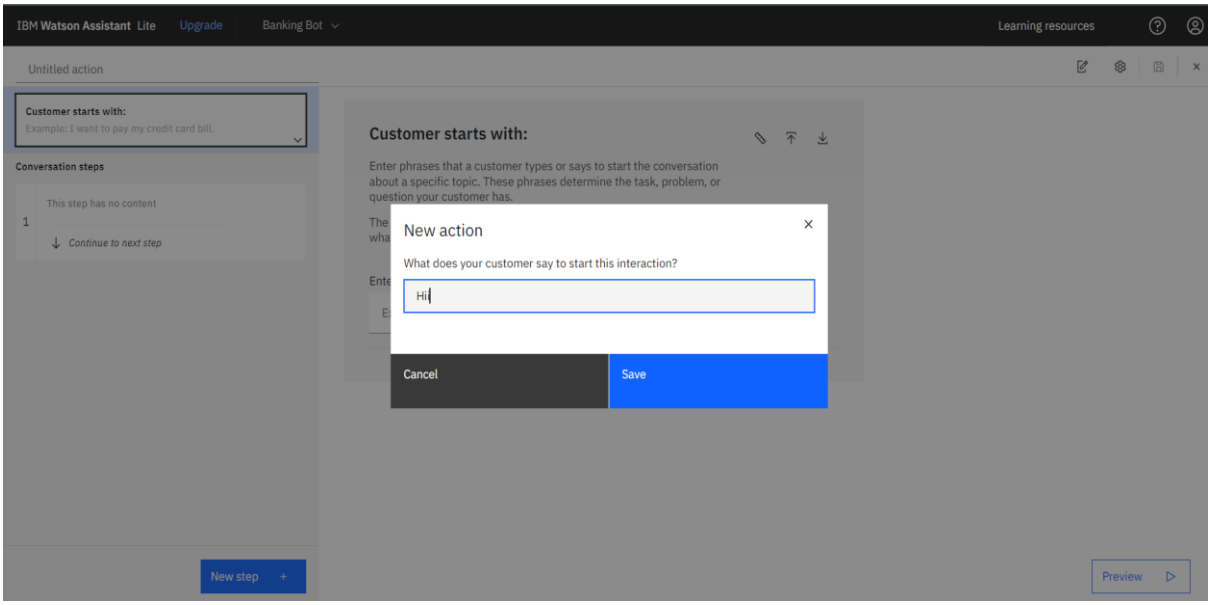
STEP 4: In the "***Personalize***" page, select "***Web***" for where you plan to deploy the assistant. For the ***"Choose Industry***" option, select ***"(N/A) I am a student***" Finally, in the "***Preview"*** section, click on "***Create."***



STEP 5: Click on ***"Create action***."

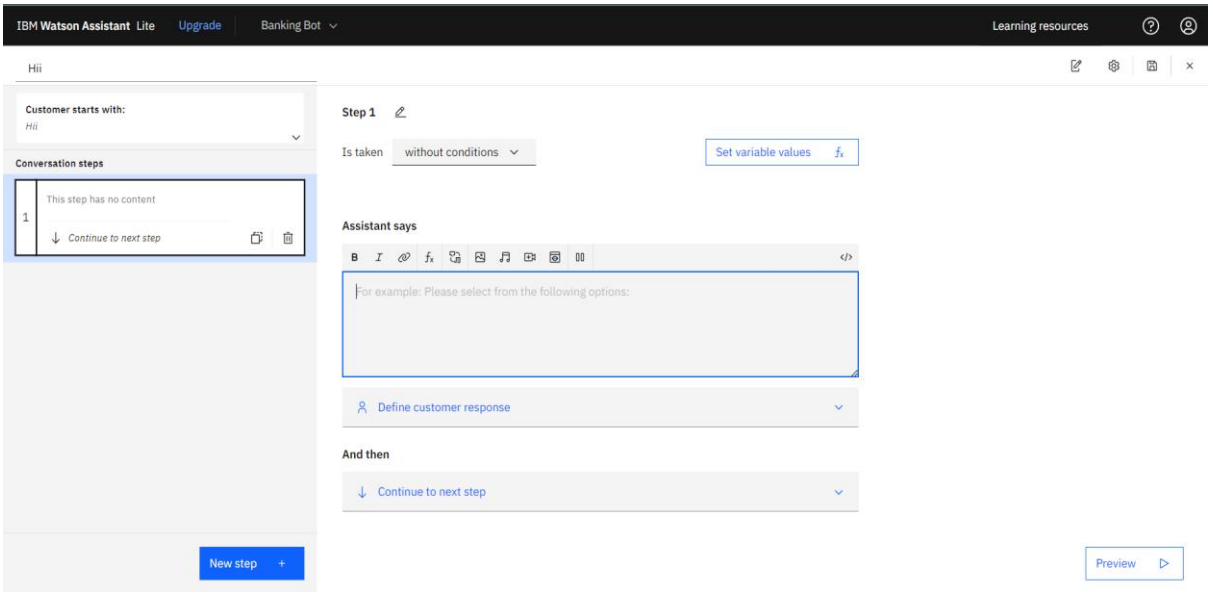
Choose "***Start from scratch."***

Under the ***new action***, begin by writing your first question to the chatbot. In this context, consider the "Customer" as the User and the "Conversation" as the assistant's reply.



STEP 6: Under the "***Assistant says***" section, provide your response to address the user's concern. Afterward, ***save your work***.

Proceed to the "***Preview***" section to verify that your question will indeed receive a corresponding answer from the chatbot.



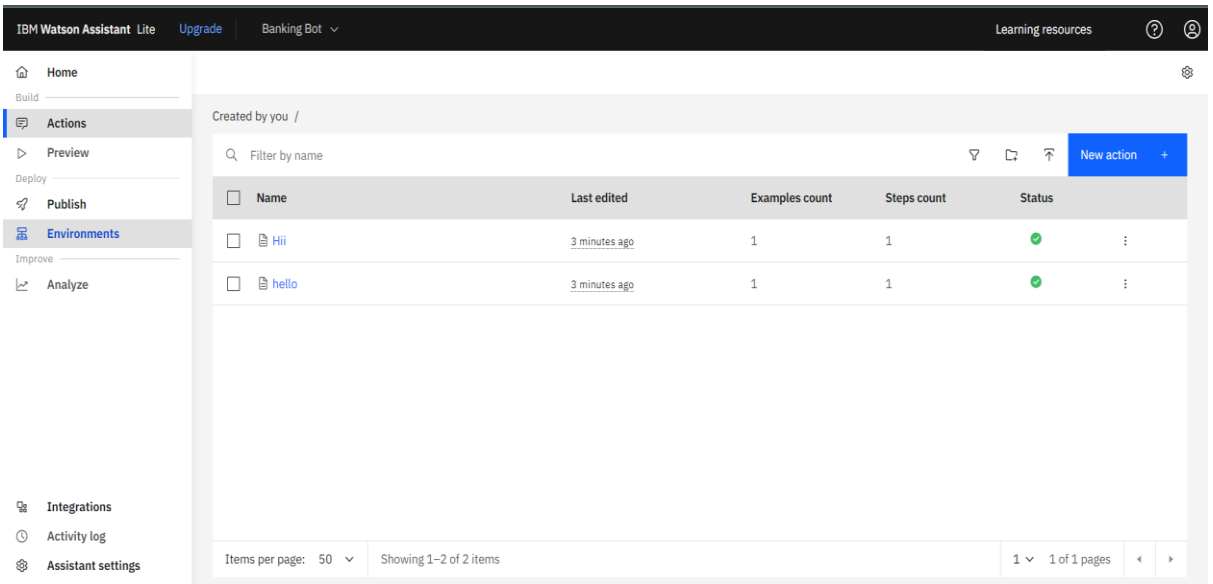
STEP 7: After completing and saving one action, proceed to "***create a new action"*** each time. Please note that if you wish to include an image in your response, click on "***image icon***" and paste the image URL.

STEP 8: To set up a loop for a specific question, utilize the ***"Define customer response***" feature, and then access the ***"Options***" section.

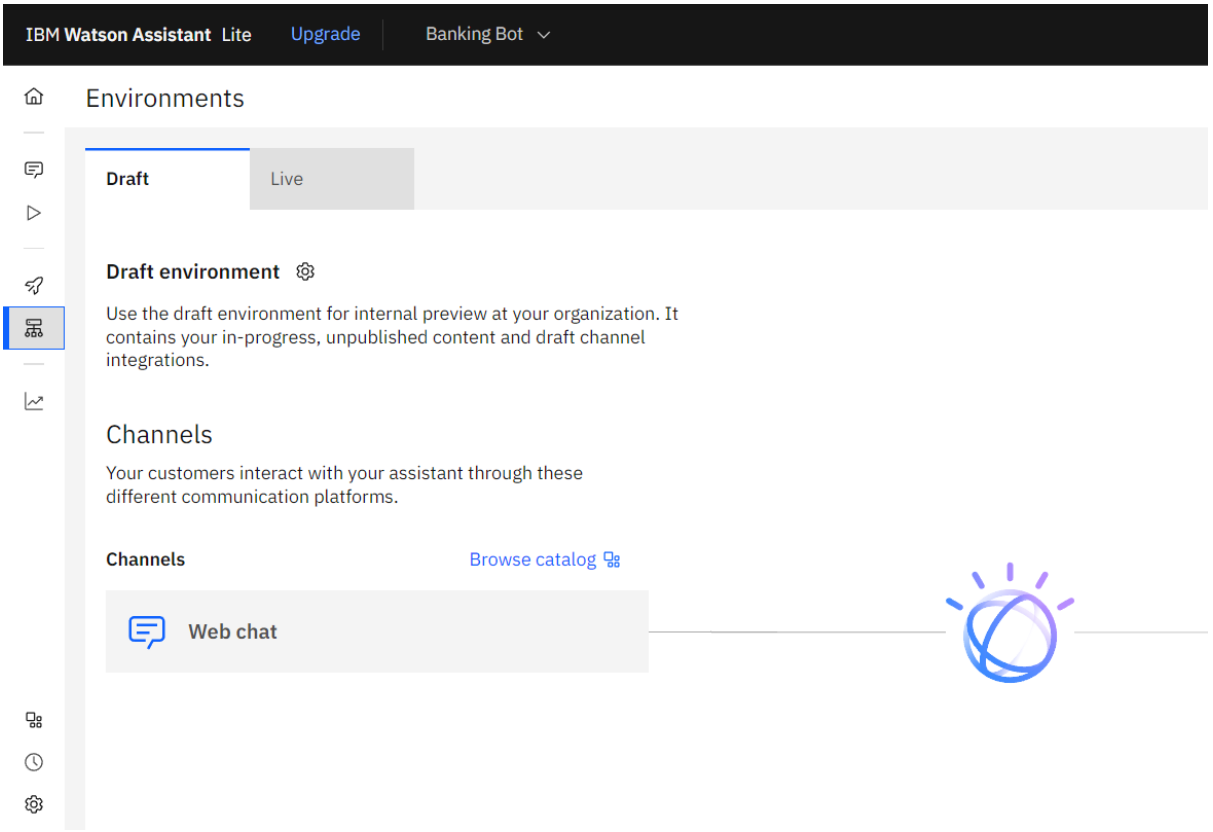
STEP 9: To progress through each action, remember to conclude the current conversation by selecting "***Continue to next step"*** and then opt for "***End the action***."

Step 10: Open ***Visual Studio Code*** and create a ***new HTML file***. Begin by writing a sample web application code.

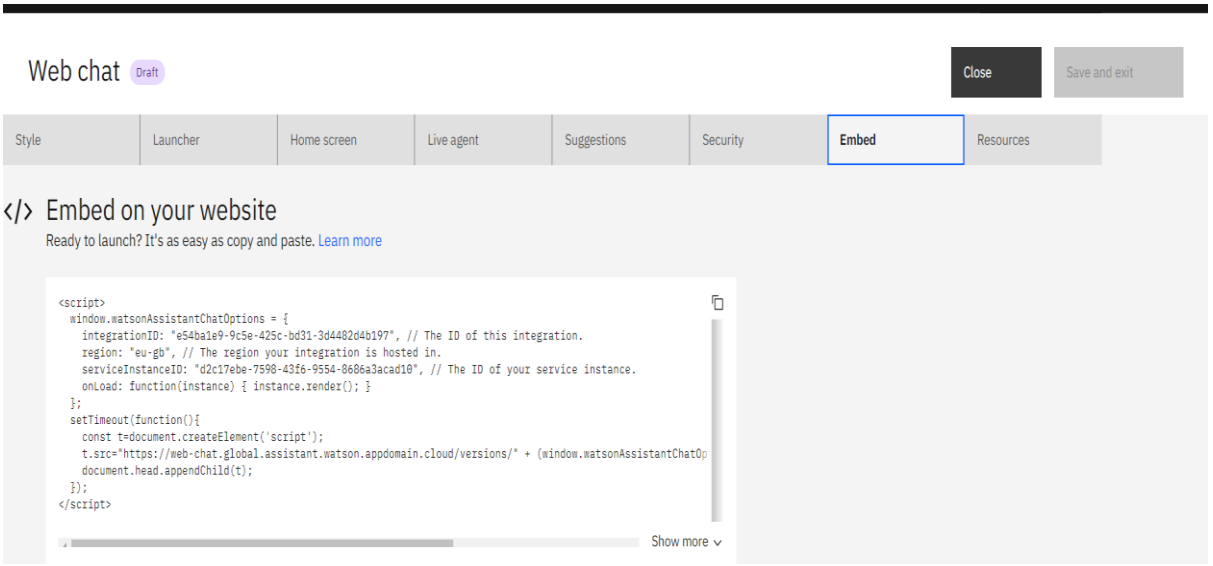
Step 11: To integrate this assistant into our sample web application, navigate to the "***Environment panel***" in Watson Assistant.



STEP 12: Within the "***Environments***" section, you'll find two options: "Draft" and "Live." Navigate to the "***Draft"*** page and click on "***Web chat***" to proceed.



STEP 13: In the "***Web Chat***" section, select ***"Embed***." Then, copy the provided ***script tag code*** and paste it into your HTML code within your Visual Studio web application.



STEP 14: Finally, when you run the web application, you should see IBM Watson Assistant successfully embedded on your web page.

RESULT:

Thus, the Creating a Chatbot using IBM Watson assistant and embed this chatbot in Web Application are implemented successfully

Ex.No	BUILD A NEURAL NETWORK MODEL TO ACCURATELY CLASSIFY THE DIGITS 0 TO 9	Date
8		

AIM:

To Build a Neural Network Model to accurately classify and differentiate between the digits 0 to 9. Once trained, the model can then predict the digit in new, unseen handwritten images.

ALGORITHM:

STEP 1: import TensorFlow and Keras for building and training the neural network.

STEP 2: Load the MNIST dataset, which contains a large number of handwritten digit images for training and testing.

STEP 3: Normalize the pixel values of the images to a range between 0 and 1.

STEP 4: Define a sequential neural network model with a flatten layer, a dense hidden layer, a dropout layer for regularization, and an output layer with softmax activation.

STEP 5: Train the model on the training data using model.fit.

STEP 6: Evaluate the model's accuracy on the test data.

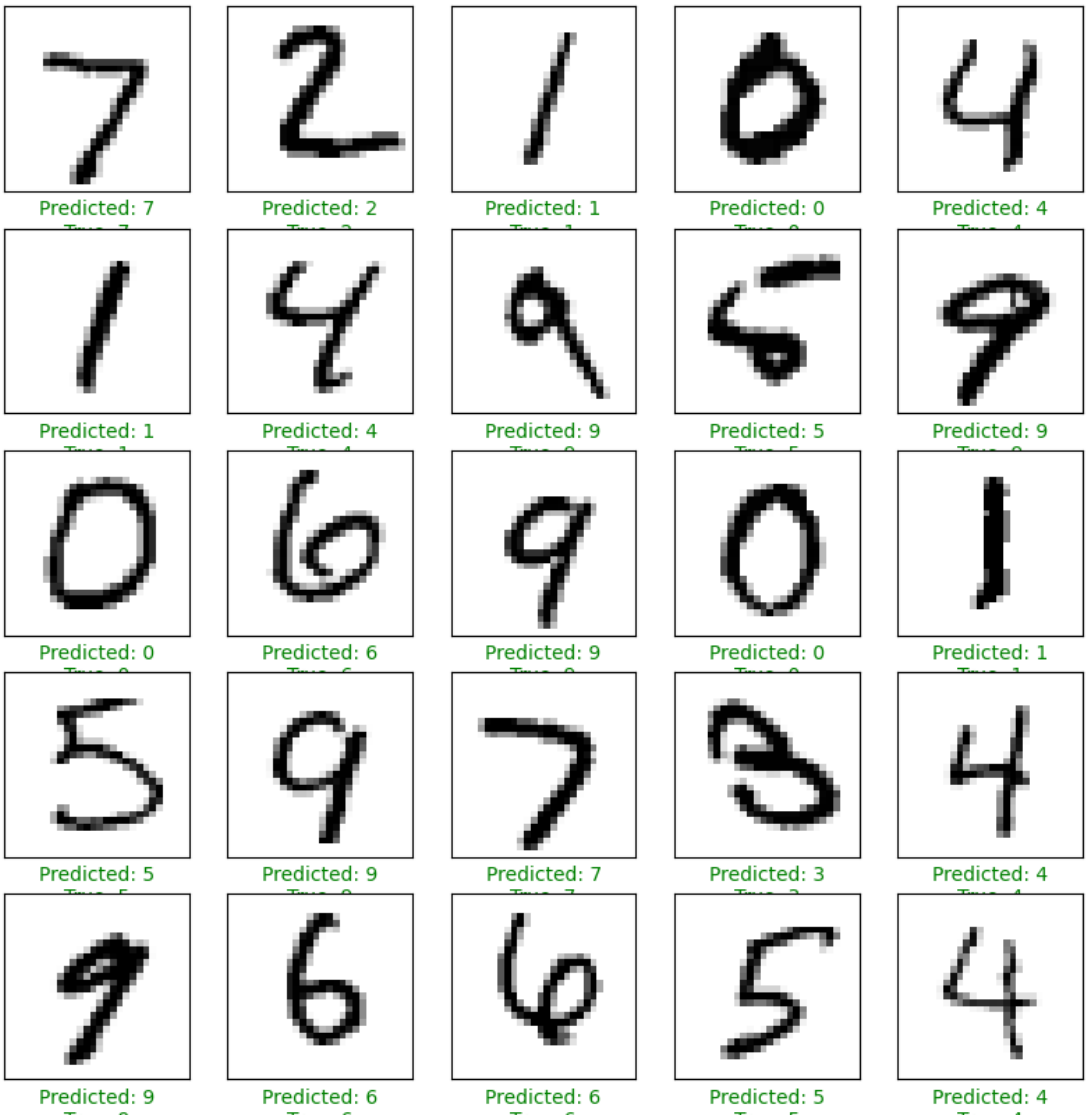
STEP 7: Make predictions on new, unseen images from the test set and visualize some of the predictions.

PROGRAM:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2),  # Dropout layer to reduce overfitting
    keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print("Test accuracy:", test_acc)
predictions = model.predict(test_images)
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    predicted_label = predictions[i].argmax()
    true_label = test_labels[i]
    if predicted_label == true_label:
        color = 'green'
```

```
    else:
        color = 'red'
    plt.xlabel(f"Predicted: {predicted_label}\nTrue: {true_label}", color=color)
plt.show()
```

OUTPUT:



RESULT:

Thus, the Python program to build a Neural Network Model to accurately classify and differentiate between the digits 0 to 9 are implemented successfully.

Ex.No	IMAGE PREPROCESSING USING OPENCV	Date
9		

AIM

To preprocess images for improved quality and feature extraction using OpenCV.

ALGORITHM

STEP 1: Install OpenCV. Ensure OpenCV library is installed in your environment.

STEP 2: Read the image file into the system.

STEP 3: Convert the image to grayscale or other necessary color spaces to simplify processing.

STEP 4: Adjust the image size to a specific width and height to standardize dimensions.

STEP 5: Smooth the image to reduce noise and blur unwanted details.

STEP 6: Detect edges in the image to highlight features and boundaries.

STEP 7: Segment the image into foreground and background using thresholding techniques.

STEP 8: Refine the image with operations such as dilation and erosion to enhance features.

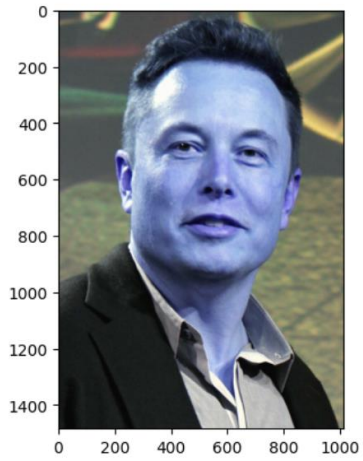
STEP 9: Save or display the pre-processed image for further analysis or use.

PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
pic=cv2.imread(r"D:\Technologies \Lab Excersizes\Elon Musk.jpg")
plt.imshow(pic)

pic=cv2.imread(r"D:\Technologies\Deep Learning Models & AI Analyst\Lab
Excersizes\Elon Musk.jpg",cv2.IMREAD_GRAYSCALE)
plt.imshow(pic)

flip_pic=np.flipud(pic)
plt.imshow(flip_pic,cmap='gray')

from scipy import ndimage
rot_pic=ndimage.rotate(pic,45)
plt.imshow(rot_pic,cmap='gray')

gauss_pic=ndimage.gaussian_filter(pic,5)
plt.imshow(gauss_pic,cmap='gray')
pic2=255-pic
cv2.imshow("Negative image",pic2)

pic.shape[0:2]
pixels = pic[100,100]
print(pixels)
print (pic. shape)
print (pic. size)
print("Mean:",pic.mean())
print("Max:",pic.max())
print("Min:",pic.min()
```

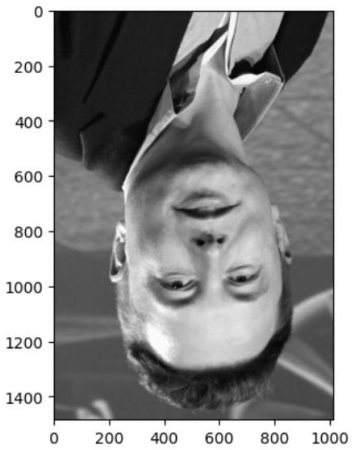
OUTPUT:

<matplotlib.image.AxesImage at 0x25f02024450>



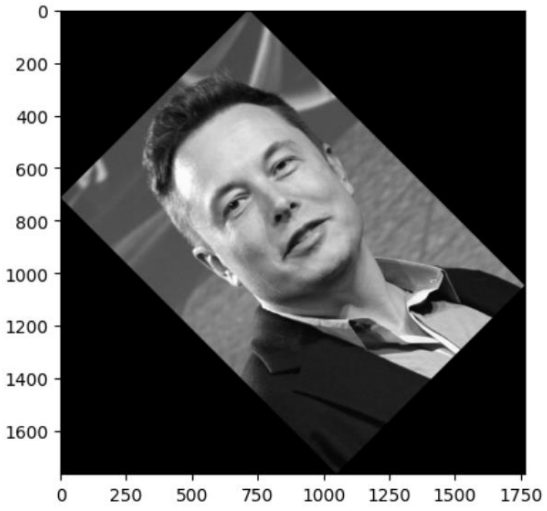
<matplotlib.image.AxesImage at 0x25f021fd950>

<matplotlib.image.AxesImage at 0x25f021b6750>



<matplotlib.image.AxesImage at 0x25f02690f50>

<matplotlib.image.AxesImage at 0x25f1bef9450>



RESULT:

Thus, the above Python program was successfully implemented and verified.

Ex.No	IMAGE FACE DETECTION AND COUNTING FACES	Date
10		

AIM

To detect and count the number of faces in an image using image processing techniques.

ALGORITHM

STEP 1: Install Required Libraries

STEP 2: Load the Image

STEP 3: Convert the image to grayscale to simplify the face detection process.

STEP 4: Load the Pre-trained Face Detection Model

STEP 5: Apply the face detection model to the grayscale image to identify face regions.

STEP 6: Draw Bounding Boxes Around Detected Faces

STEP 7: Count the Number of Faces

STEP 8: Save or display the image with detected faces and the count of faces.

PROGRAM:

```
import cv2
import matplotlib.pyplot as plt

def load_image(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img_rgb
def detect_faces(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))
    return faces
def visualize_faces(image, faces):
    # Draw rectangles around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
image_path = 'D:\Technologies\ Lab Excersizes\Ee.jpg'
image = load_image(image_path)
faces = detect_faces(image)
print(f'Number of faces detected: {len(faces)}')
visualize_faces(image, faces)
```

OUTPUT:

```
Number of faces detected: 2
```

RESULT:

Thus, the above Python program was successfully implemented and verified.