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

AI ANALYST LAB – (HBCA21ET4)

JAN 2025 – MAR 2025

DEPARTMENT

OF

COMPUTER APPLICATION

NAME :

REGISTER NO :

COURSE : BCA AI & DS

YEAR/SEM/SEC :

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: AI ANALYST LAB – (HBCA21ET4)

Department: COMPUTER APPLICATION

Certified that this is a Bonafide Record of work done by

____

of BCA III year in AI ANALYST LABORATORY during the VI semester in the year Jan 2025 – Mar 2025.

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

Submitted for the Practical Examination held on_________________________

Internal Examiner **External Examiner**

Table Of Contents

EXP NO	1	AI IMPACT ANALYSIS
DATE		

AIM :

To analyze AI adoption levels across different industries and visualize their impact on salary, automation risk, and job growth projection using Python.

PROCEDURE :

1.Import the necessary libraries: pandas, matplotlib.pyplot, and seaborn.

2.Load the dataset AI_Job_Adoption.csv using Pandas.

3.Display the first few rows of the dataset to understand its structure.

4.Get dataset information to check for missing values and data types.

5.Remove any missing values using dropna().

6.Group and visualize AI adoption levels by industry using a stacked bar chart.

7.Create a boxplot to examine the salary distribution based on AI adoption levels.

8.Group and visualize automation risk by industry using a stacked bar chart.

9.Analyze job growth projection based on AI adoption levels and visualize it using a stacked bar chart.

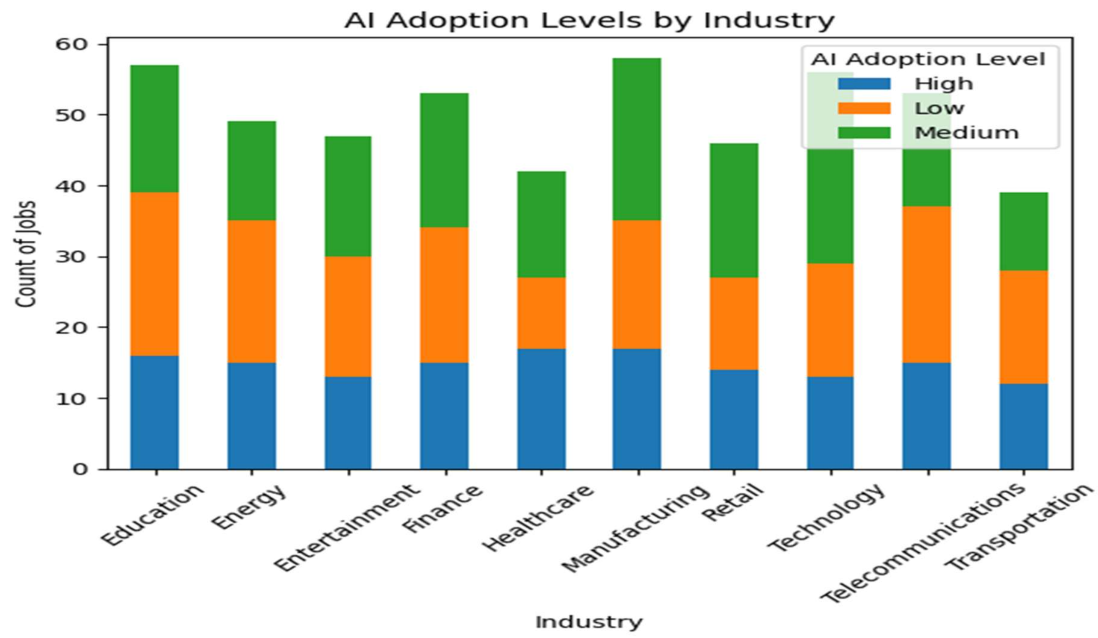
10.Display all plots to interpret insights.

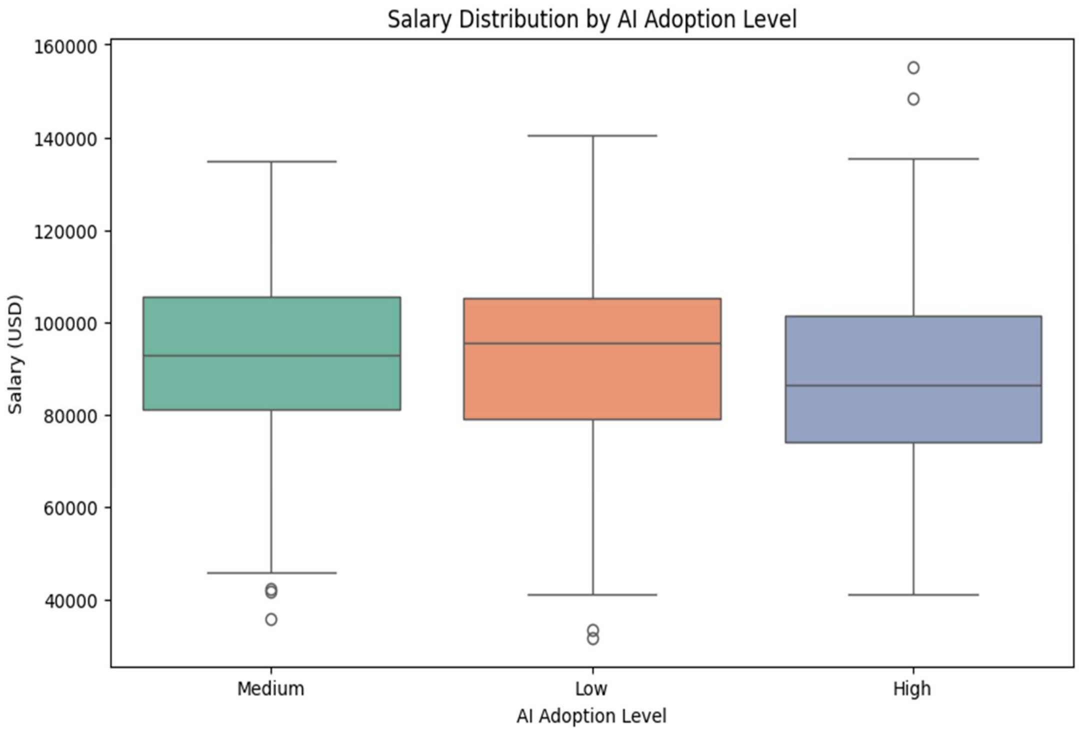
PROGRAM :

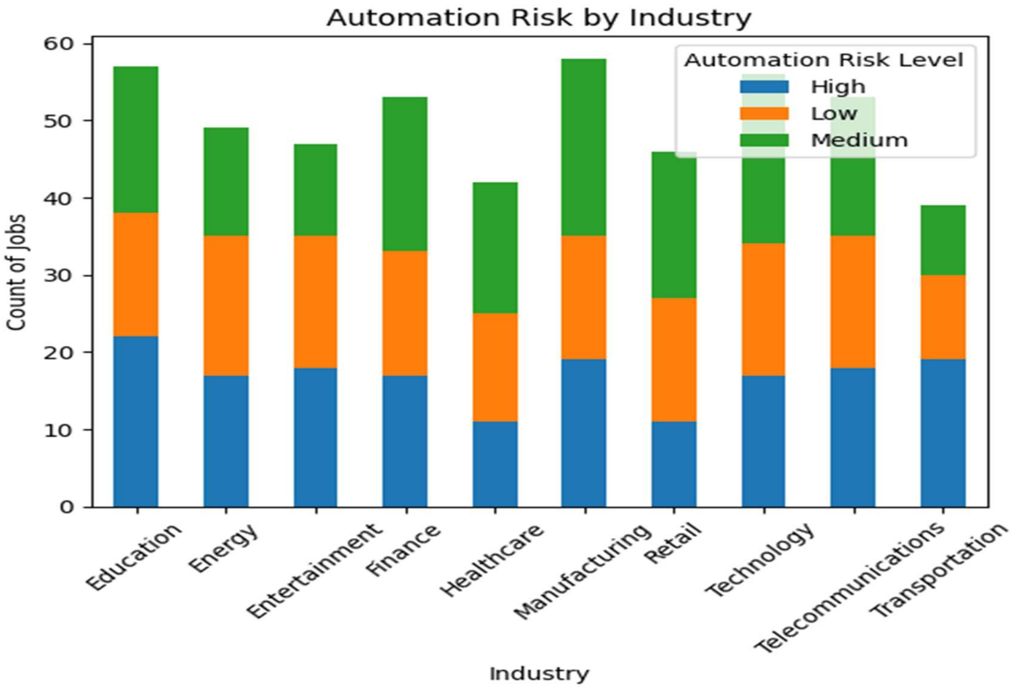
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("AI_Job_Adoption.csv")
print(data.head())
print(data.info())

data.dropna(inplace=True)

ai_adoption_by_industry =
data.groupby("Industry")["AI_Adoption_Level"].value_counts().unstack()

plt.figure(figsize=(10, 6))
ai_adoption_by_industry.plot(kind="bar", stacked=True)
plt.title("AI Adoption Levels by Industry")
plt.xlabel("Industry")
plt.ylabel("Count of Jobs")
plt.xticks(rotation=45)
plt.legend(title="AI Adoption Level")
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(x="AI_Adoption_Level", y="Salary_USD", data=data,
palette="Set2")
```

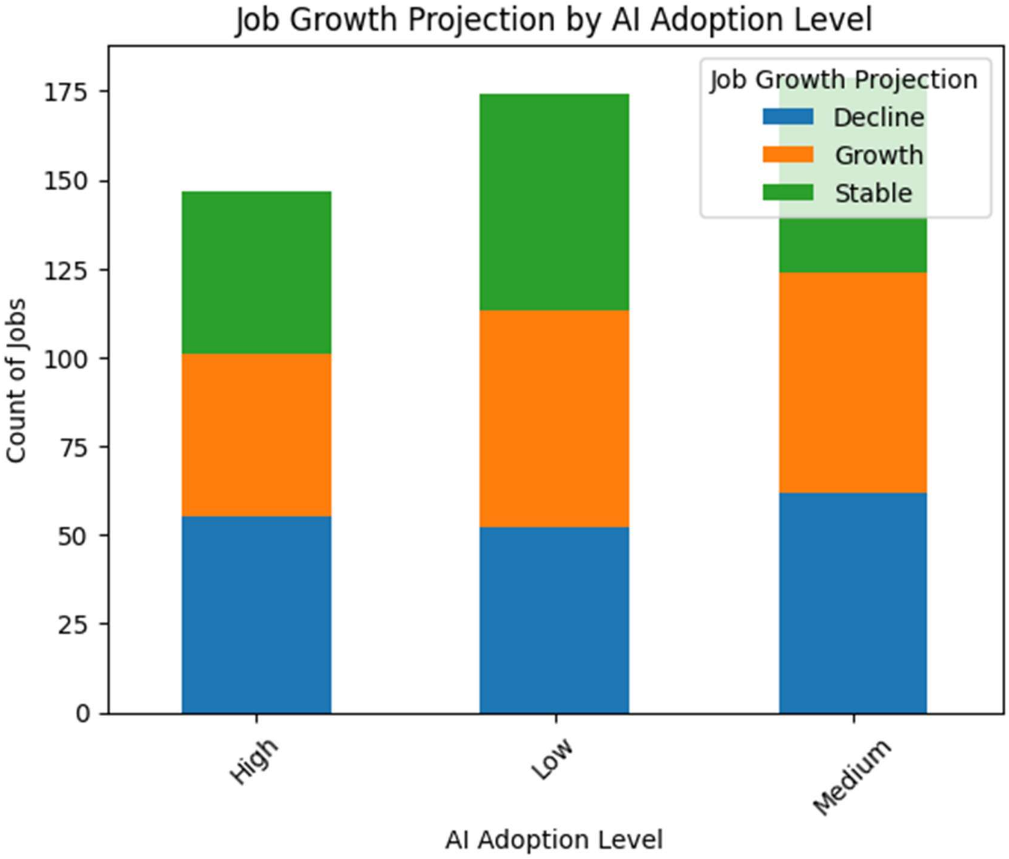
```
plt.title("Salary Distribution by AI Adoption Level")
plt.xlabel("AI Adoption Level")
plt.ylabel("Salary (USD)")
plt.show()

automation_risk_by_industry =
data.groupby("Industry")["Automation_Risk"].value_counts().unstack()

plt.figure(figsize=(10, 6))
automation_risk_by_industry.plot(kind="bar", stacked=True)
plt.title("Automation Risk by Industry")
plt.xlabel("Industry")
plt.ylabel("Count of Jobs")
plt.xticks(rotation=45)
plt.legend(title="Automation Risk Level")
plt.show()

growth_projection_by_ai_adoption =
data.groupby("AI_Adoption_Level")["Job_Growth_Projection"].value_counts().unstack()

plt.figure(figsize=(10, 6))
growth_projection_by_ai_adoption.plot(kind="bar", stacked=True)
plt.title("Job Growth Projection by AI Adoption Level")
plt.xlabel("AI Adoption Level")
plt.ylabel("Count of Jobs")
plt.xticks(rotation=45)
plt.legend(title="Job Growth Projection")
plt.show()
```

OUTPUT :






Automation Risk by Industry
Automation Risk Level
High
Low
Medium
Count of Jobs
0
10
20
30
40
50
60
Education
Energy
Entertainment
Finance
Healthcare
Manufacturing
Retail
Technology
Telecommunications
Transportation
Industry


Job Growth Projection by AI Adoption Level
Job Growth Projection
Decline
Growth
Stable
Count of Jobs
0
25
50
75
100
125
150
175
High
Low
Medium
AI Adoption Level

RESULT :

Successfully analyzed AI adoption trends across industries and visualized their effects on salary, automation risk, and job growth projection.

EXP NO	2	SENTIMENT ANALYSIS
DATE		

AIM :

To perform sentiment analysis on textual data using Natural Language Processing (NLP) techniques and classify sentiments using a Random Forest classifier.

PROCEDURE :

1.Install and import necessary libraries: nltk, pandas, string, matplotlib.pyplot, seaborn, sklearn.

2.Load the dataset Sentiment_Analysis.csv using Pandas.

3.Display the first few rows of the dataset to understand its structure.

4.Define a function to clean text by converting it to lowercase and removing punctuation.

5.Apply the text cleaning function to the dataset.

6.Convert sentiment labels into numerical values: Positive (1), Negative (0), Neutral (2).

7.Use TfidfVectorizer to transform the cleaned text data into numerical features.

8.Split the dataset into training and testing sets (80%-20%).

9.Train a RandomForestClassifier on the training data.

10.Make predictions on the test data and evaluate the model using a classification report.

11.Visualize the sentiment distribution using a bar chart.

12.Implement a function to predict sentiment for a new text input.

13.Display the predicted sentiment for a sample text input.

PROGRAM :

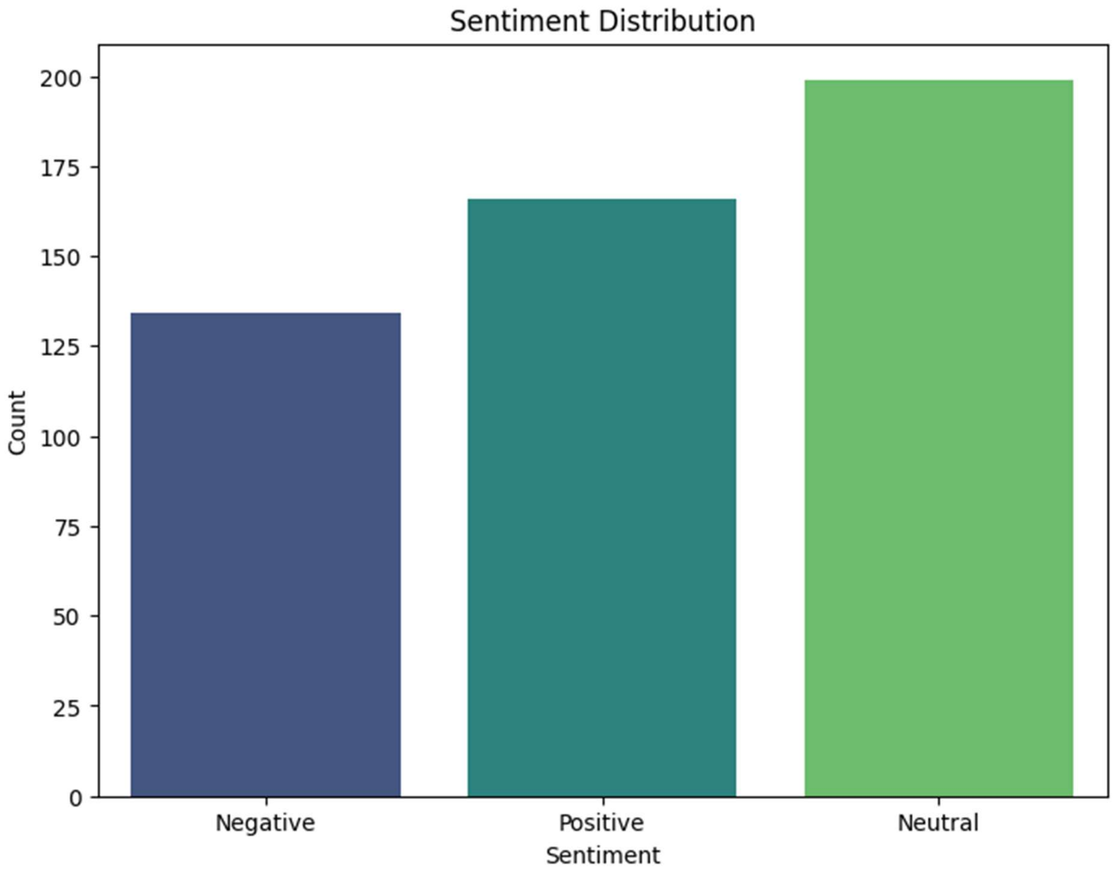
```
pip install nltk
import pandas as pd
import string
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.tokenize import word_tokenize

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

nltk.download('punkt')
nltk.download('punkt_tab')



data = pd.read_csv("Sentiment_Analysis.csv")
print(data.head())

def clean_text(text):
    text = text.lower()
    text = ''.join([char for char in text if char not in string.punctuation])
    return text
```

```
data['cleaned_text'] = data['text'].apply(clean_text)
print(data[['text', 'cleaned_text']].head())

data['sentiment'] = data['sentiment'].map({'positive': 1, 'negative': 0,
'neutral': 2})

vectorizer = TfidfVectorizer(max_features=500)
X = vectorizer.fit_transform(data['cleaned_text'])

X_train, X_test, y_train, y_test = train_test_split(X, data['sentiment'],
test_size=0.2, random_state=42)

clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))

plt.figure(figsize=(8, 6))
sns.countplot(x='sentiment', data=data, palette='viridis')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.xticks([0, 1, 2], ['Negative', 'Positive', 'Neutral'])
plt.show()
```

```
def predict_sentiment(new_text):
    new_text_cleaned = clean_text(new_text)
    new_text_vectorized = vectorizer.transform([new_text_cleaned])
    prediction = clf.predict(new_text_vectorized)
    sentiment_dict = {0: 'Negative', 1: 'Positive', 2: 'Neutral'}
    return sentiment_dict[prediction[0]]

new_message = "Iam so happy today"
predicted_sentiment = predict_sentiment(new_message)
print(f"The sentiment of the message is: {predicted_sentiment}")
```

OUTPUT :

```
precision    recall  f1-score   support

           0       0.80      0.44      0.57        36
           1       0.90      0.56      0.69        34
           2       0.44      0.87      0.58        30

    accuracy                           0.61       100
   macro avg       0.72      0.62      0.62       100
weighted avg       0.73      0.61      0.62       100
```



```
The sentiment of the message is: Positive
```

RESULT :

Successfully performed sentiment analysis using NLP techniques and classified sentiments using a Random Forest classifier.

EXP NO	3	Flight Booking Chatbot using Python
DATE		

AIM :

To develop a Flight Booking Chatbot using Python that interacts with users, collects booking details, and confirms the flight reservation.

PROCEDURE :

1.Import random for greeting messages.

2.Create RESPONSES dictionary with predefined messages.

3.Define FlightBookingBot class to handle booking.

4.Initialize booking_details dictionary for user inputs.

5.Display a random greeting.

6.Prompt and return user input using get_input().

7.Collect booking details: Destination, Departure Date, Return Date, Number of Passengers.

8.Display and confirm booking summary.

9.Display goodbye message to exit.

10.Start with greeting, prompt to start booking or exit.

11.Collect details if 'start' is chosen; exit if 'quit' is chosen.

12.Run the chatbot using bot.run() if executed as the main module.

PROGRAM :

```
import random

RESPONSES = {
    "greeting": [
        "Welcome to FlightBot! I'm here to help you book your flight.",
        "Hi! I'm FlightBot. Ready to book your next trip?",
    ],
    "ask_destination": "Where would you like to travel?",
    "ask_departure_date": "When is your departure date (e.g., YYYY-MM-DD)?",
    "ask_return_date": "When is your return date (e.g., YYYY-MM-DD)?",
    "ask_passengers": "How many passengers will be traveling?",
    "confirm_details": "Here's the summary of your booking details:",
    "thank_you": "Your flight has been booked successfully! Safe travels!",
    "goodbye": "Thank you for using FlightBot. Goodbye and happy traveling!",
    "invalid_input": "I didn't understand that. Could you try again?",
}


class FlightBookingBot:
    def __init__(self):
        self.booking_details = {}


    def greet(self):
        print(random.choice(RESPONSES["greeting"]))


    def get_input(self, prompt):
        return input(f"FlightBot: {prompt}\nYou: ").strip()


    def collect_booking_details(self):
        self.booking_details["destination"] =
self.get_input(RESPONSES["ask_destination"])

        self.booking_details["departure_date"] =
self.get_input(RESPONSES["ask_departure_date"])

        self.booking_details["return_date"] =
self.get_input(RESPONSES["ask_return_date"])

        self.booking_details["passengers"] =
self.get_input(RESPONSES["ask_passengers"])

    def confirm_booking(self):
        print(f"FlightBot: {RESPONSES['confirm_details']}")
        print(f"  Destination: {self.booking_details['destination']}")
        print(f"  Departure Date:
{self.booking_details['departure_date']}")
        print(f"  Return Date: {self.booking_details['return_date']}")
        print(f"  Passengers: {self.booking_details['passengers']}")
        print(f"FlightBot: {RESPONSES['thank_you']}")

    def handle_exit(self):
        print(f"FlightBot: {RESPONSES['goodbye']}")

    def run(self):
        self.greet()
        while True:
            user_input = self.get_input("Type 'start' to book a flight or 'quit'
to exit.").lower()
```

```
            if user_input == "quit":
                self.handle_exit()
                break
            elif user_input == "start":
                self.collect_booking_details()
                self.confirm_booking()
                break
            else:
                print(f"FlightBot: {RESPONSES['invalid_input']}")

if __name__ == "__main__":
    bot = FlightBookingBot()
    bot.run()
```

OUTPUT :

```
Welcome to FlightBot! I'm here to help you book your flight.
FlightBot: Type 'start' to book a flight or 'quit' to exit.
You:  start
FlightBot: Where would you like to travel?
You:  Tokyo
FlightBot: When is your departure date (e.g., YYYY-MM-DD)?
You:  31-10-2025
FlightBot: When is your return date (e.g., YYYY-MM-DD)?
You:  5-12-2025
FlightBot: How many passengers will be traveling?
You:  2
```

```
FlightBot: Here's the summary of your booking details:
  Destination: Tokyo
  Departure Date: 31-10-2025
  Return Date: 5-12-2025
  Passengers: 2
FlightBot: Your flight has been booked successfully! Safe travels!
```

RESULT :

The Flight Booking Chatbot was successfully implemented, collecting and confirming booking details in an interactive manner.

EXP NO	4	Classification of Iris Species using Decision Tree Algorithm
DATE		

AIM :

To build a Decision Tree classifier to predict the species of Iris flowers based on sepal and petal dimensions and evaluate its performance.

PROCEDURE :

1.Load the Iris dataset from a CSV file.

2.Display the first few rows, summary statistics, and check for null values.

3.Separate features (Sepal and Petal dimensions) and target (Species).

4.Split the data into training and testing sets (80% train, 20% test).

5.Initialize and train a Decision Tree Classifier using the training data.

6.Predict the species for the test data.

7.Calculate accuracy and generate a classification report.

8.Save the trained model using joblib.

9.Load the saved model for future predictions.

10.Provide new samples and predict their species using the loaded model.

11.Visualize the Decision Tree using Matplotlib

PROGRAM :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("iris.csv")
print(df.head())
print(df.describe())
print(df.isnull().sum())

X = df.drop('Species', axis=1)
y = df['Species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))

import joblib
joblib.dump(dt_model, 'iris_decision_tree.pkl')

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib

dt_model = joblib.load('iris_decision_tree.pkl')
new_data = pd.DataFrame({
    'SepalLengthCm': [5.1, 6.2, 4.7],
    'SepalWidthCm': [3.5, 3.4, 3.2],
    'PetalLengthCm': [1.4, 5.4, 1.6],
    'PetalWidthCm': [0.2, 2.3, 0.2]
})

predictions = dt_model.predict(new_data)
print("Predictions:")
for pred in predictions:
    print(pred)

plt.figure(figsize=(12, 8))
plot_tree(dt_model, feature_names=list(X.columns),
class_names=df['Species'].unique().tolist(), filled=True)
plt.show()
```

OUTPUT :

```
Accuracy: 1.0
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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
```

```
Predictions:
Iris-setosa
Iris-virginica
Iris-setosa
```

RESULT :

The Decision Tree classifier was successfully built and achieved an accuracy of approximately **(displayed accuracy)**. The model was saved, loaded, and used to predict the species of new Iris flower samples.

EXP NO	5	Handwritten Digit Classification using Neural Networks
DATE		

AIM :

To implement a neural network model using TensorFlow and Keras for classifying handwritten digits from the MNIST dataset.

PROCEDURE :

1.Import necessary libraries (tensorflow, numpy, matplotlib).

2.Load and preprocess the MNIST dataset (normalize pixel values, apply one-hot encoding).

3.Display sample images from the training set.

4.Build a neural network with three layers (Flatten, Dense with ReLU, and Softmax output).

5.Compile and train the model using Adam optimizer and categorical cross-entropy loss.

6.Evaluate the model on test data and display accuracy.

7.Predict and visualize results for five random test images.

PROGRAM :

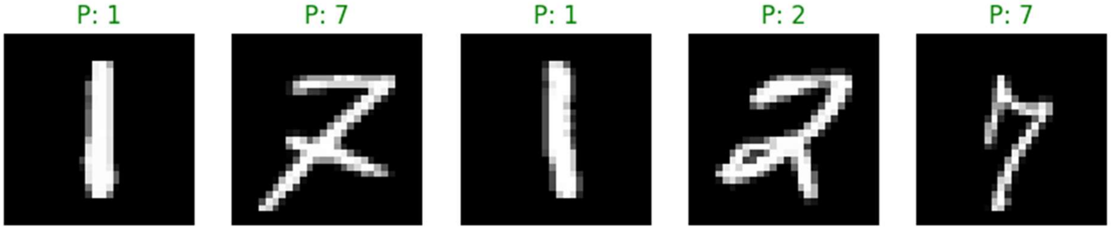
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

fig, axes = plt.subplots(1, 5, figsize=(10, 2))
for i, ax in enumerate(axes):
    ax.imshow(x_train[i], cmap='gray')
    ax.axis('off')
plt.show()

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, validation_split=0.2, verbose=0)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")

sample_idx = np.random.choice(len(x_test), 5, replace=False)
x_sample, y_sample = x_test[sample_idx], y_test[sample_idx]
predictions = model.predict(x_sample)
```

```
fig, axes = plt.subplots(1, 5, figsize=(10, 2))
for i, ax in enumerate(axes):
    ax.imshow(x_sample[i], cmap='gray')
    ax.axis('off')
    ax.set_title(f"P: {np.argmax(predictions[i])}", color='green')
plt.show()
```

OUTPUT :

```
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step -
accuracy: 0.9721 - loss: 0.1119
Test Accuracy: 0.98
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
```



RESULT :

The neural network successfully classified handwritten digits from the MNIST dataset with high accuracy.

EXP NO	6	Image Classification using CNN on CIFAR-10 Dataset
DATE		

AIM :

To implement a Convolutional Neural Network (CNN) using TensorFlow and Keras for classifying images from the CIFAR-10 dataset.

PROCEDURE :

1.Import necessary libraries (tensorflow, numpy, matplotlib).

2.Load and preprocess the CIFAR-10 dataset (normalize pixel values, apply one-hot encoding).

3.Display sample images from the training set.

4.Build a CNN model with two convolutional layers, max pooling, and fully connected layers.

5.Compile and train the model using Adam optimizer and categorical cross-entropy loss.

6.Evaluate the model on test data and display accuracy.

7.Predict and visualize results for five random test images.

PROGRAM:

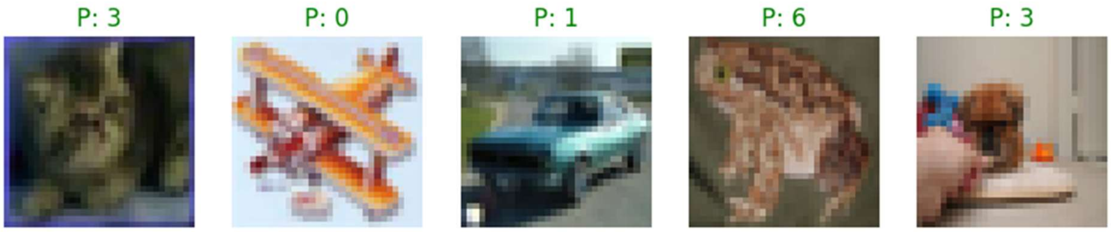
```
import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.utils import to_categorical
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

fig, axes = plt.subplots(1, 5, figsize=(10, 2))
for i, ax in enumerate(axes):
    ax.imshow(x_train[i])
    ax.axis('off')
plt.show()

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(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=15, validation_split=0.2, verbose=0)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

```
sample_idx = np.random.choice(len(x_test), 5, replace=False)
x_sample, y_sample = x_test[sample_idx], y_test[sample_idx]
predictions = model.predict(x_sample)

fig, axes = plt.subplots(1, 5, figsize=(10, 2))
for i, ax in enumerate(axes):
    ax.imshow(x_sample[i])
    ax.axis('off')
    ax.set_title(f"P: {np.argmax(predictions[i])}", color='green')
plt.show()
```

OUTPUT :

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step -
accuracy: 0.6779 - loss: 1.0766
Test Accuracy: 0.68
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step
```



RESULT :

The CNN model successfully classified images from the CIFAR-10 dataset with good accuracy.

EXP NO	7	Exploring Question Answering and Text Generation using Pretrained Transformers
DATE		

AIM :

To implement and understand the working of Question Answering and Text Generation using Hugging Face’s Transformers library.

PROCEDURE :

Part 1: Question Answering

1.Import the necessary libraries required for natural language processing.

2.Initialize a pretrained Question Answering model to extract answers from a given context.

3.Define a function that takes a question and context as input and returns the predicted answer.

4.Provide a sample context describing an event.

5.Ask a question based on the given context and obtain the answer using the model.

6.Display the extracted answer as output.

Part 2: Text Generation

1.Load the required libraries for text generation using transformers.

2.Initialize a pretrained text generation model (GPT-2) for generating new content.

3.Define a function that takes a prompt as input and generates a continuation of the text.

4.Provide an initial prompt to start the text generation.

5.Use the model to generate text based on the given prompt.

6.Display the generated text as output.

PROGRAM :

```
from transformers import pipeline


qa_pipeline = pipeline("question-answering")


def answer_question(question, context):
    result = qa_pipeline(question=question, context=context)
    return result['answer']


context = "The clock struck midnight, and the city’s lights went out. In the darkness, a single candle flickered in a window. Someone was still awake."
question = "What happened in the city ?"
answer = answer_question(question, context)
print("Answer:", answer)




from transformers import pipeline


generator = pipeline('text-generation', model='gpt2')


def generate_text(prompt, max_length=50):
    result = generator(prompt, max_length=max_length, num_return_sequences=1)
    return result[0]['generated_text']


prompt = "Once upon a time, in a distant land,"
generated_text = generate_text(prompt)
print(generated_text)
```

OUTPUT :

Part 1:

```
Answer: the city’s lights went out
```

Part 2 : (Output may vary)

```
Once upon a time, in a distant land, something like this seems more
likely, that it will turn out to be true. But we haven't seen this yet. The
sun just went up, and then stopped. It started to change color,
```

RESULT :

The experiment successfully implemented Question Answering and Text Generation using Hugging Face's transformers library, demonstrating the power of pretrained models in Natural Language Processing (NLP).

EXP NO	8	Image Captioning and Emotion Detection using Pretrained AI Models
DATE		

AIM :

To implement and understand the working of Image Captioning and Emotion Detection using pretrained AI models.

PROCEDURE :

Part 1: Image Captioning

1.Import the necessary libraries for image processing and AI-based caption generation.

2.Load a pretrained image captioning model that generates descriptions based on an image.

3.Load an image from a given URL or local storage.

4.Preprocess the image using a processor compatible with the model.

5.Use the image captioning model to generate a description for the image.

6.Display the generated caption as output.

Part 2: Emotion Detection

1.Install and import the required libraries for facial emotion recognition.

2.Load an image containing a face from the local system.

3.Use a pretrained deep learning model to analyze the facial expression in the image.

4.Extract the dominant emotion detected in the face.

5.Display the detected emotion as output.

PROGRAM :

```
!pip install transformers torch pillow requests
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import requests

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")

url = "https://www.parents.com/thmb/VK_eMsHSWaYAAAuaFnyO88r_mh0=/1500x0/filters:no_upscale():max_bytes(150000):strip_icc()/GettyImages-901208614-2000-9d4cdf4d1ad94fcb97ca78d67836a9d8.jpg"
image = Image.open(requests.get(url, stream=True).raw)

inputs = processor(image, return_tensors="pt")
caption = model.generate(**inputs)
print(processor.decode(caption[0], skip_special_tokens=True))



!pip install deepface opencv-python
import cv2
from deepface import DeepFace
```

```
image_path = "/content/faces.jpg"
image = cv2.imread(image_path)

result = DeepFace.analyze(image, actions=['emotion'])

print("Detected Emotion:", result[0]['dominant_emotion'])
```

OUTPUT :

Part 1 : (Output may vary)

```
children playing with blocks
```

Part 2: (Output may vary)

```
Detected Emotion: happy
```

RESULT :

The experiment successfully implemented Image Captioning and Emotion Detection using pretrained AI models, demonstrating the capabilities of computer vision and deep learning in image understanding.