Experiment - 1

Basic Python codes

1) Find the squareroot of number

```
import math
 n = int (input ())
 print (math.sqrt (n))
```

-> 4

2.0

2) Swap two numbers

```
a = int (input ("enter 1st num:"))
b = int (input ("enter 2nd num:"))
c = a
a = b
b = c
print(a)
print(b)
```

-> enter 1st num = 2

enter 2nd num = 3

3

2

3) Check num is positive or not

```
n = int (input())
if (n> =0):
 print ("num is positive")
 else:
 print ("num is negative")
```

-> 4

num is positive

4) Check num is even or odd

```
n=int(input())
if (n% 2==0):
Print ("num is even")
else:
print ("num is odd")
```

-> 5

num is odd

5) Find the factorial of num

```
import math
n= int(input())
print(math.factorial(n))
```

-> 3
6

6) Check whether the number is palindrome

```
n=input()
if (n==n [::-1]):
    print("Palindrome")
    else:
    Print(" not palindrome")
```

-> 323
palindrome

7) Use pandas & create a dataset, print age & age>25

```
import pandas as pd
data = {"Name": ["Hari", "Vamshi", "Aftan"],
        "Age" : [20, 25,27],
        "City" : ["bgl", "hyd", "pune"]}
df = pd.DataFrame(data)
print(df)
```

	Name	Age	City
0	Hari	20	bgl
1	Vamshi	25	hyd
2	Alton	27	pune

df["Age"]

	Age
0	20
1	25
2	27

df [df ["Age"] >25]

	Name	Age	City
2	Vamshi	27	Pune

8) Implement AND gate with 3 inputs

```
def AND_gate (a, b, c):
        return (a and b & c
    input = [ (0, 0, 0),
              (0, 0, 1),
              (0, 1, 0),
              (0, 1, 1),
              (1, 0, 0),
              (1, 0, 1),
              (1, 1, 0),
              (1, 1, 1) ]
print ("ABC | output").
print ("_ _ _ _ _")
for a, b, c in inputs:
print(f "{a} {b} {c} |{int(AND_gate(a, b, c))}")
```

e>25

→

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

O/P verified

11/8/25

Experiment - 2

a)
```
from transformers import pipeline
sentiment_analyzer = pipeline("sentiment analysis")
text = " I am feeling happy"
Sentiment_result = sentiment_analyzer(text)
print ("Sentiment analysis:")
print(f"Sentiment : {sentiment_result [0]['label']}
    (score: {sentiment_result [0]['score']:.2f
```

Output:-

b) Sentiment Analysis:
Sentiment: POSITIVE (score : 1.00)

```
from transformer import pipeline
translator_en_to_fr = pipeline ("translation_in_to_fr",
                   model = "Helsinki-NLP/opus-mt-en-fr")
text = " I am feeling happy"
translation_result = translator_in_to_fr (text)
print ("Translation:")
print (f" Translated text : {translation_result [0]
                   [translation_text]}\n")
```

Output:
config.json 1.42kB
python_model.bin 100%
generation_config.json : 100%
tokenizer_config.json : 100%
model.safetensors : 100%
source.spm : 100%
target.spm : 100%
vocab.json.

Translation:
Translated text: Je me sens heureuse.

O/p verified
8/11/24

c)

```
! Pip install pytesseract
! pip install pillow
! pip install transformer

import pytesseract
from PIL import Image
from transformers import pipeline
input = "/content/image.p
input-image = Image.open(input)
text_output = Pytesseract.image_to_string(input_image).strip()
print (text_output)

from transformers import pipeline
Sentiment analyzer = pipeline ("sentiment = analysis")
text = text_output
Sentiment_result = sentiment_analyzer(text)
print ("Sentiment analysis:")
print (f"Text: {text}")
print (f" Sentiment: {sentiment_result[0]['label']}
    Score: {sentiment_result [0] ['score']:.2f}").
```

Output:

Sentiment analysis:
Text: It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness

Sentiment: NEGATIVE Score: 0.98

O/p ✓

13/8/25

Experiment-3

Variational Autoencoder

```
import numpy as np
import Tensorflow as tf
import keras
from keras import layers.
```

Algorithm:-

1) Define Sampling layer

-> Take mean & log variance from encoder output

-> sample z using epsilion * Exp(0.5 * log_var) + mean

2) Build Encoder

-> input (28,28.1) image

-> Apply Conv2D layers to extract features

-> flatten and use dense layer

-> Output:- mean, log_var & Sampled latent vector z

3) Build Decoder:-

-> input: latent vector (latent_dim)

-> Dense - reshape to (7.7.64)

-> Conv2D Transpose layer to upsample back to (28,28,1) image

4) Define VAE model

-> Combine encoder decoder

-> train the model apply gradients to update weights.

5) Train VAE

-> load & normalize fashion MNIST data

-> fit model for desired epochs.

6) Plot latent space

- → create a 2D grid of latent point
- → for each (x_i, y_i), decode to generate an image
- → place image in correct spot in large figure grid
- → label areas as "z[0]" & "z[1]"
- → Show the generated grid image.

O/p verified

13/8/25

Experiment - 4

Generative Adversarial Network

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

Algorithm:-

1) Load & prepare data:
Get MNIST images, scale them, add a channel, & create a batched dataset.

2) Build generator:-
Create a model that takes noise & generates images.

3) Build discriminator:-
Create a model that takes an image & o/p's if it's real or fake (using conv2D)

4) Setup Loss & Optimizers:-
Use binary cross-entropy for loss, define generator & discriminator loss function. Use Adam optimizer

5) Define Training Step:-
Create a function to perform one step: generate fake images, evaluate real/fake with discriminator, calculate losses, compute gradients & update model weights

6) Run Training Loop:-
Iterate through epochs & dataset batches, calling training step. Generate & save images after epoch

O/P
20/8/25

Experiment - 5

Image Generation:

→ Python Code:

```
# 1) Install dependencies
# import libraries
import torch
from diffusers import PixArtAlphaPipeline

# 2) Load a Transformer
model_id = "PixArt-alpha/PixArt-XL-2-512x512"
dtype = torch.float16 if torch.cuda.is_available() else
torch.float32

# 3) Prompt for face
prompt = ("Ultra - realistic close-up portrait of
a young adult human, neutral expression"
 "natural soft lighting, 85mm lens bokeh, high
detail skin, photographic quality")
negative_Prompt = ("blurry low quality, extra finger
    Extra eyes, deformed")

# 4) Generate image
generator = torch.Generator(device = device). manu
                                              seed(42
image = pipe(
    ....
    ....
    ...). images[0]
```

5) Save & display
image. save ("face.png")
print ("saved to face.png")

Show image inside colab.
image. show().

O/p verified
20/8/25

Experiment - 7

Conditional GAN

1) Initialization:

-> Set hyperparameters

-> Initalize Generator G and Discriminator D

-> Define loss function and optimizers

2) Data Preperation:

-> Load MNIST Dataset

-> Normalize images to the range [-1, 1]

-> Convert labels into embeddings for Conditioning

3) Training Loop:-

a) For each batch

-> Generate noise vector z & sample labels

-> Generate fake images G(z,y)

-> Train Discriminator with Real & Fake

b) Update Parameters

-> Back propogate & update D

-> Back propogate & update G

4) Observation Per Epoch :-

-> Record Discriminator loss & Generator loss

-> Observe training stability (D-loss vs G-loss

5) Visualization:-

-> Every 5 epoches, generate digits for lables 0-9

-> Compare the clarity & quality of digits across epoches

-> Note improvements in shape, sharpness & diversity

6) Final Observation:-

-> After 91 epoches, evaluate how realistic the generated digits are

-> Observe whether the Generator produces label-conditioned digits correctly.

O/P verified
1/9/25

What is LLM

Large language Model:-

In AI & NLP, an LLM is a type of artificial intelligence model trained on vast amounts of text data to ~~understand~~ understand and generate human like language. Examples include GPT-4, GPT-3 & others. These models can perform tasks like answering questions, writing, translating, summarizing & more.

Experiment - 8

Retrieval - Augmented Generation

Input:

-> A set of structured & unstructured document
-> A user query
-> Pretrained embedding model
-> Pretrained generation model

Output:

-> A generated answer to the query based on retrieved documents.

Algorithm:-

1) **Preprocessing Documents:-**

-> Convert all structured document into a textual format that can be embedded
-> keep unstructured document as they are

2) **Embedding Documents:-**

-> Use a pretrained embedding model to compute vector embeddings for all documents.
-> Store these doc embeddings in the vector index structure for fast similarity search

3) **Query Processing & Retrieval**

-> Embed the incoming user query using the same embedding model
-> Perform similarity search on the vector i[illegible] with query embedding to retrieve the [illegible] K most relevant documents.

4) Answer Generation:-

→ Concatenate the retrieved documents into single textual content

→ Construct an input prompt for the generation model combining the query and retrieved contentstly.

→ Use a pretrained generative language model to generate a natural language answer unconditionally on prompt.

O/P verified

15/9/25

Experiment -10

Gradio App

1) Install Gradio

Use !Pip install gradio to make sure the library is available

2) Install & Import Gradio library

3) Define the function that processes input & produces output

4) Create Interface

wrap the function with a Gradio Interface or Blocks to connect input & outputs.

5) Gradio interface by linking

* function (fn = ...)
* Input Component (inputs = ---)
* Output Component (outputs = ----)

6) Launch the interface using .launch ()

7) Choose Input/output widgets.

Textbox → for text input

Slider → for numeric range

Checkbox → for true/false

Radio/Drop down → for Multiple - choice op

Image / Video / Audio → for multimedia in

JSON → for structured data

Blocks + Rows/columns → for custom lay...

8) Launch the App:-

Call. launch() to open a local Gradio app in the browser.

O/P verified

1/9/25

Github link verified for all ten experiments

8/10/2025

Exp - 11 a

Set up a FastAPI ChatApp

Algorithm:

1) Install requried packages: fastAPI. uvicorn nest asyncio. Pyngrok.
2) Import necessary modules
3) Kill existing ngrok processes
4) Apply nest asyncio patch
5) Create a FastAPI app
6) Define a message data model
7) Create API endpoints
8) kill any existing ngrok tunnels.
9) Start ngrok tunnel on port 8000
10) set ngrok auth token
11) Configure & run uvicorn server
12) Run their uvicorn server asynchronously inside the current event loop.

O/p verified
11/10/2025

Experiment - 11b

FastAPI chat Application with ngrok tunnel for public access in colab.

Algorithm:-

1) Install required packages (fast API, uvicorn, nest asyncio pyngrok)

2) Kill any existing ngrok processes & wait for termination.

3) Apply nest_asyncio to allow nested async event loops in colab.

4) Create a fastAPI app with endpoints to send & retreive chat messages stored in memory.

5) Kill any previous ngrok tunnels for a clean start.

6) Set ngrok authentication token.

7) Start an ngrok tunnel exposing port - 8000 & print the public url.

8) Configure & run the uvicorn server inside the current async loop to serve the Fast API app.

O/P verified

11/10/2025