Exp1-Techniques for Data Pre-processing: Mean Removal, Scaling, Normalization

a) **Mean Removal**

```
import csv
import statistics
def remove_mean(input_file, output_file):
    # Read the CSV file and extract the column of numbers
    with open(input_file, 'r') as file:
        reader = csv.reader(file)
        data = [float(row[0]) for row in reader]
    # Calculate the mean
    mean_value = statistics.mean(data)
    # Remove the mean from each data point
    mean_removed_data = [x - mean_value for x in data]
    # Write the mean-removed data to a new CSV file
    with open(output_file, 'w', newline='') as file:
        writer = csv.writer(file)
        for value in mean_removed_data:
            writer.writerow([value])
if __name__ == "__main__":
    input_file = "data.csv"
    output_file = "mean_removed_data.csv"
    remove_mean(input_file, output_file)
```

b) **Scaling**

```
import csv
def min_max_scaling(input_file, output_file, new_min=0, new_max=1):
    # Read the CSV file and extract the column of numbers
    with open(input_file, 'r') as file:
        reader = csv.reader(file)
        data = [float(row[0]) for row in reader]
    # Find the minimum and maximum values in the data
    current_min = min(data)
    current_max = max(data)

    # Perform Min-Max scaling on each data point
    scaled_data = [(x - current_min) / (current_max - current_min) * (new_max -
new_min) + new_min for x in data]

    # Write the scaled data to a new CSV file
    with open(output_file, 'w', newline='') as file:
        writer = csv.writer(file)
        for value in scaled_data:
            writer.writerow([value])

if __name__ == "__main__":
    input_file = "data.csv"
    output_file = "scaled_data.csv"
    min_max_scaling(input_file, output_file)
```

c) <u>Normalization</u>

```
import csv
def min_max_scaling(input_file, output_file, new_min=0, new_max=1):
    # Read the CSV file and extract the column of numbers
    with open(input_file, 'r') as file:
        reader = csv.reader(file)
        data = [float(row[0]) for row in reader]

    # Find the minimum and maximum values in the data
    current_min = min(data)
    current_max = max(data)

    # Perform Min-Max scaling on each data point
    normalized_data = [(x - current_min) / (current_max - current_min) *
(new_max - new_min) + new_min for x in data]

    # Write the normalized data to a new CSV file
    with open(output_file, 'w', newline='') as file:
        writer = csv.writer(file)
        for value in normalized_data:
            writer.writerow([value])

if __name__ == "__main__":
    input_file = "data.csv"
    output_file = "normalized_data.csv"
    min_max_scaling(input_file, output_file)
```

EXP 2 :

```
#  2 A  Naïve Bayes Classifier :

import pandas as pd

f = pd.DataFrame({'Weather':['Sunny', 'Rainy', 'Sunny', 'Sunny'],

            'Wind':['Mild', 'Mild', 'High', 'Mild'],

            'Temp':['Moderate', 'Mild', 'Moderate', 'Mild'],

            'go':['Yes', 'No', 'Yes', 'Yes']})

print(f.columns)


from sklearn.naive_bayes import GaussianNB as g

from sklearn.preprocessing import LabelEncoder as le

from sklearn.model_selection import train_test_split as tt


l = le()

for i in f.columns:

   f[i] = l.fit_transform(f[i])

   x = f.iloc[:, :3]

   y = f.iloc[:, 3]


xtr, xte, ytr, yte = tt(x, y, test_size=0.3)

gg = g()

gg.fit(xtr, ytr)

y_pred = gg.predict(xte)


from sklearn.metrics import accuracy_score

print(accuracy_score(yte, y_pred))
```

# 2.b.Support vector machine

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd
```

```
dataset = pd.read_csv(r"C:\Users\Online\Desktop\AI LA 2\Social_Network _Ads.csv")

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random_state = 0)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

print(cm)

accuracy_score(y_test,y_pred)

from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step =
0.01),np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha =
0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

   plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],c = ListedColormap(('pink', 'green'))(i), label = j)

plt.title('SVM (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()
```

2.c.Logistic regression

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score

from matplotlib.colors import ListedColormap


dataset = pd.read_csv(r"C:\Users\Online\Desktop\AI LA 2\diabetes (1).csv")


x = dataset.iloc[:, [4, 7]].values

y = dataset.iloc[:, 8].values


xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 0)


sc_x = StandardScaler()

xtrain = sc_x.fit_transform(xtrain)

xtest = sc_x.transform(xtest)


print (xtrain[0:10, :])


classifier = LogisticRegression(random_state = 0)

classifier.fit(xtrain, ytrain)

y_pred = classifier.predict(xtest)


cm = confusion_matrix(ytest, y_pred)
```

```
print ("Confusion Matrix : \n", cm)


print ("Accuracy : ", accuracy_score(ytest, y_pred))


X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
          stop = X_set[:, 0].max() + 1, step = 0.01),
          np.arange(start = X_set[:, 1].min() - 1,
          stop = X_set[:, 1].max() + 1, step = 0.01))


plt.contourf(X1, X2, classifier.predict(
      np.array([X1.ravel(), X2.ravel()]).T).reshape(
      X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))


plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())


for i, j in enumerate(np.unique(y_set)):
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)


plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Diabetes')
plt.legend()
plt.show()
```

# 2.d.DECISION TREE

```
import pandas as pd
import math
import numpy as np
```

```
data = pd.read_csv("dataset.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))



def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain


def ID3(examples, attrs):
    root = Node()


    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)

root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

# 2.e.Random forest

```
import pandas as pd

data=pd.read_csv(r"C:\Users\Online\Desktop\AI LA 2\heart.csv")

X =data.iloc[:,[1,2,3,4,5,6,7,8,9,10,11,12]].values

y =data.iloc[:,13].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()

rfc.fit(X_train, y_train)

y_pred=rfc.predict(X_test)

from sklearn import metrics

print("Classification Accuracy:", metrics.accuracy_score(y_test, y_pred)*100)

cm=metrics.confusion_matrix(y_test,y_pred)

print(cm)

import seaborn as sn

from matplotlib import pyplot as plt

plt.figure(figsize=(5,4))

sn.heatmap(cm,annot=True)

plt.xlabel('Predicted value')

plt.ylabel('Actual value')

plt.show()
```

3. KMEANS

```
import pandas as pd

import numpy as np

import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

from sklearn.preprocessing import MinMaxScaler

iris = pd.read_csv(r"C:\Users\Online\Downloads\Iris.csv")

x = iris.iloc[:, [ 1,2,3,4]].values


from sklearn.cluster import KMeans

wcss = []


for i in range(1, 11):

    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)

    kmeans.fit(x)

    wcss.append(kmeans.inertia_)



kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)

y_kmeans = kmeans.fit_predict(x)

plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'blue', label = 'Iris-setosa')

plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')

plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'red', label = 
'Centroids')


plt.legend()


plt.show()
```

EXP 4 – NLTK

```
import nltk

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.corpus import stopwords


# Download NLTK resources if not already installed

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('omw-1.4')


# Sample text

text = "The quick brown foxes are jumping over the lazy dogs. The dogs are not amused."


# Tokenization

tokens = word_tokenize(text)


# Remove stop words

stop_words = set(stopwords.words('english'))

filtered_tokens = [word for word in tokens if word.lower() not in stop_words]


# Stemming

stemmer = PorterStemmer()

stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]


# Lemmatization

lemmatizer = WordNetLemmatizer()

lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
```

```
# Display the results
print("Original Text:", text)
print("\nTokenization:", tokens)
print("\nFiltered Tokens (without stop words):", filtered_tokens)
print("\nStemmed Tokens:", stemmed_tokens)
print("\nLemmatized Tokens:", lemmatized_tokens)
```

EXP 5 – NLTK USING BAG OF WORDS

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist

nltk.download('punkt')
nltk.download('stopwords')

def preprocess_text(text):
   stop_words = set(stopwords.words('english'))
   word_tokens = word_tokenize(text)
   filtered_words = [word.lower() for word in word_tokens if word.isalpha() and word.lower() not in
stop_words]
   return filtered_words

def create_bow_model(texts):
   all_words = []
   for text in texts:
      words = preprocess_text(text)
      all_words.extend(words)

   word_freq = FreqDist(all_words)
   bow_model = {word: freq for word, freq in word_freq.items()}
```

```
    return bow_model

# Example usage
texts = [
    "The cat sat on the mat, and the mat was comfortable.",
    "She sang a sweet song, a song that touched everyone's heart.",
    "Coding coding can be challenging, but coding is also incredibly rewarding.",
]

bow_model = create_bow_model(texts)

# Print the Bag of Words model
print("Bag of Words Model:")
for word, freq in bow_model.items():
    print(f"{word}: {freq}")
```

EXP 6 : TOPIC MODELLING : IDENTIFYING PATTERNS IN TEXT DATA

```
import csv
import re

def identify_patterns(csv_file_path, column_name):
    patterns = {}

    with open(csv_file_path, 'r') as csvfile:
        reader = csv.DictReader(csvfile)

        for row in reader:
            text = row[column_name]

            # Example pattern: finding words that start with 'pattern'
            pattern_matches = re.findall(r'Female', text, flags=re.IGNORECASE)

            # Update patterns dictionary with matches
            for match in pattern_matches:
                if match in patterns:
                    patterns[match] += 1
                else:
                    patterns[match] = 1

    return patterns
csv_file_path = r"C:\Users\Online\Desktop\AI LAB 6\Social_Network _Ads (1).csv"  # Update with your CSV file path
column_name = 'Gender'     # Update with the actual column name in your CSV file

result = identify_patterns(csv_file_path, column_name)

# Display the identified patterns and their counts
for pattern, count in result.items():
    print(f"Pattern: {pattern}, Count: {count}")
```

EXP 7 : HMM

```
import numpy as np
from hmmlearn import hmm

# Step 1: Define Model Parameters
n_states = 2  # Number of hidden states (Rainy and Sunny)

# Transition matrix (A): Probability of transitioning from one state to another
```

```
trans_matrix = np.array([[0.7, 0.3], [0.4, 0.6]])

# Emission matrix (B): Probability of observing an emission given the current state
emission_matrix = np.array([[0.1, 0.4, 0.5], [0.6, 0.3, 0.1]])

# Initial state probabilities (π): Probability distribution of starting in each state
initial_probs = np.array([0.6, 0.4])

# Step 2: Create HMM Model
model = hmm.MultinomialHMM(n_components=n_states,
                          startprob_prior=initial_probs,
                          transmat_prior=trans_matrix,
                          n_iter=100)

# Step 3: Generate Training Data (for simplicity, you can use a pre-existing dataset)
# Observations: 0 - Umbrella, 1 - Jacket, 2 - T-shirt
train_data = np.array([[0, 1, 2, 0, 1, 2, 0, 2, 1]])

# Reshape the array if needed
train_data = train_data.reshape(-1, 1)

# Step 4: Fit the Model
model.fit(train_data)

# Step 5: Predict States for a New Sequence
```

```
new_data = np.array([[0, 2, 1]])  # Umbrella, T-shirt, Jacket

new_data = new_data.reshape(-1, 1)

predicted_states = model.predict(new_data)



# Map numerical predictions to weather states

weather_states = ['Rainy', 'Sunny']

predicted_states_text = [weather_states[state] for state in predicted_states]



# Display Results

print("Predicted Weather States:", predicted_states_text)
```

EXP 8 : A HEURISTIC SEARCH TECHINQUE

Concept of Heuristic Search Technique

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed

Heuristic Search Techniques in AI

Other names for these are Informed Search, Heuristic Search, and Heuristic Control Strategy. These are effective if applied correctly to the right types of tasks and usually demand domain-specific information.

Before move on to describe certain techniques, let’s first take a look at the ones we generally observe. Below, we name a few.

- Best-First Search
- A* Search
- Bidirectional Search
- Tabu Search
- Beam Search
- Simulated Annealing
- Hill Climbing
  - Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSP)

Let’s talk of a magic square. This is a sequence of numbers- usually integers- arranged in a square grid. The numbers in each row, each column, and each diagonal all add up to a constant which we call the *Magic Constant*. Let’s implement this with Python.

A* Search Algorithm and Its Basic Concepts

A* algorithm works based on heuristic methods, and this helps achieve optimality. A* is a different form of the best-first algorithm.

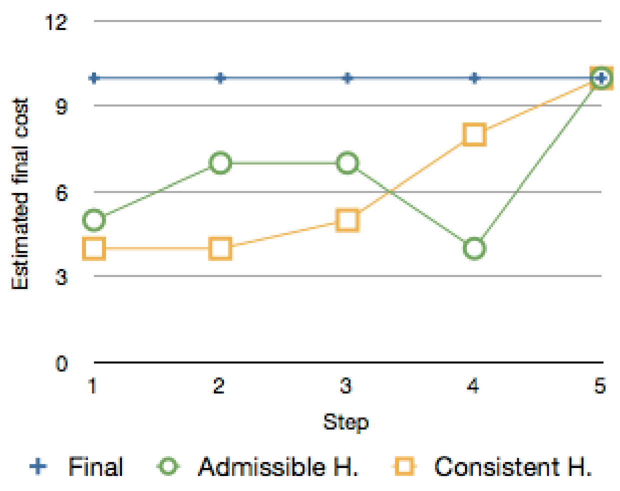
When A* enters into a problem, firstly, it calculates the cost to travel to the neighboring nodes and chooses the node with the lowest cost. If The f(n) denotes the cost, A* chooses the node with the lowest f(n) value. Here 'n' denotes the neighboring nodes. The calculation of the value can be done as shown below:

f(n)=g(n)+h(n)f(n)=g(n)+h(n)

g(n) = shows the shortest path's value from the starting node to node n
h(n) = The heuristic approximation of the value of the node

The heuristic value has an important role in the efficiency of the A* algorithm. To find the best solution, you might have to use different heuristic functions according to the type of the problem. However, the creation of these functions is a difficult task, and this is the basic problem we face in AI

What is a Heuristic Function?



Essentially, a heuristic function helps algorithms to make the best decision faster and more efficiently. This ranking is based on the best available information and helps the algorithm decide the best possible branch to follow. Admissibility and consistency are the two fundamental properties of a heuristic function.

Admissibility:

A heuristic function is admissible if it can effectively estimate the real distance between a node 'n' and the end node. It never overestimates; if it ever does, it will be denoted by 'd', which also denotes the accuracy of the solution.

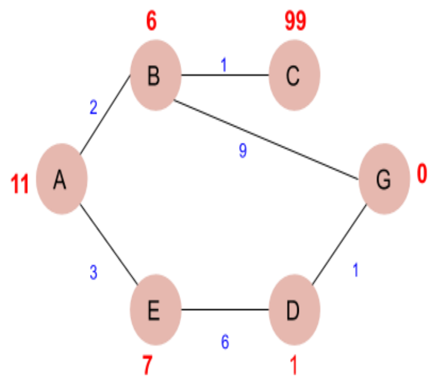
Consistency:

A heuristic function is consistent if the estimate of a given heuristic function turns out to be equal to or less than the distance between the goal (n) and a neighbor and the cost calculated to reach that neighbor.

A* is indeed a very powerful algorithm used to increase the performance of artificial intelligence. It is one of the most popular search algorithms in AI. The sky is the limit when it comes to the potential of this algorithm. However, the efficiency of an A* algorithm highly depends on the quality of its heuristic function.

Implementation with Python

In this section, we are going to find out how the A* search algorithm can be used to find the most cost-effective path in a graph. Consider the following graph below.



The numbers written on edges represent the distance between the nodes, while the numbers written on nodes represent the heuristic values. Let us find the most cost-effective path to reach from start state A to final state G using the A* Algorithm.

Let's start with node A. Since A is a starting node, therefore, the value of g(x) for A is zero, and from the graph, we get the heuristic value of A is 11, therefore

g(x) + h(x) = f(x)
0+ 11 =11
Thus for A, we can write
A=11
Now from A, we can go to point B or point E, so we compute f(x) for each of them

$A \rightarrow B = 2 + 6 = 8$
$A \rightarrow E = 3 + 6 = 9$

Since the cost for $A \rightarrow B$ is less, we move forward with this path and compute the f(x) for the children nodes of B

Since there is no path between C and G, the heuristic cost is set to infinity or a very high value

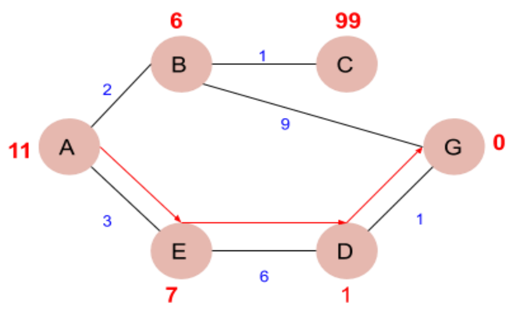
$A \rightarrow B \rightarrow C = (2 + 1) + 99 = 102$
$A \rightarrow B \rightarrow G = (2 + 9) + 0 = 11$

Here the path $A \rightarrow B \rightarrow G$ has the least cost but it is still more than the cost of $A \rightarrow E$, thus we explore this path further
$A \rightarrow E \rightarrow D = (3 + 6) + 1 = 10$

Comparing the cost of $A \rightarrow E \rightarrow D$ with all the paths we got so far and as this cost is least of all we move forward with this path. And compute the f(x) for the children of D.

$A \rightarrow E \rightarrow D \rightarrow G = (3 + 6 + 1) + 0 = 10$

Now comparing all the paths that lead us to the goal, we conclude that $A \rightarrow E \rightarrow D \rightarrow G$ is the most cost-effective path to get from A to G.



EXP 9 : TIC TAC TOE

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)


def check_winner(board, player):
    # Check rows, columns, and diagonals
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
        return True
    return False


def is_board_full(board):
    return all(board[i][j] != " " for i in range(3) for j in range(3))


def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    players = ["X", "O"]
    current_player = players[0]

    while True:
        print_board(board)
```

```
# Get player move
while True:
    row = int(input("Enter row (0, 1, or 2): "))
    col = int(input("Enter column (0, 1, or 2): "))
    if 0 <= row < 3 and 0 <= col < 3 and board[row][col] == " ":
        break
    else:
        print("Invalid move. Try again.")

# Make the move
board[row][col] = current_player

# Check for a winner
if check_winner(board, current_player):
    print_board(board)
    print(f"Player {current_player} wins!")
    break

# Check for a tie
if is_board_full(board):
    print_board(board)
    print("It's a tie!")
    break

# Switch to the other player
```

```
        current_player = players[1] if current_player == players[0] else players[0]

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

EXP 10 : SINGLE AND MULTI LAYERED PERCEPTRON

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load and preprocess the Iris dataset
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_test = StandardScaler().fit_transform(X_train), StandardScaler().fit_transform(X_test)

# Define and compile a single-layer neural network
model_single_layer = models.Sequential([layers.Dense(64, 'relu', input_shape=(4,)), layers.Dense(3, 'softmax')])
model_single_layer.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_single_layer.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test))

# Evaluate the single-layer model
single_layer_accuracy = accuracy_score(y_test, np.argmax(model_single_layer.predict(X_test), axis=1))
print(f"\nSingle-layer Neural Network - Accuracy: {single_layer_accuracy}")

# Define and compile a multi-layer neural network
model_multi_layer = models.Sequential([layers.Dense(64, 'relu', input_shape=(4,)), layers.Dense(32, 'relu'), layers.Dense(3, 'softmax')])
model_multi_layer.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_multi_layer.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test))

# Evaluate the multi-layer model
multi_layer_accuracy = accuracy_score(y_test, np.argmax(model_multi_layer.predict(X_test), axis=1))
print(f"\nMulti-layer Neural Network - Accuracy: {multi_layer_accuracy}")
```

EXP 11 : BUILDING LINEAR REGRESSION USING ANN

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Generate some random data for demonstration
np.random.seed(0)
X_train = np.random.rand(100, 1)
y_train = 2 * X_train + 1 + 0.1 * np.random.randn(100, 1)

# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=(1,))
])

# Compile the model
model.compile(optimizer='sgd', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=100, verbose=0)

# Plot the training loss over epochs
plt.plot(history.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error Loss')
plt.title('Training Loss')
plt.show()

# Make predictions on new data
X_test = np.array([[0.2], [0.5], [0.8]])
predictions = model.predict(X_test)

# Print the predictions
for i in range(len(X_test)):
    print(f"Input: {X_test[i][0]}, Predicted)
```

EXP 12 : IMAGE CLASSIFIER : AN APPLICATION OOF DEEP LEARNING

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

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

# Load and preprocess the MNIST dataset

mnist = tf.keras.datasets.mnist

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

X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel values

# Define the CNN model

model = Sequential([

    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),

    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
model.fit(X_train, y_train, epochs=5, validation_split=0.1)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'\nTest accuracy: {test_acc}')
```