EXP1:

```
%pip install pgmpy
import pandas as pd
import numpy as np
from pgmpy.models import BayesianNetwork, DiscreteBayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
model = DiscreteBayesianNetwork([('Prize','Host'),('Choice','Host')])
cpd_prize = TabularCPD(variable='Prize', variable_card = 3,
                      values=[[1/3],[1/3],[1/3]],
                      state_names={'Prize':[1,2,3]})
cpd_choice = TabularCPD(variable="Choice", variable_card = 3,
                      values=[[1/3],[1/3],[1/3]],
                      state_names={'Choice':[1,2,3]})
host_values = np.zeros((3,9))
host_doors= [1,2,3]
for i, prize in enumerate(host_doors):
 for j, choice in enumerate(host_doors):
   valid = [door for door in host_doors if door != prize and door != choice]
   for v in valid:
     host_values[v-1][3*i + j] = 1/len(valid)
cpd_host = TabularCPD(variable="Host", variable_card = 3,
                      values=host_values,
                      evidence=['Prize',"Choice"],
                      evidence_card=[3,3],
                      state_names={'Host':[1,2,3], 'Prize':[1,2,3],
                      'Choice':[1,2,3]})
model.add_cpds(cpd_prize, cpd_choice, cpd_host)
model.check_model()
inference = VariableElimination(model)
query_result = inference.query(variables= ['Prize'],
                              evidence = {'Choice':1, 'Host':3})
print("P(Prize | Choice = 1, Host = 3):")
print(query_result)
print("\n Best Strategy: switch to the door with highest probability")
```

EXP2:

```
# ---------- Import Libraries ----------
from textblob import TextBlob
# ---------- Cognitive Function ----------
def analyze_sentiment(text):
   blob = TextBlob(text)
   polarity = blob.sentiment.polarity  # value between -1 (neg) and +1 (pos)
     if polarity > 0:
       sentiment = "😊 Positive"
   elif polarity < 0:
       sentiment = "😞 Negative"
   else:
       sentiment = "😐 Neutral"
   return sentiment, polarity
# ---------- Run Application ----------
print("🧠 Simple Cognitive Sentiment Analyzer 🧠")
print("----------------------------------------")
while True:
   text = input("\nEnter a sentence (or type 'exit' to quit): ")
   if text.lower() == 'exit':
       print("Goodbye! 👋")
       break
   sentiment, score = analyze_sentiment(text)
   print(f"Sentiment: {sentiment} (Score: {score:.2f})")
```

EXP3: *Dataset reqd*

```
import pandas as pd
df = pd.read_csv('/content/Marvel_Comics.csv')
df

df.head()

df.tail()

df.isnull().sum()
df.duplicated().sum()
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

```
#Histogram
import matplotlib.pyplot as plt
import seaborn as sns
plt.hist(df['comic_name'])
plt.title('Comic Name')
plt.show()
#ScatterPlot
plt.scatter(df['active_years'], df['comic_name'])
plt.xlabel('Active Years')
plt.ylabel('Cover Artist')
plt.show()
#Box Plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=df)
plt.title("Boxplot of Iris Dataset Features", fontsize=14)
plt.xlabel("Value")
plt.ylabel("Feature")
plt.show()
```

Exp4:

```
from copy import deepcopy
import matplotlib.pyplot as plt
# ---------- Helper Function ----------
def checkElementHelper(x, S):
   for e in S:
       if x == e[1]:
           return e[0]
   return 0
# ---------- Fuzzy Operations ----------
def union(setA, setB):
   X, Y = deepcopy(setA), deepcopy(setB)
   Z = []
   for i in X:
       mb = checkElementHelper(i[1], Y)
       Z.append([max(mb, i[0]), i[1]])
       if mb != 0:
           Y.remove([mb, i[1]])
   Z = Z + Y
   return Z
```

```
def intersection(setA, setB):
   X, Y = deepcopy(setA), deepcopy(setB)
   Z = []
   for i in X:
       mb = checkElementHelper(i[1], Y)
       if min(mb, i[0]) != 0:
           Z.append([min(mb, i[0]), i[1]])
   return Z
def complement(setA):
   Z = deepcopy(setA)
   for i in Z:
       i[0] = 1 - i[0]
   return Z
def fuzzy_sum(setA, setB):
   Z = []
   for i, j in zip(setA, setB):
       Z.append([i[0] + j[0] - (i[0]*j[0]), i[1]])
   return Z
def fuzzy_product(setA, setB):
   Z = []
   for i, j in zip(setA, setB):
       Z.append([i[0]*j[0], i[1]])
   return Z
def bounded_sum(setA, setB):
   Z = []
   for i, j in zip(setA, setB):
       Z.append([min(1, i[0]+j[0]), i[1]])
   return Z
def bounded_difference(setA, setB):
   Z = []
   for i, j in zip(setA, setB):
       Z.append([max(0, i[0]+j[0]-1), i[1]])
   return Z
# ---------- Visualization ----------
def visualize(setA, setB, result, title):
   plt.figure(figsize=(8, 5))
   plt.plot([x[1] for x in setA], [x[0] for x in setA], 'o-', label='Set A')
   plt.plot([x[1] for x in setB], [x[0] for x in setB], 's-', label='Set B')
   plt.plot([x[1] for x in result], [x[0] for x in result], '^-',
label='Result')
```

```
    plt.title(title)
    plt.xlabel("Elements")
    plt.ylabel("Membership Degree")
    plt.legend()
    plt.grid(True)
    plt.show()
# ---------- Sample Sets ----------
setA = [[0.1, 1], [0.3, 2], [0.6, 3], [0.8, 4], [1.0, 5]]
setB = [[0.2, 1], [0.4, 2], [0.7, 3], [0.5, 4], [0.9, 5]]
# ---------- Menu ----------
while True:
    print("\n--- Fuzzy Logic Operations Menu ---")
    print("1. Union")
    print("2. Intersection")
    print("3. Complement (of A)")
    print("4. Fuzzy Sum")
    print("5. Fuzzy Product")
    print("6. Bounded Sum")
    print("7. Bounded Difference")
    print("8. Exit")
    choice = input("Enter your choice (1-8): ")
    if choice == '1':
        result = union(setA, setB)
        visualize(setA, setB, result, "Fuzzy Union")
    elif choice == '2':
        result = intersection(setA, setB)
        visualize(setA, setB, result, "Fuzzy Intersection")
    elif choice == '3':
        result = complement(setA)
        visualize(setA, setB, result, "Complement of A")
    elif choice == '4':
        result = fuzzy_sum(setA, setB)
        visualize(setA, setB, result, "Fuzzy Sum")
    elif choice == '5':
        result = fuzzy_product(setA, setB)
        visualize(setA, setB, result, "Fuzzy Product")
    elif choice == '6':
        result = bounded_sum(setA, setB)
        visualize(setA, setB, result, "Bounded Sum")
    elif choice == '7':
        result = bounded_difference(setA, setB)
        visualize(setA, setB, result, "Bounded Difference")
    elif choice == '8':
        print("Exiting...")
        break
    else:
        print("Invalid choice! Try again.")
```

Exp5:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
# Step 2: Define hyperparameters
TIME_STEPS = 10      # Number of time steps in each sequence
FEATURES = 1         # Number of features per step
UNITS = 50           # LSTM units
EPOCHS = 20          # Training epochs
BATCH_SIZE = 8       # Batch size
LEARNING_RATE = 0.001
# Step 3: Create sample sequential data (sine wave)
x = np.linspace(0, 100, 500)
y = np.sin(x)
# Prepare data into sequences
X, Y = [], []
for i in range(len(y) - TIME_STEPS):
    X.append(y[i:i+TIME_STEPS])
    Y.append(y[i+TIME_STEPS])
X = np.array(X)
Y = np.array(Y)
# Reshape input for LSTM [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], FEATURES))
# Step 4: Create the LSTM model
model = Sequential([
    LSTM(UNITS, input_shape=(TIME_STEPS, FEATURES)),
    Dense(1)
])
# Step 5: Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
             loss='mse',
             metrics=['mae'])
# Step 6: Train the model
history = model.fit(X, Y, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1)
# Step 7: Evaluate model performance
loss, mae = model.evaluate(X, Y, verbose=0)
print(f"Model Evaluation — Loss: {loss:.4f}, MAE: {mae:.4f}")
# Step 8: Make predictions
predictions = model.predict(X)
# Optional — visualize predictions
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
plt.plot(Y, label='Actual')
plt.plot(predictions, label='Predicted')
plt.title("LSTM Prediction vs Actual (Sine Wave)")
plt.legend()
plt.show()
```

Exp6:

```
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.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Step 2: Load sample dataset (Iris dataset for simplicity)
from sklearn.datasets import load_iris
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
# Add some random missing values for demo
df.loc[5:10, 'sepal length (cm)'] = np.nan
# Step 3: Perform EDA (Exploratory Data Analysis)
print("\n🔹 Dataset Info:")
print(df.info())
print("\n🔹 First 5 Rows:")
print(df.head())
print("\n🔹 Summary Statistics:")
print(df.describe())
# Visualization examples
sns.pairplot(df, hue='target')
plt.show()
plt.figure(figsize=(8,4))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()
# Step 4: Handle Missing Values
print("\nMissing values before handling:")
print(df.isnull().sum())
# Fill missing values with column mean
df.fillna(df.mean(numeric_only=True), inplace=True)
print("\nMissing values after handling:")
print(df.isnull().sum())
# Step 5: Feature Scaling
X = df.drop('target', axis=1)
y = df['target']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 6: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
print(f"\nShapes -> X_train: {X_train.shape}, X_test: {X_test.shape}")
# Step 7: Prepare ANN Model (Optional demonstration)
model = Sequential([
   Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
   Dense(8, activation='relu'),
   Dense(3, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
print("\nTraining ANN model...")
history = model.fit(X_train, y_train, epochs=20, batch_size=8,
validation_split=0.2, verbose=1)
# Step 8: Evaluate Model
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"\n✅ Model Evaluation — Loss: {loss:.4f}, Accuracy: {acc:.4f}")
```

Exp7:

```python
# Step 1: Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt
# Step 2: Create a very small dataset manually
data = {
    'Age': [22, 25, 30, 35, 40, 45, 28, 50, 32, 27],
    'Income': [25000, 40000, 32000, 50000, 60000, 80000, 35000, 90000, 48000,
30000],
    'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female'],
    'Education': ['Bachelor', 'Master', 'High School', 'PhD', 'Bachelor',
'Master', 'PhD', 'Bachelor', 'Master', 'High School'],
    'Purchased': [0, 1, 0, 1, 1, 1, 0, 1, 1, 0]  # Target column
}
df = pd.DataFrame(data)
print("✅ Small Dataset Created Successfully!")
print(df)
# Step 3: Encode categorical data
encoder = LabelEncoder()
for col in ['Gender', 'Education']:
    df[col] = encoder.fit_transform(df[col])
# Step 4: Split features and target
X = df.drop('Purchased', axis=1)
y = df['Purchased']
# Step 5: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 6: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.3, random_state=42)
# Step 7: Define models
models = {
   "Logistic Regression": LogisticRegression(),
   "Random Forest": RandomForestClassifier(n_estimators=50, random_state=42),
   "AdaBoost": AdaBoostClassifier(n_estimators=50, random_state=42),
   "SVM": SVC(probability=True, random_state=42)
}
# Step 8: Train & evaluate each model
results = []
for name, model in models.items():
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   y_prob = model.predict_proba(X_test)[:, 1]
   acc = accuracy_score(y_test, y_pred)
   auc = roc_auc_score(y_test, y_prob)
   results.append([name, acc, auc])
   print(f"\n🔹 {name}: Accuracy = {acc:.3f}, AUC = {auc:.3f}")
# Step 9: Show results
results_df = pd.DataFrame(results, columns=["Model", "Accuracy", "AUC"])
print("\n📊 Model Performance Comparison:")
print(results_df)
# Step 10: Plot ROC Curves
plt.figure(figsize=(6,5))
for name, model in models.items():
   y_prob = model.predict_proba(X_test)[:, 1]
   fpr, tpr, _ = roc_curve(y_test, y_prob)
   plt.plot(fpr, tpr, label=name)
plt.plot([0,1], [0,1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Small Dataset Classifiers")
plt.legend()
plt.grid(True)
plt.show()
```

Exp8:

```
# Import libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Scale data (important for models like SVM & Logistic Regression)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Define models
models = {
   "Logistic Regression": LogisticRegression(),
   "Random Forest": RandomForestClassifier(),
   "SVM": SVC(),
   "AdaBoost": AdaBoostClassifier()
}
# Train and evaluate models
results = []
for name, model in models.items():
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   acc = accuracy_score(y_test, y_pred)
   prec = precision_score(y_test, y_pred, average='weighted')
   rec = recall_score(y_test, y_pred, average='weighted')
   f1 = f1_score(y_test, y_pred, average='weighted')
   results.append([name, acc, prec, rec, f1])
# Display results
results_df = pd.DataFrame(results, columns=["Model", "Accuracy", "Precision",
"Recall", "F1 Score"])
print(results_df)
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Create sine wave data
x = np.linspace(0, 100, 500)
y = np.sin(x)

# Prepare sequences (5 previous values → predict next)
X, Y = [], []
for i in range(len(y) - 5):
   X.append(y[i:i+5])
   Y.append(y[i+5])

X = np.array(X).reshape(-1, 5, 1)
Y = np.array(Y).reshape(-1, 1)

# RNN model
model = Sequential([
   SimpleRNN(16, activation='tanh', input_shape=(5, 1)),
   Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=20, verbose=0)

# Predictions
pred = model.predict(X)
plt.plot(Y, label='Actual')
plt.plot(pred, label='Predicted')
plt.title("RNN - Sine Wave Prediction")
plt.legend()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Create sine wave data
x = np.linspace(0, 100, 500)
y = np.sin(x)

# Prepare sequences (5 previous values → predict next)
X, Y = [], []
for i in range(len(y) - 5):
    X.append(y[i:i+5])
    Y.append(y[i+5])

X = np.array(X).reshape(-1, 5, 1)
Y = np.array(Y).reshape(-1, 1)

# LSTM model
model = Sequential([
    LSTM(16, activation='tanh', input_shape=(5, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=20, verbose=0)

# Predictions
pred = model.predict(X)
plt.plot(Y, label='Actual')
plt.plot(pred, label='Predicted')
plt.title("LSTM - Sine Wave Prediction")
plt.legend()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Create sine wave data
x = np.linspace(0, 100, 500)
y = np.sin(x)

# Prepare data (use 5 previous values → predict next)
X, Y = [], []
for i in range(len(y) - 5):
    X.append(y[i:i+5])
    Y.append(y[i+5])

X = np.array(X)
Y = np.array(Y)

# ANN model
model = Sequential([
    Dense(16, activation='relu', input_shape=(5,)),
    Dense(8, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=20, verbose=0)

# Predictions
pred = model.predict(X)
plt.plot(Y, label='Actual')
plt.plot(pred, label='Predicted')
plt.title("ANN - Sine Wave Prediction")
plt.legend()
plt.show()
```

```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
data = pd.read_csv("heart.csv")

# Split features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ---1️⃣ AdaBoost ---
ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)
y_pred_ada = ada.predict(X_test)

print("AdaBoost Accuracy:", accuracy_score(y_test, y_pred_ada))
print("AdaBoost Report:\n", classification_report(y_test, y_pred_ada))

# ---2️⃣ Random Forest ---
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Random Forest Report:\n", classification_report(y_test, y_pred_rf))
```

```
# Step 1: Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Step 2: Load your CSV dataset
data = pd.read_csv("your_dataset.csv")  # replace with your CSV file name
print(data.head())

# Step 3: Define X (features) and y (target)
# Suppose your target column is 'target' which contains 0s and 1s
X = data.drop('target', axis=1)
y = data['target']

# Step 4: Split into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Step 6: Predict on test data
y_pred = model.predict(X_test)

# Step 7: Evaluate performance
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
# ---- Import libraries ----
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# ---- Step 1: Load your CSV dataset ----
# Example: replace 'data.csv' with your file name
data = pd.read_csv('data.csv')

# See the first few rows
print(data.head())

# ---- Step 2: Select features (X) and target (y) ----
# Example: suppose your CSV has columns 'Experience' and 'Salary'
X = data[['Experience']]    # independent variable(s)
y = data['Salary']          # dependent variable

# ---- Step 3: Split data ----
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ---- Step 4: Create and train model ----
model = LinearRegression()
model.fit(X_train, y_train)

# ---- Step 5: Predict ----
y_pred = model.predict(X_test)

# ---- Step 6: Evaluate ----
print("Coefficient (slope):", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# ---- Step 7: Visualize ----
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted Line')
plt.title("Linear Regression (CSV Dataset)")
plt.xlabel("Experience")
plt.ylabel("Salary")
plt.legend()
plt.show()
```