Introduction to Artificial Intelligence And Machine Learning Lab

P. Laxmi Pavan Kumar

22765A4204

Machine Learning Experiments

Experiment -1

Aim:

To Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.

Description:

The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.

Dataset:

Weather conditions to play game or not.

Python Code:

```
import csv
num_attributes = 6
a = []
with open ('data.csv', 'r') as csv file:
   reader = csv.reader (csvfile)
   for row in reader:
      a.append (row)
      print(row)
  type(reader)
```

Output:

```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
a[0][:-1]
```

Output:

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

```
print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)
for j in range(0,num_attributes):
    hypothesis[j] = a[0][j]
hypothesis
```

Output:

The initial value of hypothesis:

['0', '0', '0', '0', '0', '0']

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

```
hypothesis = = a[0][:-1]
```

Output:

True

```
print("\n Find S: Finding a Maximally Specific Hypothesis\n")


for i in range(0,len(a)):
    if a[i][num_attributes]=='Yes':
        for j in range(0,num_attributes):
            print(a[i][j], end=' ')
            if a[i][j]!=hypothesis[j]:
                hypothesis[j]='?'
            else :
                hypothesis[j]= a[i][j]
    print("\n\nFor Training instance No:{} the hypothesis is ".format(i), hypothesis)


print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

Output:

Find S: Finding a Maximally Specific Hypothesis

Sunny Warm Normal Strong Warm Same

For Training instance No:0 the hypothesis is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

Sunny Warm High Strong Warm Same

For Training instance No:1 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

For Training instance No:2 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

Sunny Warm High Strong Cool Change

For Training instance No:3 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples :

Experiment-2

Aim:

To Implement and demonstrate CEA algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.

Description:

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example.

Dataset:

Weather conditions to play game or not.

Python Code:

```
import numpy as np

import pandas as pd

data = pd.read_csv(path+'/enjoysport.csv')

concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

print("\nTarget Values are: ",target)

def learn(concepts, target):

    specific_h = concepts[0].copy()

    print("\nInitialization of specific_h and genearal_h")

    print("\nSpecific Boundary: ", specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]

    print("\nGeneric Boundary: ",general_h)

for i, h in enumerate(concepts):

        print("\nInstance", i+1 , "is ", h)

        if target[i] == "yes":

            print("Instance is Positive ")

            for x in range(len(specific_h)):

                if h[x]!= specific_h[x]:

                    specific_h[x] ='?'

                    general_h[x][x] ='?'
```

```
        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")

['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Output:

Instances are:

[[‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’]

[‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’]

[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’]]

Target Values are: [‘yes’ ‘yes’ ‘no’ ‘yes’]

Initialization of specific_h and genearal_h

Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary: [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 1 is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’] Instance is Positive

Specific Boundary after 1 Instance is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 1 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 2 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’] Instance is Positive

Specific Boundary after 2 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 2 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 3 is [‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’] Instance is Negative

Specific Boundary after 3 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 3 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘same’]]

Instance 4 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’] Instance is Positive

Specific Boundary after 4 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]

Generic Boundary after 4 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Final Specific_h: [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]

Final General_h: [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’]]

Experiment-3

Aim:

Implement Linear and multi Linear Regression.

Description:

Linear Regression:

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Equation: $y= a0+a1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Multiple Linear Regression:.

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x1, x2, x3, ...,xn. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

Equation:

Y= b₀+b₁x₁+ b₂x₂+ b₃x₃+..... bnxn ..... (a)

Where,

Y= Output/Response variable

b0, b1, b2, b3 , bn....= Coefficients of the model.

x1, x2, x3, x4,...= Various Independent/feature variable.

R-squared (R2) score:

It measures the proportion of the variance in the dependent variable (y)that is explained by the independent variable(s) (x)in the model. R2 score ranges from 0 to 1, where a value of 1 indicates that the model explains all the variability in the dependent variable, and a value of 0 indicates that the model does not explain any variability in the dependent variable. The formula for R2 score is:

Equation: R2 =1-(SS_ res/SS_ tot)

where SS _ res is the sum of squared residuals (the difference between the predicted values and the actual values of y) and SS _tot is the total sum of squares (the difference between the actual values of y and the mean value of y).

Mean absolute error (MAE):

It is a measure of the average magnitude of the errors between the predicted values and the actual values of y. MAE is calculated as the average of the absolute differences between the predicted values and the actual values of y. The formula for MAE is:

Equation: $MAE=(1/n) *\Sigma|yi-\hat{y}i|$

where n is the number of observations, y is the actual value of y, and ŷi is the predicted value of y.

Dataset: Attendance No of certifications Marks.

Python Code:

Linear Regression:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
 from sklearn.metrics import r2_score
a=pd.read_csv("data,csv")
df=pd.DataFrame(a)
 print(df)
 x=df[['attendence']] y=df[["marks"]]
 print(y.head())
 print(x.head())
 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)
 model=LinearRegression()
 model.fit(x_train,y_train)
 y_predict=model.predict(x_test)
print(y_predict)
 r2=r2_score(y_test,y_predict)
 print(r2)
 print(model.predict([[62]]))
```

Output:

```
    Unnamed: 0  attendence  marks
0            1          70     80
1            2          71     81
2            3          72     82
3            4          73     83
4            5          74     84
5            6          75     85
6            7          76     86
7            8          77     87
8            9          78     88
9           10          79     89
10          11          80     90
11          12          81     91
12          13          82     92
13          14          83     93
14          15          84     94
15          16          85     95
16          17          86     96
17          18          87     97
18          19          88     98
19          20          89     99

   marks
0     80
1     81
2     82
3     83
4     84
   attendence
0          70
1          71
2          72
3          73
4          74

Y predicted values

[[86.]
 [99.]
 [95.]
 [88.]
 [80.]
 [84.]]
```

R2 score:- 1.0

Y predicted value :-

[[70.]]

Multi Linear Regression:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split
```

```
df=pd.read_csv(r"/content/Multi.csv")
df=pd.DataFrame(df)
print(df)
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
print(f'size of x: {x.shape} and y: {y.shape}')
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print(f'size of x_test: {x_test.shape} and x_train: {x_train.shape}')
print(f'size of y_test: {y_test.shape} and y_train: {y_train.shape}')
from sklearn.linear_model import LinearRegression
model=LinearRegression()
a=model.fit(x_train,y_train)
y_predict=a.predict(x_test)
from sklearn.metrics import r2_score,mean_squared_error
print(f'r2 score fit {r2_score(y_test,y_predict)}')
print(f'mean score error {mean_squared_error (y_test,y_predict)}')
a=pd.DataFrame({'Actual':y_test,'Predict':y_predict})
print(a)
```

Output:

	attandence	courses	backlogs	marks
0	71	3	0	80
1	72	2	0	75
2	73	1	0	70
3	74	0	2	40
4	75	4	0	90
5	76	2	0	76
6	77	0	2	43
7	78	1	1	55
8	79	4	0	91
9	80	2	0	76
10	81	3	0	80
11	82	2	0	76
12	83	1	2	43
13	84	0	3	34
14	85	4	0	95
15	86	3	0	87
16	87	2	0	76
17	88	1	2	51
18	89	3	0	98
19	90	0	0	85

sizeof x: (20,3) andy: (20,)

sizeof x_test:(4,3)andx_train:(16,3)

sizeof y_test:(4,)andy_train:(16,)

r2 score fit 0.9342747169332467

mean score error 28.24133256774555

Experiment – 4

Aim: Implement Polynomial Regression.

Description:

Polynomial regression is a type of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an n th degree polynomial.The aim of polynomial regression is to find the best-fit curve that represents the relationship between the variables.

Polynomial regression can be useful in situations where the relationship between the variables is not linear, but rather has a curved or nonlinear shape. It can be used in various fields such as finance, economics, physics, and engineering.

Dataset: Attendance No of certifications Marks.

Python Code:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

 from sklearn.linear_model import LinearRegression

 from sklearn.metrics import mean_squared_error,accuracy_score

 from sklearn.preprocessing import PolynomialFeatures

  data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality
/winequality-white.csv')

 X = data.drop(['quality'], axis=1)

y = data['quality']

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

  poly = PolynomialFeatures(degree=2, include_bias=False)

X_train_poly = poly.fit_transform(X_train)

X_test_poly = poly.transform(X_test)

model = LinearRegression()

 model.fit(X_train_poly, y_train)

y_pred = model.predict(X_test_poly)

 print("Accuracy:", accuracy_score(y_test,y_pred))
```

Output:

Accuracy: 85

Experiment – 5

Aim: Implement Logistic Regression.

Description:

Logistic Regression is a type of classification algorithm used to predict a binary outcome (i.e., a value of 0 or 1) based on one or more input variables. The algorithm models the probability of the binary outcome using a logistic function, which is a mathematical function that maps any input value to a probability between 0 and 1. The logistic function is an S-shaped curve that starts at 0 when the input is very negative, rises steeply in the middle, and levels off at 1 when the input is very positive.

The logistic regression algorithm works by fitting a line to the input variables that maximizes the likelihood of the observed outcomes. The line is called the decision boundary, and it separates the two classes (i.e., 0 and 1) in the input space. The algorithm then uses the decision boundary to predict the class of new input data.

$P(Y=1|X) = 1 / (1 + \exp(-z))$ where $z = b0 + b1X1 + b2X2 + ... + bn*Xn$

Here, b0 is the intercept term, and b1, b2, ..., bn are the coefficients associated with the input variables X1, X2, ..., Xn. The logistic function, $1 / (1 + \exp(-z))$, maps the linear combination of input variables to a probability between 0 and 1.

Python Code:

```
import pandas as pd
 import numpy as np
 from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
 from sklearn.linear_model import LogisticRegression
 from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
data = load_iris()
X = data['data']
 y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
 model = LogisticRegression()
model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
 sensitivity = recall_score(y_test, y_pred)
 cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
```

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("TP:", tp)
print("FP:", fp)
print("TN:", tn)
print("FN:", fn)
```

Output:-

```
Accuracy: 0.9777777777777777
Precision: 0.9809523809523809
Sensitivity: 0.9777777777777777
TP: 14
FP: 0
TN: 15
FN: 1
```

Experiment – 6

Aim: Implement Decision Tree-Regressor.

Description:

Decision tree is a supervised learning algorithm used for both classification and regression tasks. In decision tree regression, the algorithm creates a model by recursively splitting the data into smaller subsets based on the input features, until it reaches a stopping criterion, such as a maximum depth or a minimum number of samples per leaf. At each split, the algorithm chooses the feature and the value of that feature that result in the greatest reduction in the variance of the target variable (i.e., the sum of squared differences between the actual and predicted values of the target variable). This process creates a tree-like model where each node represents a split on a feature and each leaf node represents a predicted value.

Python Code:

```
import pandas as pd

from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error

data = pd.read_csv('attendance_marks.csv')

X = data.drop('marks', axis=1)

y = data['marks']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeRegressor(random_state=42)
```

Experiment – 7

Aim: Implement Decision Tree-Classifier.

Description:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

Python Code:

```
import pandas as pd
 from sklearn.tree import DecisionTreeClassifier
 from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix
from sklearn.datasets import load_iris
 iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
 y = pd.Series(iris.target)
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)
 print(f'Accuracy: {accuracy:.2f}')
 print(f'Precision: {precision:.2f}')
 print('Confusion Matrix:')
print(cm)
```

Output:

```
Accuracy: 1.00
Precision: 1.00
Confusion Matrix: [[10 0 0]
                   [ 0 9 0]
                   [ 0 0 11]]
```

Experiment – 8

Aim: Implement Random Forest-Regressor.

Description:

Random Forest is a popular machine learning algorithm used for both classification and regression tasks. It belongs to the family of ensemble methods, which combine multiple models to improve their accuracy and robustness. Random Forest is a type of decision tree ensemble method that uses multiple decision trees to make a prediction.

The basic idea behind Random Forest is to create a large number of decision trees and combine their predictions to make a final prediction. Each decision tree in the Random Forest is trained on a randomly selected subset of the training data, and at each node of the tree, a random subset of features is considered for splitting. This helps to reduce overfitting and improve the generalization performance of the model.

Python Code:

```
import pandas as pd
 from sklearn.ensemble import RandomForestRegressor
 from sklearn.model_selection import train_test_split
 from sklearn.metrics import r2_score
 from sklearn.datasets import load_boston
  boston = load_boston()
 X = pd.DataFrame(boston.data, columns=boston.feature_names)
 y = pd.Series(boston.target)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
  model = RandomForestRegressor(n_estimators=100, random_state=42)
print(f'Accuracy: {accuracy:.2f}')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = r2_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Output:-

```
Accuracy: 0.87
```

Experiment – 9

Aim: Implement Forest Classifier.

Description:

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

Python Code:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score

 iris = load_iris()

 X = iris.data

y = iris.target

 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

 model = RandomForestClassifier(n_estimators=100, random_state=42)

 model.fit(X_train, y_train)

 y_pred = model.predict(X_test)

 accuracy = accuracy_score(y_test, y_pred)

 precision = precision_score(y_test, y_pred, average='weighted')

 conf_mat = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

 print(f"Precision: {precision:.2f}")

print("Confusion Matrix:")

 print(conf_mat)
```

Output:

```
Accuracy: 1.00

Precision: 1.00

Confusion Matrix:

[[10 0 0]

 [ 0 9 0]

 [ 0 0 11]]
```

Data preprocessing and correlation

Data preprocessing and correlation analysis for a cricket dataset:

Python Code:

```
import pandas as pd
import numpy as np
import seaborn as sns import matplotlib.pyplot as plt
df = pd.read_csv("cricket.csv")
print(df.head())
print(df.isna().sum())
df.drop(["PLAYER", "Pos", "HS", "Avg", "100", "50"], axis=1, inplace=True)
df["Inns"] = pd.to_numeric(df["Inns"], errors="coerce")
df["Runs"] = pd.to_numeric(df["Runs"], errors="coerce")
df["BF"] = pd.to_numeric(df["BF"], errors="coerce")
df["SR"] = pd.to_numeric(df["SR"], errors="coerce")
print(df.isna().sum())
corr = df.corr()
sns.heatmap(corr, annot=True, cmap="YlGnBu")
print(corr)
```

Output:

```
          PLAYER    Span  Mat  Inns  NO   Runs     HS    Ave    BF      SR  100  50
0
0  SR Tendulkar  1989-2013  463   452  41  18426  200*  44.83  21367  86.23  49  96
20
1   RT Ponting  1995-2012  375   365  39  13704   164  42.03  17046  80.39  30  82
20
2    JH Kallis  1996-2014  328   314  53  11579  139*  44.36  15885  72.89  17  86
17
3    ST Jayasuriya  1989-2011  445   433  18  13430   189  32.36  14725  91.20  28
68  34
4  DPMD Jayawardene  1998-2015  448   418  39  12650  144*  33.37  16020  78.96
19  77   28
```

```
PLAYER    0
Span      0
Mat       0
Inns      0
NO        0
Runs      0
HS        0
Ave       0
BF        0
SR        0
100       0
50        0
0         0
dtype: int64
```

```
Span      0
Mat       0
Inns      6
NO        6
Runs      6
HS        6
Ave       6
BF        6
SR        6
100       6
50        6
0         0
dtype: int64
```

```
           Mat      Inns        NO      Runs        BF        SR       100
50         0

Mat   1.000000  0.996146  0.905562  0.910104  0.882194  0.468139  0.757196
0.765968 -0.181809

Inns  0.996146  1.000000  0.893116  0.897532  0.876428  0.473458  0.750869
0.758170 -0.180510

NO    0.905562  0.893
```

Artificial Intelligence Experiments

Experiment – 1

Aim: Implement Water Jug Problem.

Description:

A Water Jug Problem : You are given two jugs a 4-gallons one a 3- gallon one, a pump whichhas unlimited water you can use to fill the Jug, and the ground on which water may be poured. Neither jug has any measuring marking on it, How can you get exactly 2 gallons of water in the 4-gallon jug ?

State Representation and initial State

We will represent a state of the problem as a tuple (x,y) where x represents the amount of water in the 4-gallon jug and y represent the amount of water in the 3-gallon jug. Note 0<_x<_4 and 0<_y<_3.
Our initial state (0,0)

Python Code

```
Print('water Jug Problem')

X=int (input('enter x')

Y=int (input('enter y')

While True:

  rule=int(input('enter rule'))

  if rule==1:

    if x<4:

      x=4

 if rule==2:

    if y<3:

      y=3

 if rule==3:

    if x>0:

      x=0

 if rule==4:

    if y>0:

      y=0

 if rule==5:

    if  x+y>=4 and y>0:

      x, y =4, y-(4-x)

  if rule==6:
```

```
      if  x+y>=3 and x>0:
          x, y =x-(3-y),3
     if rule==7:
      if  x+y<=4 and y>0:
          x, y =x+y, 0
     if rule==8:
      if  x+y<=4 and x>0:
          x, y =0,x+y
print("x=" , x)
print("y=" , y)
if (x==2):
    print("goal reached")
     break
```

Output:

```
Water Jug Problem
Enter x =0
Enter y=0
Enter rule =2
X=0
Y=3
Enter rule =5
X=0
Y=3
Enter rue =1
X=4
Y=0
Enter rule =6
X=1
Y=3
Enter rule =4
X=1
Y= 0
```

Enter rule =8

X=0

Y=1

Enter rule =1

X=4

Y=1

Enter rule =6

X=2

Y=3

Goal reached

Java Code:

```
import java.util.Scanner;

public class WaterJugProblem {
    public static void main(String[] args) {
        System.out.println("Water Jug Problem");

        Scanner scanner = new Scanner(System.in);
        int x = 0, y = 0;

        while (true) {
            System.out.print("Enter rule: ");
            int rule = scanner.nextInt();

            if (rule == 1) {
                if (x < 4) {
                    x = 4;
                }
            } else if (rule == 2) {
                if (y < 3) {
                    y = 3;
                }
```

```
} else if (rule == 3) {
   if (x > 0) {
      x = 0;
   }
} else if (rule == 4) {
   if (y > 0) {
      y = 0;
   }
} else if (rule == 5) {
   if (x + y >= 4 && y > 0) {
      x = 4;
      y = y - (4 - x);
   }
} else if (rule == 6) {
   if (x + y >= 3 && x > 0) {
      x = x - (3 - y);
      y = 3;
   }
} else if (rule == 7) {
   if (x + y <= 4 && y > 0) {
      x = x + y;
      y = 0;
   }
} else if (rule == 8) {
   if (x + y <= 4 && x > 0) {
      x = 0;
      y = x + y;
   }
}

System.out.println("x = " + x);
System.out.println("y = " + y);
```

```
            if (x == 2) {
                System.out.println("Goal reached");
                break;
            }
        }
    }
}
```

Experiment – 2

Aim: Implement Tic tac-toe.

Python code:

```
import numpy as np
import random
from time import sleep
 def create_board():
    return(np.array([[0, 0, 0],
                     [0, 0, 0],
                     [0, 0, 0]]))
def possibilities(board):
    l = []
  for i in range(len(board)):
        for j in range(len(board)):
          if board[i][j] == 0:
              l.append((i, j))
    return(l)
def random_place(board, player):
    selection = possibilities(board)
    current_loc = random.choice(selection)
    board[current_loc] = player
    return(board)

def row_win(board, player):
    for x in range(len(board)):
        win = True
for y in range(len(board)):
            if board[x, y] != player:
                win = False
                continue
   if win == True:
```

```
            return(win)
    return(win)

def col_win(board, player):
    for x in range(len(board)):
        win = True
for y in range(len(board)):
            if board[y][x] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)

def diag_win(board, player):
    win = True
    y = 0
    for x in range(len(board)):
        if board[x, x] != player:
            win = False
    if win:
        return win
    win = True
    if win:
        for x in range(len(board)):
            y = len(board) - 1 - x
            if board[x, y] != player:
                win = False
    return win

def evaluate(board):
```

```
  winner = 0

    for player in [1, 2]:
        if (row_win(board, player) or
                col_win(board, player) or
                diag_win(board, player)):

            winner = player

    if np.all(board != 0) and winner == 0:
        winner = -1
    return winner

def play_game():
    board, winner, counter = create_board(), 0, 1
    print(board)
    sleep(2)

    while winner == 0:
        for player in [1, 2]:
            board = random_place(board, player)
            print("Board after " + str(counter) + " move")
            print(board)
            sleep(2)
            counter += 1
            winner = evaluate(board)
            if winner != 0:
                break
    return(winner)

print("Winner is: " + str(play_game()))
```

Output:

[[0 0 0]

[0 0 0]

[0 0 0]]

Board after 1 move

[[0 0 0]

[0 0 0]

[1 0 0]]

Board after 2 move

[[0 0 0]

[0 2 0]

[1 0 0]]

Board after 3 move

[[0 1 0]

[0 2 0]

[1 0 0]]

Board after 4 move

[[0 1 0]

[2 2 0]

[1 0 0]]

Board after 5 move

[[1 1 0]

[2 2 0]

[1 0 0]]

Board after 6 move

[[1 1 0]

[2 2 0]

[1 2 0]]

Board after 7 move

[[1 1 0]

[2 2 0]

[1 2 1]]

Board after 8 move

[[1 1 0]

[2 2 2]

[1 2 1]]

Winner is: 2

Experiment – 3

Aim: Implement Prolog.

Code:

```
got(devi,first).
went(devi,kulumanali).
went(rahul,kulumanali).
happy(rahul):-
    got(rahul,first);
    went(rahul,kulumanali).
```

Output:

```
?-happy[devi].
True
?-happy[Rahul].
False
?-trace.
True
[trace]?- happy(devi).
Call:(10) happy(devi) ?creep
Call:(11) happy(devi,first) ?creep
Fail: (11)got (devi,first) ?creep
Fail: (10) happy (devi)? Creep
False
```

Experiment – 4

Aim: Implement Monkey Banana Problem.

Python Code:

```
on(floor,monkey).
on(floor,box).
in(room,monkey).
in(room,box).
at(ceiling,banana).
strong(monkey).
grasp(monkey).
climb(monkey,box).
push(monkey,box):-
        strong(monkey).
under(banana,box):-
        push(monkey,box).
canreach(banana,monkey):-
        at(floor,banana);
    at(ceiling,banana),
    under(banana,box),
    climb(monkey,box).
canget(banana,monkey):-
        canreach(banana,monkey),grasp(monkey).
```

Output:

```
?-['D:/monkey.pl']
?-Trace
[trace] ?-canreach (banana,monkey).
Call: (10) canreach (banana,monkey) ?No previous search
Call: (10) canreach (banana,monkey) ? creep
Call: (11) at floor(floor,banana) ? No previous search
Call: (11) at floor(floor,banana) ?creep
fail: (11) at floor(floor,banana) ?creep
redo : (10) canreach (banana,monkey) ? creep
```

Call: (11) at (ceiling,banana) ?creep

Exist: (11) at (ceiling,banana) ?creep

Call: (11) under (banana,Box) ?creep

Call: (12) push (banana,Box) ?creep

Call: (13) strong(Monkey) ?creep

Exist: (13) strong(Monkey) ?creep

Exist: (12) push (banana,Box) ?creep

Exist: (11) under (banana,Box) ?creep

Call: (11) climb(monkey,box)?creep

Exist: (11) climb(monkey,box)?creep

Exist : (10) canreach (banana,monkey) ? creep