Assignment-1

1. **Write a Python Program to Solve N-Queen Problem without using Recursion.**
   Input :

```
def isSafe(row, col, board):
    for r in range(row):
        c = board[r]
        if c == col or abs(c - col) == abs(r - row):
            return False
    return True

def nQueen(n):
    board = [-1] * n
    row = 0
    col = 0
    while row < n:
        while col < n:
            if isSafe(row, col, board):
                board[row] = col
                row += 1
                col = 0
                break
            else:
                col += 1
        else:
            row -= 1
            if row < 0:
                return [-1]
            col = board[row] + 1
            board[row] = -1
    return [x + 1 for x in board]

n = 4
ans = nQueen(n)
print(" ".join(map(str, ans)))
```

   Output :

```
priyankayadav@priyanka aimllab % python nqueen1.py
2 4 1 3
```

2. **Write a Python Program to implement the Backtracking approach to solve N Queen's problem.**
   Input :

```
def isSafe(board, row, col, n):
    for i in range(row):
        if board[i] == col or abs(board[i] - col) == abs(i -
row):
            return False
    return True

def solveNQueen(row, board, n):
    if row == n:
        return True

    for col in range(n):
        if isSafe(board, row, col, n):
            board[row] = col
            if solveNQueen(row + 1, board, n):
                return True
            board[row] = -1
    return False

def nQueen(n):
    board = [-1] * n
    if solveNQueen(0, board, n):
        return [x + 1 for x in board]
    else:
        return [-1]
n = 4
ans = nQueen(n)
print(" ".join(map(str, ans)))
```

Output :

```
priyankayadav@priyanka aimllab % python nqueen2.py
2 4 1 3
```

3. **Write a Python Program to implement Min-Max Algorithm.**

Input :

```
def minmax(depth, nodeIndex, isMax, scores, h):
    if depth == h:
        return scores[nodeIndex]

    if isMax:
        return max(
            minmax(depth + 1, nodeIndex * 2, False, scores,
h),
            minmax(depth + 1, nodeIndex * 2 + 1, False,
scores, h)
        )
    else:
        return min(
            minmax(depth + 1, nodeIndex * 2, True, scores,
h),
            minmax(depth + 1, nodeIndex * 2 + 1, True,
scores, h)
        )

scores = [3, 5, 6, 9, 1, 2, 0, -1]
h = 3
optimal_value = minmax(0, 0, True, scores, h)
print("The optimal value is:", optimal_value)
```

Output :

```
priyankayadav@priyanka aimllab % python minmax3.py
The optimal value is:5
```

4. **Write a Python Program to implement Alpha-Beta Pruning Algorithm.**

Input :

```
def alphabeta(depth, nodeIndex, isMax, values, alpha, beta,
h):
    if depth == h:
        return values[nodeIndex]

    if isMax:
        best = float('-inf')
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i,
False, values, alpha, beta, h)
```

```
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = float('inf')
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i,
True, values, alpha, beta, h)
            best = min(best, val)
            beta = min(beta, best)

            if beta <= alpha:
                break
        return best
values = [3, 5, 6, 9, 1, 2, 0, -1]
h = 3

alpha = float('-inf')
beta = float('inf')

optimal_value = alphabeta(0, 0, True, values, alpha, beta,
h)
print("The optimal value is:", optimal_value)
```

Output :

```
priyankayadav@priyanka aimllab % python alphaBeta4.py
The optimal value is: 3
```

NQueen without recursion

```
def isSafe(row, col, board):
    for r in range(row):
        c = board[r]
        if c == col or abs(c - col) == abs(r - row):
            return False
    return True

def nQueen(n):
    board = [-1] * n
    row = 0
    col = 0
    while row < n:
        while col < n:
            if isSafe(row, col, board):
                board[row] = col
                row += 1
                col = 0
                break
            else:
                col += 1
        else:
            row -= 1
            if row < 0:
                return [-1]
            col = board[row] + 1
            board[row] = -1
    return [x + 1 for x in board]

n = 4
ans = nQueen(n)
print(" ".join(map(str, ans)))
```

O/p — 2 4 1 3

NQueen with backtracking

```
def isSafe(board, row, col, n):
    # Check if there's a queen in the same column or diagonals
    for i in range(row):
        if board[i] == col or abs(board[i] - col) == abs(i - row):
            return False
    return True

def solveNQueen(row, board, n):
    if row == n:
        return True  # All queens placed
    for col in range(n):
        if isSafe(board, row, col, n):
            board[row] = col  # Place queen
            if solveNQueen(row + 1, board, n):
                return True  # If solution found, return
            board[row] = -1  # Backtrack
    return False  # No valid position in this row

def nQueen(n):
    board = [-1] * n  # board[i] = column position of queen in row i
    if solveNQueen(0, board, n):
        return [x + 1 for x in board]  # Convert to 1-based index
    else:
        return [-1]

# Example usage
n = 4
ans = nQueen(n)
print(" ".join(map(str, ans)))
```

O/p — 2 4 1 3

Min max

```
def minmax(depth, nodeIndex, isMax, scores, h):
    # Base case: leaf node reached
    if depth == h:
        return scores[nodeIndex]

    if isMax:
        return max(
            minmax(depth + 1, nodeIndex * 2, False, scores, h),
            minmax(depth + 1, nodeIndex * 2 + 1, False, scores, h)
        )
    else:
        return min(
            minmax(depth + 1, nodeIndex * 2, True, scores, h),
            minmax(depth + 1, nodeIndex * 2 + 1, True, scores, h)
        )

# Example leaf nodes of the game tree (assumed)
scores = [3, 5, 6, 9, 1, 2, 0, -1]
```

```python
h = 3  # height of tree = log2(len(scores))

# Call minmax for root node (0), depth 0, and maximizing player True
optimal_value = minmax(0, 0, True, scores, h)
print("The optimal value is:", optimal_value)
```

Optimal value = 5

Alpha beta

```python
def alphabeta(depth, nodeIndex, isMax, values, alpha, beta, h):
    # Base case: if it's a leaf node
    if depth == h:
        return values[nodeIndex]

    if isMax:
        best = float('-inf')
        # Recur for left and right children
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i, False,
values, alpha, beta, h)
            best = max(best, val)
            alpha = max(alpha, best)

            # Alpha Beta Pruning
            if beta <= alpha:
                break
        return best
    else:
        best = float('inf')
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i, True, values,
alpha, beta, h)
            best = min(best, val)
            beta = min(beta, best)

            # Alpha Beta Pruning
            if beta <= alpha:
                break
        return best

# Example leaf nodes
```

```python
values = [3, 5, 6, 9, 1, 2, 0, -1]
h = 3  # Height of the game tree: log2(len(values))

# Initial values of alpha and beta
alpha = float('-inf')
beta = float('inf')

# Call alphabeta for root node (0), depth 0, maximizing player True
optimal_value = alphabeta(0, 0, True, values, alpha, beta, h)
print("The optimal value is:", optimal_value)
```

Optimal value = 5

```python
graph = {
   '5' : ['3', '7'],
   '3' : ['2', '4'],
   '7' : ['8'],
   '2' : [],
   '4' : ['8'],
   '8' : []
}

visited = []
queue = []

def bfs(graph, start):
   visited.append(start)
   queue.append(start)
   while queue:
       m = queue.pop(0)
       print(m, end=" ")
       for i in graph[m]:
           if i not in visited:
               visited.append(i)
               queue.append(i)

bfs(graph, '5')
```

```python
graph = {
   '5' : ['3', '7'],
   '3' : ['2', '4'],
   '7' : ['8'],
   '2' : [],
   '4' : ['8'],
   '8' : []
}

visited = []
stack = []

def dfs(graph, start):
   visited.append(start)
```

```
    stack.append(start)
    while stack:
        m = stack.pop()
        print(m, end=" ")
        for i in reversed(graph[m]):
            if i not in visited:
                visited.append(i)
                stack.append(i)

dfs(graph, '5')
```

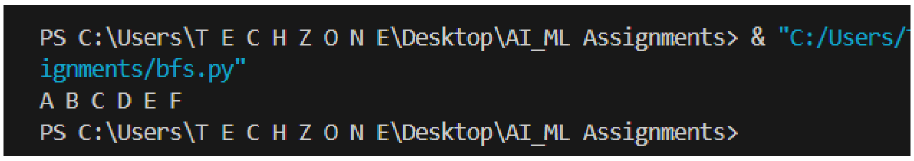
Assignment-2

1. Write a Python Program to implement Breadth First Search

Code:

```python
def bfs(graph, start):
    visited = []
    queue = [start]

    while queue:
        node = queue.pop(0)
        if node not in visited:
            print(node, end=" ")
            visited.append(node)
            queue.extend(graph[node])

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

bfs(graph, 'A')
```

Output:



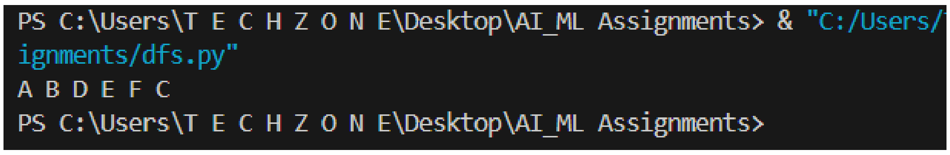
```
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments> & "C:/Users/
ignments/bfs.py"
A B C D E F
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments>
```

2. Write a Python Program to implement Depth First Search

Code:

```python
def dfs(graph, start, visited=[]):
    print(start, end=" ")
    visited.append(start)
    for node in graph[start]:
        if node not in visited:
            dfs(graph, node, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

dfs(graph, 'A')
```

Output:



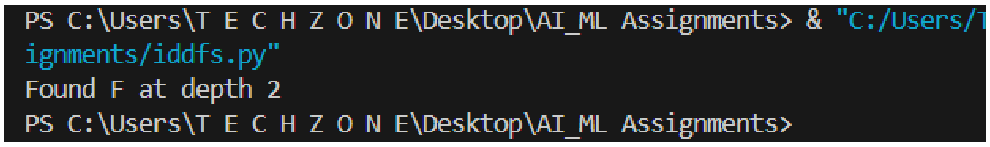
```
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments> & "C:/Users/
ignments/dfs.py"
A B D E F C
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments>
```

3. Write a Python Program to implement Iterative Deepening Depth First search (IDDFS)

Code:

```python
def dls(graph, node, target, depth):
    if node == target:
        return True
    if depth == 0:
        return False
    for n in graph[node]:
        if dls(graph, n, target, depth-1):
            return True
    return False

def iddfs(graph, start, target, max_depth):
    for depth in range(max_depth):
        if dls(graph, start, target, depth):
            print("Found", target, "at depth", depth)
            return
    print("Not Found")

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

iddfs(graph, 'A', 'F', 5)
```

Output:



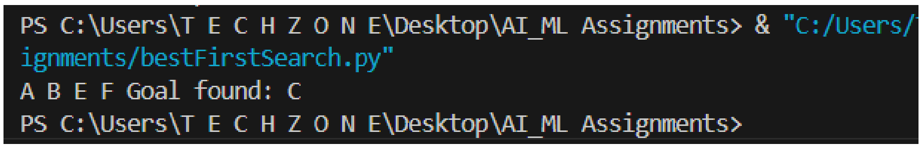
```
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments> & "C:/Users/T
ignments/iddfs.py"
Found F at depth 2
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments>
```

4. Write a Python Program to implement Best First Search

Code:

```python
def best_first_search(graph, start, goal, h):
    visited = []
    nodes = [start]

    while nodes:
        nodes.sort(key=lambda x: h[x])
        node = nodes.pop(0)
        if node == goal:
            print("Goal found:", node)
            return
        if node not in visited:
            print(node, end=" ")
            visited.append(node)
            nodes.extend(graph[node])

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

heuristic = {'A': 3, 'B': 2, 'C': 4, 'D': 6, 'E': 1, 'F': 0}

best_first_search(graph, 'A', 'C', heuristic)
```

Output:



```
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments> & "C:/Users/
ignments/bestFirstSearch.py"
A B E F Goal found: C
PS C:\Users\T E C H Z O N E\Desktop\AI_ML Assignments>
```

1. Write a Python Program to implement A* Algorithm

Input :

```python
import heapq

def astar(start, goal, graph, h):
   open_list = []
   heapq.heappush(open_list, (0, start))
   g = {start: 0}
   parent = {start: None}
   while open_list:
       f, node = heapq.heappop(open_list)
       if node == goal:
           path = []
           while node is not None:
               path.append(node)
               node = parent[node]
           return path[::-1]
       for n, cost in graph[node]:
           new_g = g[node] + cost
           if n not in g or new_g < g[n]:
               g[n] = new_g
               f = new_g + h[n]
               heapq.heappush(open_list, (f, n))
               parent[n] = node

graph = {
   'A':[('B',1),('C',3)],
   'B':[('D',3),('E',1)],
   'C':[('F',5)],
   'D':[],
   'E':[('G',2)],
   'F':[('G',2)],
   'G':[]
}
h = {'A':7,'B':6,'C':5,'D':4,'E':3,'F':2,'G':0}
print(astar('A','G',graph,h))
```

Output :

```
Ubuntu@Ubuntu assignment 3 % python3 Q1.py
['A', 'B', 'E', 'G']
```

2. Write a Python Program to implement AO* Algorithm

Input :

```python
graph = {
   'A':[['B','C']],
   'B':[['D','E']],
   'C':[['F','G']],
   'D':[],
   'E':[],
   'F':[],
   'G':[]
}
h = {'A':10,'B':6,'C':4,'D':3,'E':2,'F':1,'G':0}
solved = {}

def aostar(node):
   if node in solved:
       return h[node]
   if not graph[node]:
       solved[node] = True
       return h[node]
   min_cost = float('inf')
   for c in graph[node]:
       cost = sum(aostar(x) for x in c)
       if cost < min_cost:
           min_cost = cost
   h[node] = min_cost
   solved[node] = True
   return h[node]

print(aostar('A'))
```

Output :

```
Ubuntu@Ubuntu assignment 3 % python3 Q2.py
6
```

3. Write a Python Program to implement for IDA* (Iterative Deepening A*) Algorithm

Input :

```python
def ida_star(start, goal, graph, h):
   def search(path, g, bound):
       node = path[-1]
       f = g + h[node]
       if f > bound:
           return f
       if node == goal:
           return path
       m = float('inf')
       for n, cost in graph[node]:
           if n not in path:
               t = search(path+[n], g+cost, bound)
               if isinstance(t, list):
                   return t
               if t < m:
                   m = t
       return m
   bound = h[start]
   path = [start]
   while True:
       t = search(path, 0, bound)
       if isinstance(t, list):
           return t
       if t == float('inf'):
           return None
       bound = t

graph = {
   'A':[('B',1),('C',4)],
   'B':[('D',2),('E',5)],
   'C':[('F',3)],
   'D':[],
   'E':[('G',2)],
   'F':[('G',2)],
   'G':[]
}
h = {'A':7,'B':6,'C':4,'D':3,'E':2,'F':1,'G':0}
print(ida_star('A','G',graph,h))
```

Output :

```
Ubuntu@Ubuntu assignment 3 % python3 Q3.py
['A', 'B', 'E', 'G']
```

1. Write a Python Program to implement K-Nearest Neighbor Algorithm for data classification, choose dataset of your own choice.

Input :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_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)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
print("KNN Accuracy:", accuracy_score(y_test, y_pred))
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-3.py
KNN Accuracy: 1.0
```

2. Write a Python Program to implement Naïve Bayes Algorithm for data classification, choose dataset of your own choice.

Input :

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_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)

nb = GaussianNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)
print("Naïve Bayes Accuracy:", accuracy_score(y_test, y_pred))
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-3-qs2.py
Naïve Bayes Accuracy: 1.0
```

3. Write a Python Program to implement Decision Trees for data classification, choose data set of your own choice.

Input :

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_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)

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred))
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-3-qs3.py
Decision Tree Accuracy: 1.0
```

4. Write a Python Program to implement Logistic Regression for data classification, choose a dataset of your own choice.

Input :

```python
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

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)

lr = LogisticRegression(max_iter=200)
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-3-qs4.py
Logistic Regression Accuracy: 1.0
```

1. Write a Python Program to implement Support Vector Machines for data classification, choose dataset of your own choice.

Input :

```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

model = SVC(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"SVM Accuracy: {acc:.2f}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-5-qs1.py
SVM Accuracy: 1.00
```

2. Write a Python Program to implement Linear regression on a dataset of your own choice.
Input :

```python
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

data = load_diabetes()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

r2 = r2_score(y_test, y_pred)
print(f"Linear Regression R² Score: {r2:.2f}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-5-qs2.py
Linear Regression R² Score: 0.45
```

3. Write a Python Program to implement Polynomial Regression on a dataset of your own choice.

Input :

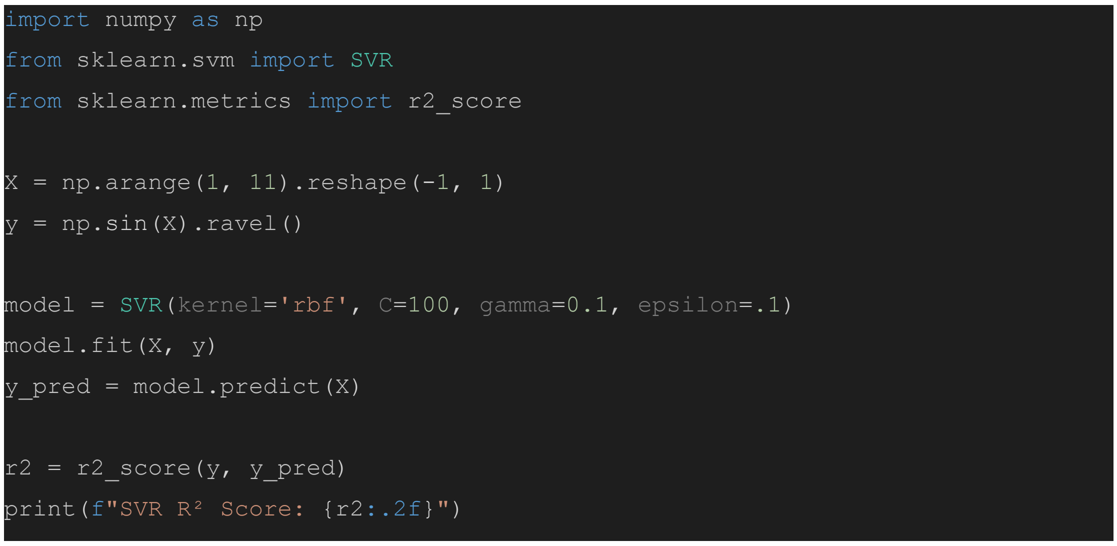
```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

X = np.arange(1, 11).reshape(-1, 1)
y = 3 * X.squeeze()**2 + 2 * X.squeeze() + 1

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)

r2 = r2_score(y, y_pred)
print(f"Polynomial Regression R² Score: {r2:.2f}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-5-qs3.py
Polynomial Regression R² Score: 1.00
```

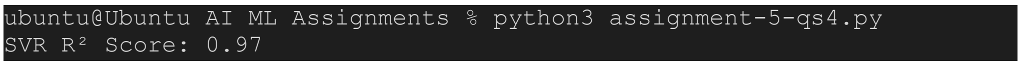
4. Write a Python Program to implement Support Vector Regression on a dataset of your own choice.

Input :



```python
import numpy as np
from sklearn.svm import SVR
from sklearn.metrics import r2_score

X = np.arange(1, 11).reshape(-1, 1)
y = np.sin(X).ravel()

model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
model.fit(X, y)
y_pred = model.predict(X)

r2 = r2_score(y, y_pred)
print(f"SVR R² Score: {r2:.2f}")
```

Output :



```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-5-qs4.py
SVR R² Score: 0.97
```

5. Write a Python Program to implement Artificial Neural Network for data classification, choose dataset of your own choice.

Input :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

model = MLPClassifier(hidden_layer_sizes=(8, 4), max_iter=1000,
random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"ANN Accuracy: {acc:.2f}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-5-qs5.py
ANN Accuracy: 0.98
```

6. Write a Python Program to implement Feed Forward Neural Network on a given dataset for data classification, choose dataset of your own choice.
Input :

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

model = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu',
solver='adam', max_iter=1000, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"Feed Forward NN Accuracy: {acc:.2f}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-5-qs6.py
Feed Forward NN Accuracy: 0.98
```

1. Write a Python Program to implement Principal Component Analysis on a dataset of your own choice.

Input :

```python
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

data = load_iris()
X = data.data

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

print(f"PCA Explained Variance Ratio: {pca.explained_variance_ratio_}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-6-qs1.py
PCA Explained Variance Ratio: [0.92461872 0.05306648]
```

2. Write a Python Program to implement Linear Discriminant Analysis on a dataset of your own choice.

Input :

```
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

y_pred = lda.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print(f"LDA Accuracy: {acc:.2f}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-6-qs2.py
LDA Accuracy: 1.00
```

3. Write a Python Program to implement Apriori Algorithm on a dataset of your own choice.
Input :

```
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

dataset = [
   ['milk', 'bread', 'eggs'],
   ['milk', 'bread'],
   ['milk', 'eggs'],
   ['bread', 'butter'],
   ['milk', 'bread', 'butter']
]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
import pandas as pd
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)

print("Apriori Frequent Itemsets:")
print(frequent_itemsets)
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-6-qs3.py
Apriori Frequent Itemsets:
     support      itemsets
0       0.8         (milk)
1       0.8        (bread)
2       0.4         (eggs)
3       0.4       (butter)
4       0.6  (milk, bread)
```

4. Write a Python Program to implement FP tree growth Algorithm on a dataset of your own choice.

Input :

```python
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

dataset = [
   ['milk', 'bread', 'eggs'],
   ['milk', 'bread'],
   ['milk', 'eggs'],
   ['bread', 'butter'],
   ['milk', 'bread', 'butter']
]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(df, min_support=0.4, use_colnames=True)
print("FP-Growth Frequent Itemsets:")
print(frequent_itemsets)
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-6-qs4.py
FP-Growth Frequent Itemsets:
     support      itemsets
0      0.8         (milk)
1      0.8        (bread)
2      0.4         (eggs)
3      0.4       (butter)
4      0.6  (milk, bread)
```

5. Write a Python Program to implement K-Means Algorithm on a dataset of your own choice.

Input :

```python
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

iris = load_iris()
X = iris.data

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(X)

print(f"K-Means Cluster Centers:\n{kmeans.cluster_centers_}")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-6-qs5.py
K-Means Cluster Centers:
[[5.006 3.418 1.464 0.244]
[6.853 3.073 5.742 2.071]
[5.883 2.741 4.393 1.433]]
```

6. Write a Python Program to implement DBSCAN Algorithm on a dataset of your own choice.

Input :

```python
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

X, _ = make_moons(n_samples=200, noise=0.05, random_state=42)

dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)

unique_clusters = len(set(labels)) - (1 if -1 in labels else 0)
print(f"DBSCAN found {unique_clusters} clusters (excluding noise).")
```

Output :

```
ubuntu@Ubuntu AI ML Assignments % python3 assignment-6-qs6.py
DBSCAN found 2 clusters (excluding noise).
```