LP-III
DAA:

1. Fibonacci

```
#include <bits/stdc++.h>
using namespace std;

class Fibonacci{
   public:
      int n;

      Fibonacci(int n){
         this->n=n;
      }

      int recursive(int i){
         if(i<=1){
            return i;
         }
         return recursive(i-1) + recursive(i-2);
      }

      void iterative(){
         int n1=0;
         int n2=1;
         int num;
         cout<<n1<<" "<<n2<<" ";
         for(int i=2;i<n;i++){
            num=n1+n2;
            n1=n2;
            n2=num;
            cout<<num<<" ";
         }
      }
};

      int main(){
         int n;
         cout<<"Enter N:";
         cin>>n;
         Fibonacci fb(n); // initialized
         cout<<"Iterative: ";
         fb.iterative();
         cout<<endl;
         cout<<"Recursive: ";
      for (int i = 0; i < n; i++)
  {
    cout << fb.recursive(i) << " ";
  }
  return 0;
}
```

2. Huffman coding

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

```
import heapq
from collections import Counter, defaultdict

class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right

    def __lt__(self, other):
        return self.freq < other.freq

def huffman_encoding(data):
    # Step 1: Calculate frequency of each character
    frequency = Counter(data)

    # Step 2: Build priority queue (min-heap)
    heap = [Node(freq, symbol) for symbol, freq in frequency.items()]
    heapq.heapify(heap)

    # Step 3: Build Huffman Tree
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.freq + right.freq, None, left, right)
        heapq.heappush(heap, merged)
    root = heap[0]

    # Step 4: Generate Huffman Codes
    huffman_codes = {}
    def generate_codes(node, current_code=""):
        if node is None:
            return
        if node.symbol is not None:  # Leaf node
            huffman_codes[node.symbol] = current_code
            return
        generate_codes(node.left, current_code + "0")
        generate_codes(node.right, current_code + "1")

    generate_codes(root)

    # Step 5: Encode data
    encoded_data = ''.join(huffman_codes[char] for char in data)
    return encoded_data, huffman_codes

# Example usage
data = "simple huffman example"
encoded_data, huffman_codes = huffman_encoding(data)
print("Huffman Codes:", huffman_codes)
print("Encoded Data:", encoded_data)
```

3. Knapsack

```
#include<bits/stdc++.h>
using namespace std;
int func(int idx, int w, vector<int> &values, vector<int> &weights, vector<vector<int>> &dp)
{
    if (idx == 0)
    {
        if (weights[0] <= w)
        {
            return values[0];
        }
        else
        {
            return 0;
        }
    }
    if (dp[idx][w] != -1)
    {
        return dp[idx][w];
    }
    int notTake = func(idx - 1, w, values, weights, dp);
    int take = INT_MIN;
    if (weights[idx] <= w)
    {
        take = values[idx] + func(idx - 1, w - weights[idx], values, weights, dp);
    }
    return dp[idx][w] = max(take, notTake);
}
int maxProfit(vector<int> &values, vector<int> &weights, int n, int w)
{
    // Write your code here
    vector<vector<int>> dp(n, vector<int>(w + 1, -1));
    return func(n - 1, w, values, weights, dp);
}

int main(){
    int n,w;
    cin >> n >> w;
    vector<int> values(n);
    vector<int> weights(n);
    for (int i = 0; i < n;i++){
        cin >> values[i];
    }
    for (int i = 0; i < n;i++){
        cin >> weights[i];
    }
    //cout << "Using Memoization: " << endl;
    cout << maxProfit(values,weights,n,w) << endl;
    // cout<< "Using Tabulation: " << endl;
    cout << tabulation(values, weights, n, w) << endl;

    return 0;
}

/*
    Input ->
    N = 5
    W = 100
    Values = {12, 35, 41, 25, 32}
    Weights = {20, 24, 36, 40, 42}

    Ouptut -> 101
*/

/*
    Complexity Analysis ->
    Memoization -> T.C -> O(N*W)
                   S.C -> O(N*W) + O(N)
    Tabulation  -> T.C -> O(N*W)
                   S.C -> O(N*W)
*/
```

4. N-Queen

```
#include <bits/stdc++.h>
#define rep(i,a,b)  for(int i=a;i<b;i++)

bool issafe(int** board, int x, int y, int n)
{
        rep(i,0,n){
                if(board[i][y]==1)
                        return false;
        }

        int row=x;
        int col= y;
        while(row>= 0 && col>= 0){
                if(board[row][col]==1)
                        return false;
                row--;
                col--;
        }

        row= x;
        col= y;
        while(row>= 0 && col< n){
                if(board[row][col]==1)
                        return false;
                row--;
                col++;
        }
        return true;
}

bool nqueen(int** arr, int x, int n){
        if(x>=n)
                return true;

        for(int col=0; col<n;col++){
                if(issafe(arr,x,col,n)){
                        arr[x][col] = 1;
                        if(nqueen(arr,x+1,n)){

                                return true;
                        }
                        arr[x][col] = 0;
                }
        }
        return false;
}

int main(){
        int n;
        std::cin>>n;
        int** board=new int*[n];

        rep(i,0,n){
                board[i] = new int[n];
                rep(j,0,n){
                        board[i][j]=0;
                }
        }

        if(nqueen(board,0,n))
        rep(i,0,n){
                rep(j,0,n){
                        std::cout<<board[i][j]<<" ";
                }
                std::cout<<"\n";
        }
}
```

---

5. Quick Sort

```
#include <iostream>
#include <vector>

int partition(std::vector<int>& arr, int low, int high) {
    int pivot = arr[high];    // Choose the last element as the pivot
    int i = low;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            std::swap(arr[i], arr[j]);
            i++;
        }
    }
    std::swap(arr[i], arr[high]);
    return i;
}

void quickSort(std::vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int main() {
    std::vector<int> arr = {10, 7, 8, 9, 1, 5};
    int n = arr.size();

    quickSort(arr, 0, n - 1);

    std::cout << "Sorted array: ";
    for (int x : arr) {
        std::cout << x << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

1. Uber

Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression and random forest regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE, etc.

Handle null values

In [38]:

```
# check for null values in the data-frame
ds.isna().sum()

# dropoff_longitude and dropoff_latitude have a single null value
# remove those rows from the data-frame
ds.dropna(inplace=True)
```

Outlier Analysis

In [42]:

```
# remove outlier(s) where passenger_count > 100
sns.scatterplot(ds, y="fare_amount", x="passenger_count")
ds = ds[ds["passenger_count"] < 150]


# remove outliers from pickup/dropoff locations
def remove_outliers(feature):
    global ds
    q3 , q1 = np.percentile( ds[feature] , [ 75 , 25 ] )
    iqr = q3 - q1
    ds = ds[ (ds[feature] >= q1 - 1.5 * iqr) & (ds[feature] <= q3 + 1.5 *
iqr) ]
remove_outliers("pickup_latitude")
remove_outliers("pickup_longitude")
remove_outliers("dropoff_latitude")
remove_outliers("dropoff_longitude")
```

Correlation Analysis

In [44]:

```
ds.corr(method="pearson")
```

Model Fitting

Linear Regression

In [45]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# train-test split
X = ds.drop('fare_amount', axis=1)
y = ds['fare_amount']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# standardize the splits
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# We cannot use the fit() method on the test dataset, because
# it could introduce bias to the testing dataset. So, we apply the transform()
# method directly on the test dataset.
X_test_scaled = scaler.transform(X_test)
```

In [46]:

```
from sklearn.linear_model import LinearRegression

# linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)
```

In [47]:

```
from sklearn.metrics import r2_score
from sklearn.metrics import root_mean_squared_error

print("R2 score: ", r2_score(y_test, y_pred))
print("RMSE: ", root_mean_squared_error(y_test, y_pred))
```

```
R2 score:  0.5280774655910482
RMSE:  3.3117630216159206
```

Random Forest Regression

In [48]:

```
from sklearn.ensemble import RandomForestRegressor

# random forest regression model
# takes more time to train (comeback after 2 mins)
model = RandomForestRegressor()
model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = model.predict(X_test_scaled)
```

In [49]:

```
from sklearn.metrics import r2_score
from sklearn.metrics import root_mean_squared_error

print("R2 score: ", r2_score(y_test, y_pred))
print("RMSE: ", root_mean_squared_error(y_test, y_pred))
```

```
R2 score:  0.5371153451076254
RMSE:  3.27989761261114
```

---

2. Email spam detection

```
from sklearn.model_selection import train_test_split
X = df.drop('Prediction', axis=1)
y = df['Prediction']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [7]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
y_pred_knn
```

In [9]:

```
from sklearn.metrics import confusion_matrix, classification_report
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("\n\nKNN Classification Report: \n", classification_report(y_test,
y_pred_knn))
```

```
from sklearn.svm import SVC
svm = SVC(kernel='linear', probability=True)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
```

```
y_pred_svm
```

In [12]:

```
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("\n\nKNN Classification Report: \n", classification_report(y_test,
y_pred_svm))
```

---

3. Neural Networks

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.

Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix

```
df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

Encoding Categorical Features

```
gender = pd.get_dummies(df['Gender'], drop_first=True)
geography = pd.get_dummies(df['Geography'], drop_first=True)
df = pd.concat([df, gender, geography], axis=1)
df = df.drop(['Geography', 'Gender'], axis=1)
df = df.dropna()
df


from sklearn.model_selection import train_test_split
X = df.drop(['Exited'], axis=1)
Y = df['Exited']
```

Data Normalization

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
X
x_train, x_test, y_train, y_test = train_test_split(X, Y)
```

Building the Neural Network Model

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(activation='relu', units=6))
model.add(Dense(activation='relu', units=6))
model.add(Dense(activation='sigmoid', units=1))
model.compile(optimizer='adam',                 loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=10, epochs=50)

y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
0.8596
```

---

4. Gradient Descent Algorithm

Implement Gradient Descent Algorithm to find the local minima of a function.

Gradient Descent is an optimization algorithm used to minimize (or maximize) functions by iteratively moving towards the optimal point. In machine learning, this is typically used to minimize the cost function.

```
#Initialize Parameters
cur_x = 2
rate = 0.01
precision = 0.000001
previous_step_size = 1
max_iters = 1000
iters = 0
df = lambda x : 2 * (x + 3) #Gradient of our function i.e (x + 3)²
```

In [4]:

```
#Run a loop to perform gradient Descent
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x
    cur_x -= rate * df(prev_x)
    previous_step_size = abs(prev_x - cur_x)
    iters += 1
print("Local Minima Occurs at :",cur_x)
Local Minima Occurs at : -2.999951128099859
```

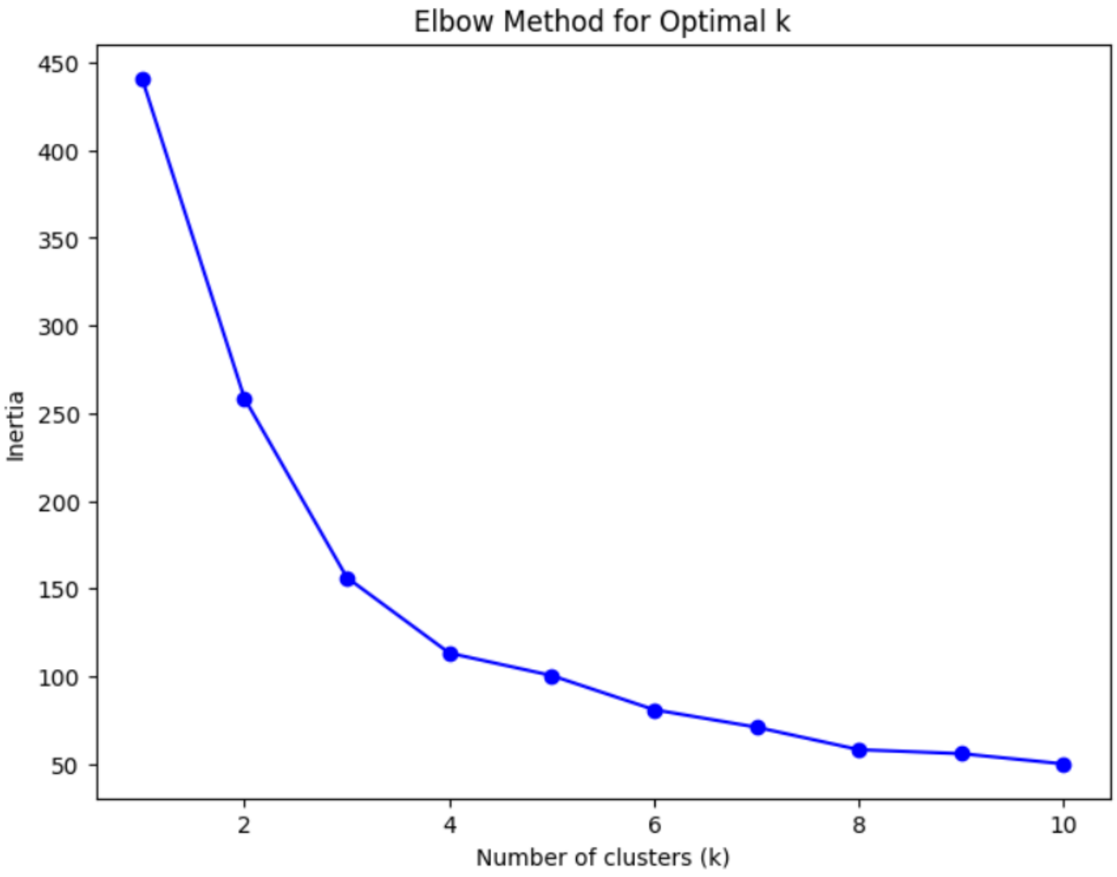
---

5. K means clustering

```
import pandas as pd

# Load dataset with a different encoding
df = pd.read_csv('sales_data_sample.csv', encoding ="unicode_escape")

# Inspect the data
print(df.head())
```

```
# Drop rows with missing values (optional)
df_cleaned = df.dropna()

# Assume we are clustering based on 'SALES', 'QUANTITYORDERED', 'PRICEEACH'
X = df_cleaned[['SALES', 'QUANTITYORDERED', 'PRICEEACH']]

# Normalize the data (optional, but recommended for K-Means)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Elbow method to find optimal number of clusters
inertia = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(8, 6))
plt.plot(K, inertia, 'bo-', color='blue')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



```
# Choose k based on elbow point (e.g., k=3)
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df_cleaned['Cluster'] = kmeans.fit_predict(X_scaled)

# Check cluster assignments
print(df_cleaned[['SALES', 'QUANTITYORDERED', 'PRICEEACH', 'Cluster']])
```

BT

1. Bank

Write a smart contract on a test network, for Bank account of a customer for following operations:

- Deposit money
- Withdraw Money
- Show balance

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract Bank{

 mapping(address=>uint)public user_account;
 mapping(address=>bool)public user_exists;

 function createAccount() public payable returns(string memory)
 {
  require(user_exists[msg.sender] == false , "Account Already Created");
  user_account[msg.sender] = msg.value;
  user_exists[msg.sender] = true;
  return "Account Created Successfully!";
 }

 function deposite(uint amount) public payable returns(string memory)
 {
  require(user_exists[msg.sender] == true , "Account does not exist");
  require(amount>0 , "Amount should be greater than 0");
  user_account[msg.sender] += amount;
  return "Amount Deposited Successfully";
 }

 function withdraw(uint amount) public payable returns(string memory)
 {
  require(user_exists[msg.sender] == true , "Account does not exist");
  require(amount>0 , "Amount should be greater than 0");
  require(user_account[msg.sender] >= amount , "Amount greater than the balance");
  user_account[msg.sender] -= amount;
  return "Amount withdrawn successfully";
 }
```

```
  function checkbalance() public view returns(uint)
  {
    return user_account[msg.sender];
  }

  function checkuser() public view returns(bool)
  {
    return user_exists[msg.sender];
  }

}
```

2. Student

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract StudentData {

  struct Student{
    uint stud_id;
    string name;
    string department;
  }

  Student[] students;

  function addStudent(uint id , string memory stud_name , string memory stud_dept)  public
{
    Student memory newStudent = Student(id, stud_name , stud_dept);
    students.push(newStudent);

  }

  function getData(uint id) public view returns(string memory , string memory)
  {
    for(uint i = 0 ; i<students.length ; i++)
    {
      if(students[i].stud_id == id)
      {
        return(students[i].name , students[i].department);
      }
    }
    return ("No Name Found" , "No Department Found");
  }

  function getNum() public view returns(uint)
  {
    return students.length;
  }

  receive() external payable {
    students.push(Student(1 , 'ABC' , 'DEF'));
  }
}
```