Department of Computer Engineering

LAB MANUAL

Laboratory Practice III

(410246)

Semester-VII

Companion Courses:

Design and Analysis of Algorithms (410241)

Machine Learning(410242)

Blockchain Technology(410243)

INDEX

Name of the Subject/Course: **410246 - Laboratory Practice III**

Class: **BE**

Sr. No.	Assignment Title
	Group A: Design and Analysis of Algorithms
1	Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.
2	Write a program to implement Huffman Encoding using a greedy strategy.
3	Write a program to solve a fractional Knapsack problem using a greedy method.
4	Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.
5	Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen_s matrix.
	Group B: Machine Learning
6	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. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset
7	Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

8	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 (5 points).
9	Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset. Dataset link : https://www.kaggle.com/datasets/abdallamahgoub/diabetes
10	Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method. Dataset link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data
Group C: Blockchain Technology	
11	Installation of MetaMask and study spending Ether per transaction.
12	Create your own wallet using Metamask for crypto transactions.
13	Write a smart contract on a test network, for Bank account of a customer for following operations: • Deposit money • Withdraw Money • Show balance
14	Write a program in solidity to create Student data. Use the following constructs: • Structures • Arrays • Fallback Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.
15	Write a survey report on types of Blockchains and its real time use cases.
16	Mini Project on DAA.
17	Mini Project on ML.
18	Mini Project on BT.

Assignment No.1

Aim: Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

Objective: Student will be able to learn

1. Concept of Fibonacci Numbers.
2. Analysis of time complexity in Non Recursive and Recursive way.

Theory:

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2

With seed values

F0 = 0 and F1 = 1.

Non Recursive Algorithm:

```
1  Algorithm fibonacci(n)
2  //Calculate nth fibonacci number
3  {
4        if(n<=1)
5        write (n);
6        else
7        {
8        fnm2:=0; fnm1:=1;
9              for i:=2 to n  do
10             { fn:=fnm1+fnm2;
11             fnm2:=fnm1;
12             fnm1:=fn;
13             }
14       write(fn)
15       }
16 }
```

Time Complexity:

Two cases (1) n = 0 or 1 and (2) n>1.

1. When n = 0 or 1, lines 4 and 5 get executed once each. Since each line has an s/e of 1, the total step count for this case is 2.

2. When n > 1, lines 4, 8, and 14 are each executed once. Line 9 gets executed n times, and lines 11 and 12 get executed n-1 times each. Line 8 has an s/e of 2, line 12 has an s/e of 2, and line 13 has an s/e of 0. The remaining lines that get executed have s/e‘s of 1. The total steps for the case n > 1 is therefore 4n + 1.

Recursive Algorithm:

```
Algorithm Fibonacci(n)
{
if (n <= 1)
return n;
else
return Fibonacci(n - 1) + Fibonacci(n - 2); }
```

Time complexity:

T(n) = T(n-1) + T(n-2) + c

= 2T(n-1) + c //from the approximation T(n-1) ~ T(n-2)

= 2*(2T(n-2) + c) + c

= 4T(n-2) + 3c

= 8T(n-3) + 7c

= 2 k * T(n - k) + (2 k - 1)*c

Let's find the value of k for which: n - k = 0

k = n

T(n) = 2 n * T(0) + (2 n - 1)*c

= 2 n * (1 + c) - c

$T(n) = 2^n$

Conclusion: We have studied Recursive and Non-Recursive way to Calculate Fibonacci Numbers.

Assignment No.2

Aim: Write a program to implement Huffman Encoding using a greedy strategy.

Objective: Student will be able to learn

1. Concept of Huffman Encoding
2. Analysis of time complexity

Theory:

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.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be ―cccd‖ or ―ccb‖ or ―acd‖ or ―ab‖.

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.

2. Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

i) Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root

ii) Extract two nodes with the minimum frequency from the min heap.

iii) Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

iv) Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Let us understand the algorithm with an example:

character Frequency

a	5
b	9
c	12
d	13
e	16
f	45

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.

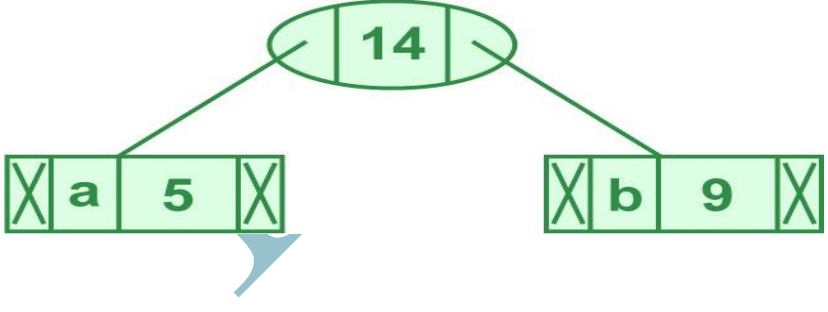


Illustration of step 2

Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25

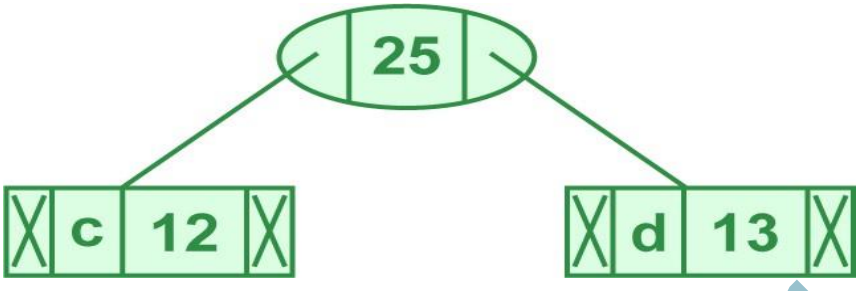


Illustration of step 3

Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes

character Frequency

Internal Node 14

e 16

Internal Node 25

f 45

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30

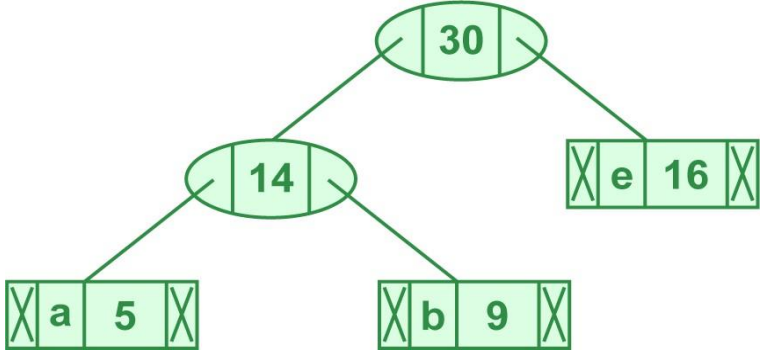


Illustration of step 4

Now min heap contains 3 nodes.

character Frequency

Internal Node 25

Internal Node 30

f 45

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55

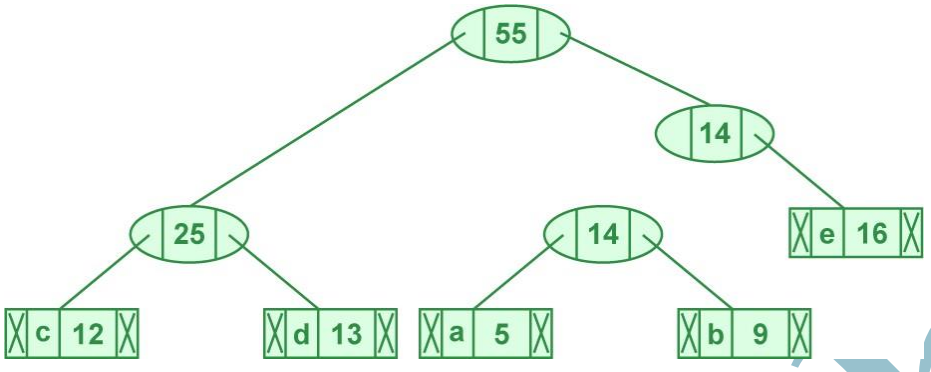


Illustration of step 5

Now min heap contains 2 nodes.

character Frequency

f 45

Internal Node 55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100

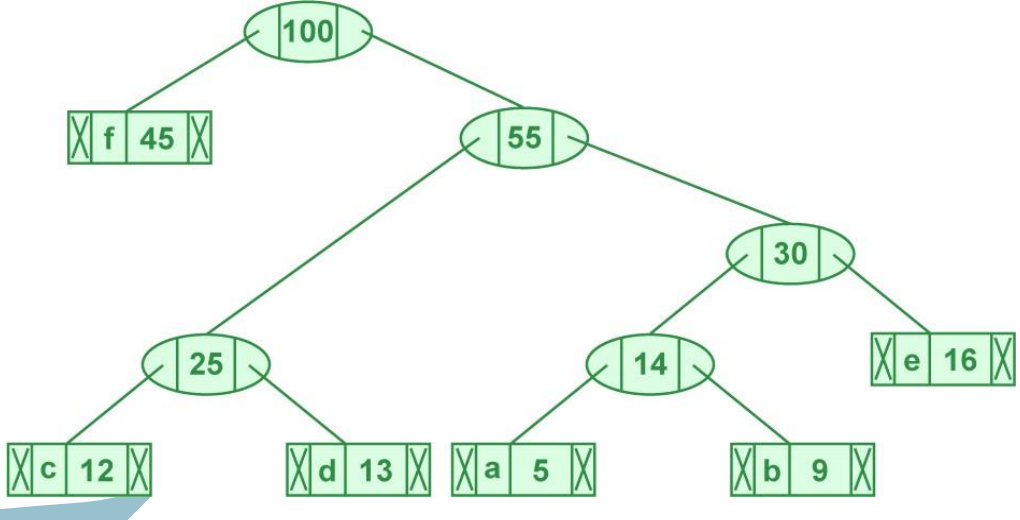


Illustration of step 6

Now min heap contains only one node.

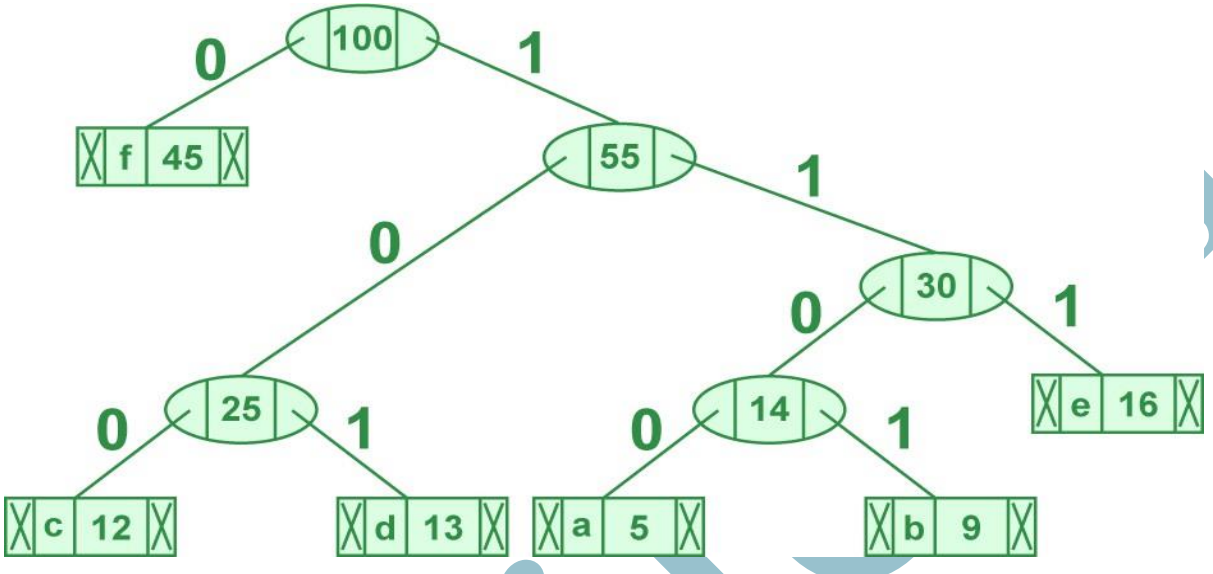
character Frequency

Internal Node 100

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



Steps to print code from Huffman Tree

The codes are as follows:

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111

Time complexity: O(nlogn) where n is the number of unique characters.

Conclusion: We have studied Huffman encoding using Greedy Method.

Assignment No.3

Aim: Write a program to solve a fractional Knapsack problem using a greedy method.

Objective: Student will be able to learn

1. Concept of Knapsack Problem
2. Concept of implementation of Fractional Knapsack using Greedy Strategy.

Theory:

Given the weights and values of N items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In Fractional Knapsack, we can break items for maximizing the total value of the knapsack

Note: In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or don‘t take it.

An efficient solution is to use the Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them until we can‘t add the next item as a whole and at the end add the next item as much as we can which will always be the optimal solution to this problem.

Algorithm GreedyKnapsack(M,N)

```
//p[1:n] and w[1:n] contain the profits and weights respectively
//of the n objects ordered such that p[i]/w[i]>=p[i+1]/w[i+1]
//m is the knapsack size and x[1:n] is the solution vector.
{       for i:=1 to n do x[i]:=0.0;     // initialize x
        U:=m;
        for i:=1 to n do
        {       if (w[i]>U) then break;
                x[i]:=1.0;
                U=U-w[i];
        } if(i<=n) then x[i]:=U/w[i];
}
```

Analysis:

The main time taking step is the sorting of all items in decreasing order of their profit/ weight ratio.

If the items are already arranged in the required order, then while loop takes O(n) time.

The average time complexity of Quick Sort is O(nlogn).

Therefore, total time taken including the sort is O(nlogn).

Conclusion: We have studied to implement fractional Knapsack using Greedy Strategy.

Assignment No.4

Aim: Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

Objective: Students will learn

1. The basic concept of 0/1 Knapsack Problem
2. The basic concept of 0/1 Knapsack using Dynamic Programming and Branch and Bound Approach.

Theory:

Knapsack Problem:

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Dynamic Programming Approach

Let *i* be the highest-numbered item in an optimal solution S for W dollars. Then *S' = S - {i}* is an optimal solution for *W - wi* dollars and the value to the solution *S* is *Vi* plus the value of the sub-problem.

We can express this fact in the following formula: define c[i, w] to be the solution for items 1,2, … , i and the maximum weight w.

The algorithm takes the following inputs

- The maximum weight W
- The number of items n
- The two sequences v = <v1, v2, …, vn> and w = <w1, w2, …, wn>

Algorithm

```
Dynamic-0-1-knapsack (v, w, n, W)
for w = 0 to W do
  c[0, w] = 0
for i = 1 to n do
  c[i, 0] = 0
  for w = 1 to W do
    if wi ≤ w then
       if vi + c[i-1, w-wi] then
          c[i, w] = vi + c[i-1, w-wi]
       else c[i, w] = c[i-1, w]
    else
       c[i, w] = c[i-1, w]
```

The set of items to take can be deduced from the table, starting at c[n, w] and tracing backwards where the optimal values came from.

If *c[i, w] = c[i-1, w]*, then item *i* is not part of the solution, and we continue tracing with c[i-1, w]. Otherwise, item *i* is part of the solution, and we continue tracing with c[i-1, w-W].

Analysis

This algorithm takes θ(*n, w*) times as table *c* has (*n* + 1).(*w* + 1) entries, where each entry requires θ(1) time to compute.

Branch and Bound Approach:

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. Branch and Bound solve these problems relatively quickly.

Let us consider below 0/1 Knapsack problem to understand Branch and Bound. Consider an example where *n* = 4, and the values are given by {10, 12, 12, 18}and the weights given by {2, 4, 6, 9}. The maximum weight is given by *W = 15*. Here, the solution to the problem will be including the first, third and the fourth objects. In solving this problem, we shall use the Least Cost- Branch and Bound method, since this shall help us eliminate exploring certain branches of the tree. We shall also be using the fixed-size solution here. Another thing to be noted here is that this problem is a maximization problem, whereas the Branch and Bound method is for minimization problems. Hence, the values will be multiplied by -1 so that this problem gets converted into a minimization problem.

Now, consider the 0/1 knapsack problem with *n* objects and total weight W. We define the upper bound(U) to be the summation of vixi (where vi denotes the value of that objects, and xi is a binary value, which indicates whether the object is to be included or not), such that the total weights of the included objects is less than *W*. The initial value of U is calculated at the initial position, where objects are added in order until the initial position is filled.

We define the cost function to be the summation of vifi, such that the total value is the maximum that can be obtained which is less than or equal to *W*. Here fi indicates the fraction of the object that is to be included. Although we use fractions here, it is not included in the final solution.

Here, the procedure to solve the problem is as follows are:

- Calculate the cost function and the Upper bound for the two children of each node. Here, the (i + 1)th level indicates whether the ith object is to be included or not.
- If the cost function for a given node is greater than the upper bound, then the node need not be explored further. Hence, we can kill this node. Otherwise, calculate the upper bound for this node. If this value is less than *U*, then replace the value of *U* with this value. Then, kill all unexplored nodes which have cost function greater than this value.
- The next node to be checked after reaching all nodes in a particular level will be the one with the least cost function value among the unexplored nodes.

- While including an object, one needs to check whether the adding the object crossed the threshold. If it does, one has reached the terminal point in that branch, and all the succeeding objects will not be included.

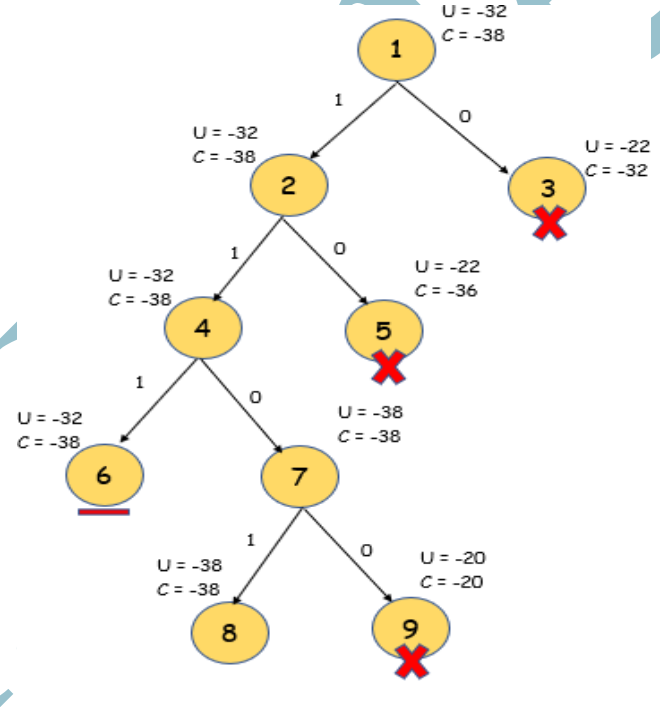
In this manner, we shall find a value of U at the end which eliminates all other possibilites. The path to this node will determine the solution to this problem.

Time and Space Complexity:

Even though this method is more efficient than the other solutions to this problem, its worst case time complexity is still given by $O(2^n)$, in cases where the entire tree has to be explored. However, in its best case, only one path through the tree will have to explored, and hence its best case time complexity is given by $O(n)$. Since this method requires the creation of the state space tree, itsspace complexity will also be exponential.

Solving an Example: Consider the problem with n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15. Here, we calculate the initital upper bound to be U = 10 + 10 + 12 = 32. Note that the 4th object cannot be included here, since that would exceed W. For the cost, we add 3/9 th of the final value, and hence the cost function is 38. Remember to negate the values after calculation before comparison.

After calculating the cost at each node, kill nodes that do not need exploring. Hence, the final state space tree will be as follows (Here, the number of the node denotes the order in which the state space tree was explored):



Note here that node 3 and node 5 have been killed after updating U at node 7. Also, node 6 is not explored further, since adding any more weight exceeds the threshold. At the end, only nodes 6 and 8 remain. Since the value of U is less for node 8, we select this node. Hence the solution is {1, 1, 0, 1}, and we can see here that the total weight is exactly equal to the threshold value in this case.

Conclusion: We can solve 0/1 knapsack problem by using Dynamic Programming and Branch and Bound Approach.

Assignment No.5

Aim: Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen_s matrix.

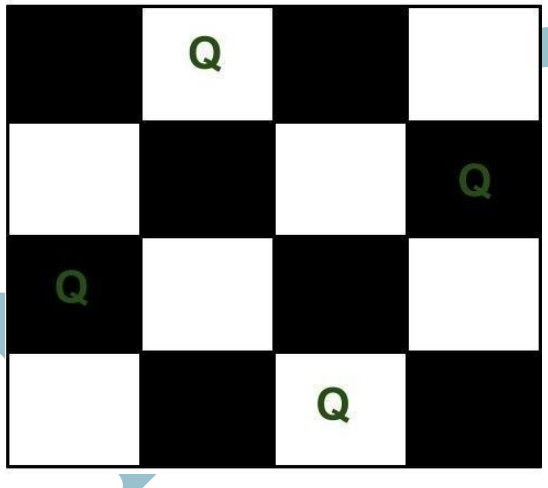
Objective:

Students will learn

1. The Basic Concepts of N Queens Problem
2. N queens using Backtracking

Theory:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, the following is a solution for the 4 Queen problems.



The expected output is a binary matrix that has 1s for the blocks where queens are placed. For example, the following is the output matrix for the above 4 queen solution.

{ 0, 1, 0, 0}

{ 0, 0, 0, 1}

{ 1, 0, 0, 0}

{ 0, 0, 1, 0}

Backtracking Algorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

1) Start in the leftmost column
2) If all queens are placed
   return true
3) Try all rows in the current column.
   Do following for every tried row.
   a) If the queen can be placed safely in this row
      then mark this [row, column] as part of the
      solution and recursively check if placing
      the queen here leads to a solution.
   b) If placing the queen in [row, column] leads to
      a solution then return true.
   c) If placing queen doesn't lead to a solution then
      unmark this [row, column] (Backtrack) and go to
      step (a) to try other rows.
4) If all rows have been tried and nothing worked,
   return false to trigger backtracking.

Time Complexity: O(N!)

Conclusion: We have studied N Queen Problem using Backtracking Approach.

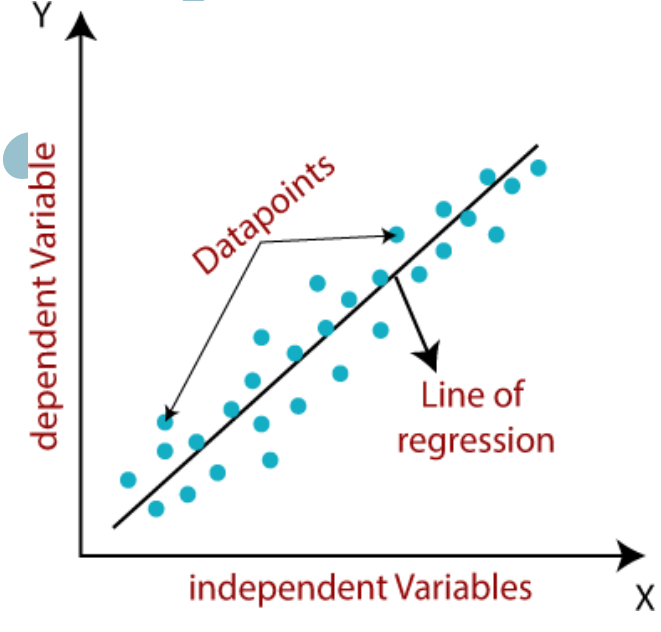
Assignment No.6

Aim: 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.
Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

Objective:
Student will learn:
1] The basic concept and implementation logic of linear regression and random forest regression model.
2] Different evaluation metrics used for regression models like R2, RMSE, etc.

Theory:

Linear Regression:
Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
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.
The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



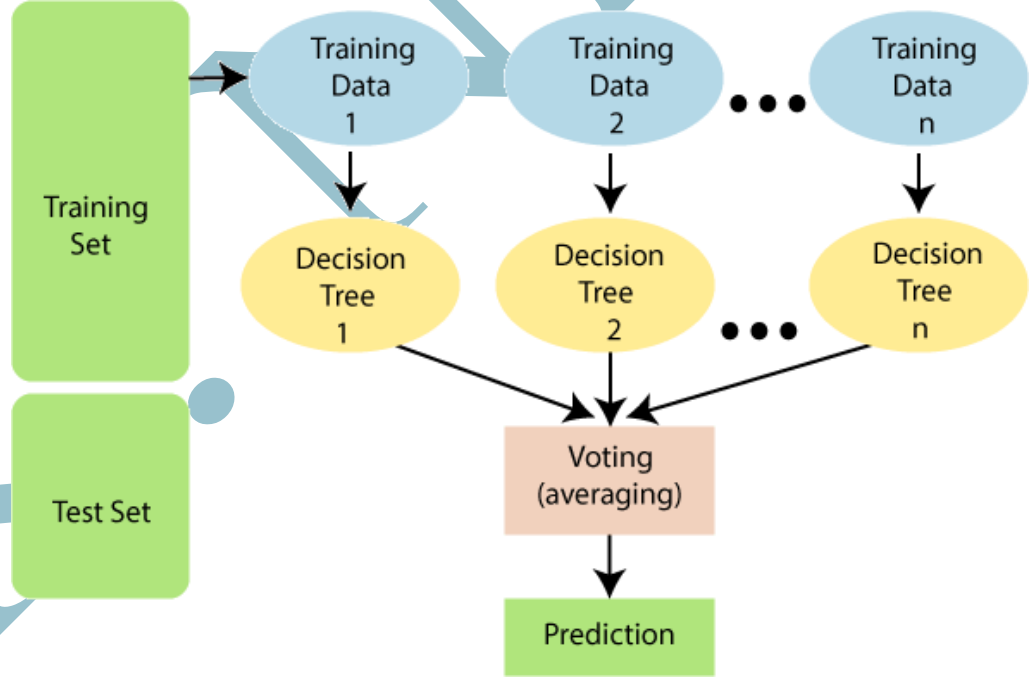
Mathematically, we can represent a linear regression as:

$y= a_0+a_1x+ \varepsilon$

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.

Random Forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
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.
The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.
The below diagram explains the working of the Random Forest algorithm:



Evaluation metrics of regression models:

1] RMSE: Root Mean Squared Error is the square root of Mean Squared error. It measures the standard deviation of residuals.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

2] R2: The coefficient of determination or R-squared represents the proportion of the variance in the dependent variable which is explained by the linear regression model. It is a scale-free score i.e. irrespective of the values being small or large, the value of R square will be less than one.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Python Packages needed

- pandas
  - Data Analytics
- numpy
  - Numerical Computing
- matplotlib.pyplot
  - Plotting graphs
- Sklearn
  - Regression Classes

Steps to establish Linear Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship are −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create the object of Linear Regression Class.
- Train the algorithm with dataset of X and y.
- Get a summary of the relationship model to know the average error in prediction. Also called residuals.
- To predict the weight of new persons, use the predict() function.

Conclusion: We have studied the Linear Regression and Random forest algorithm. Also implemented and evaluated the models using R2 and RMSE scores.

Assignment No.7

Aim: Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

Objective:

Student will learn:

1] The basic concept and implementation logic of K-Nearest Neighbors algorithm.

2] The basic concept and implementation logic of Support Vector Machine algorithm.

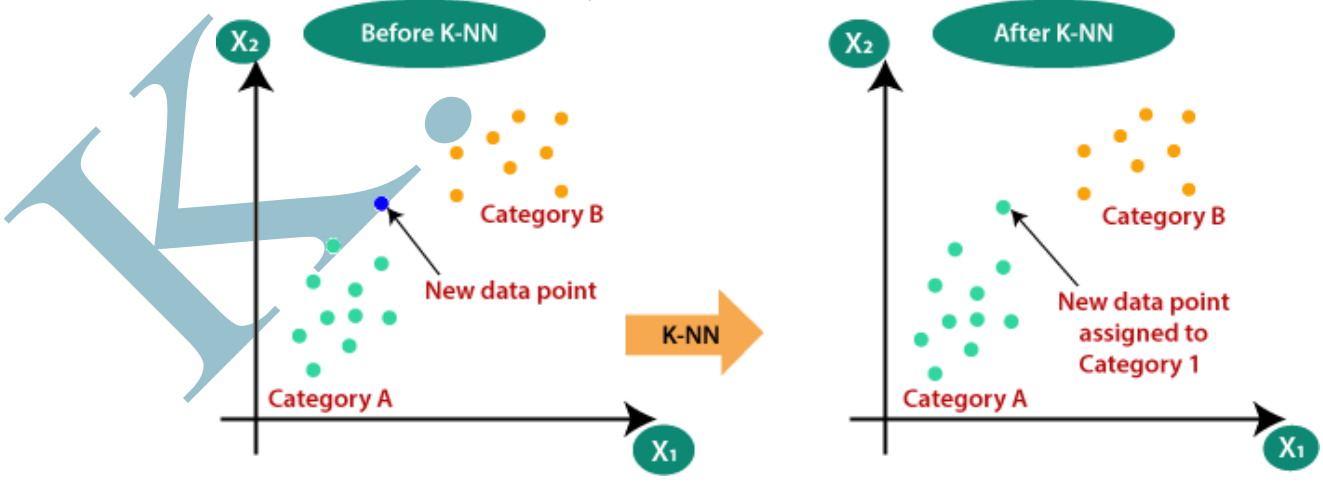
3] Different evaluation metrics used for classification models like accuracy, precision, recall, F-score, etc.

Theory:

K-Nearest Neighbor (KNN) Algorithm:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example: Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset.



How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors

- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

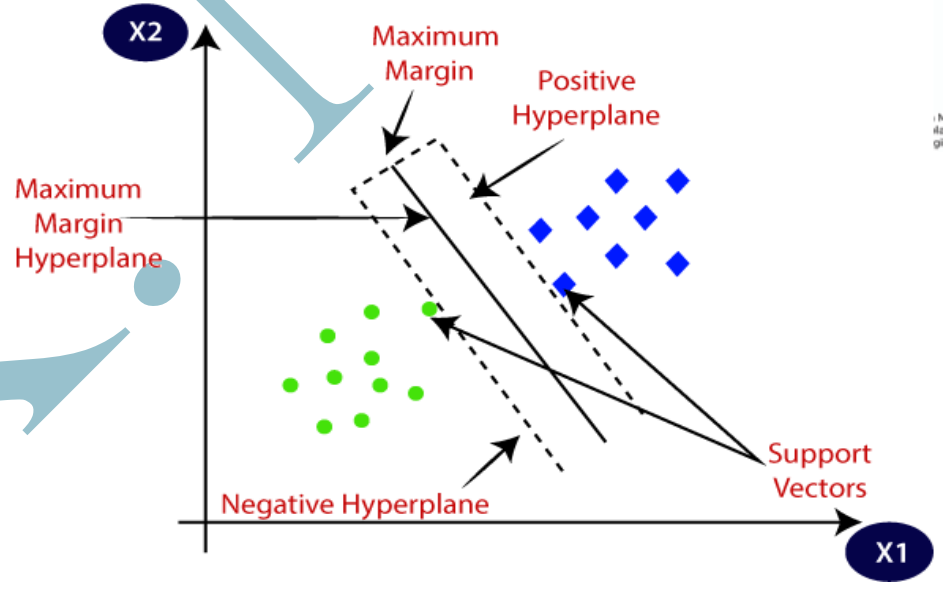
Support Vector Machine (SVM) Algorithm:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyper plane:



SVM algorithm can be used for Face detection, image classification, text categorization, etc.

For analysis of performance of KNN and SVM, use different evaluation metrics like accuracy, precision, recall, F-score, etc.

Conclusion: We have studied the KNN and SVM algorithm. Also implemented and evaluated the models using accuracy, precision, recall, F-score.

Assignment No.8

Aim: 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 (5 points).

Objective:

Student will learn:

1] The basic concept and implementation logic of normalization of data.

2] The basic concept and implementation logic of accuracy score and confusion matrix.

Theory:

Normalization in Machine Learning:

Normalization is one of the most frequently used data preparation techniques, which helps us to change the values of numeric columns in the dataset to use a common scale. Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

Mathematically, we can calculate normalization with the below formula:

Xn = (X - Xminimum) / (Xmaximum - Xminimum)

- Xn = Value of Normalization
- Xmaximum = Maximum value of a feature
- Xminimum = Minimum value of a feature

Accuracy Score:

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

Confusion Matrix in Machine Learning:

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

Conclusion: We have studied the concept of normalization of data, accuracy score and confusion matrix. Also implemented and calculated the accuracy score.

Assignment No.9

Aim: Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.
Dataset link: https://www.kaggle.com/datasets/abdallamahgoub/diabetes

Objective:
Student will learn:
1] The basic concept and implementation logic of K-Nearest Neighbors.
2] The basic concept and implementation logic of accuracy, error rate, precision and recall.

Theory:

K-Nearest Neighbor (KNN) Algorithm:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Confusion Matrix in Machine Learning:

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an error matrix. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.

- It looks like the below table:

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative (TN):** Model has given prediction No, and the real or actual value was also No.
- **True Positive (TP):** The model has predicted yes, and the actual value was also true.
- **False Negative (FN):** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive (FP):** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.

Accuracy = TP+TN/TP+FP+FN+TN

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? We have got recall of 0.631 which is good for this model as it's above 0.5.

Recall = TP/TP+FN

Error Rate - what percentage of our prediction are wrong.

Error Rate = 1-Accuracy

Conclusion: We have studied the K-Nearest Neighbors algorithm. Also implemented and evaluated the models using accuracy, error rate, precision, and recall.

Assignment No.10

Aim: Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.
Dataset link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data

Objective:
Student will learn:
1] The basic concept and implementation logic of K-Means clustering/hierarchical clustering.
2] The basic concept of elbow method used to determine the number of clusters.

Theory:
K-Means Clustering Algorithm:

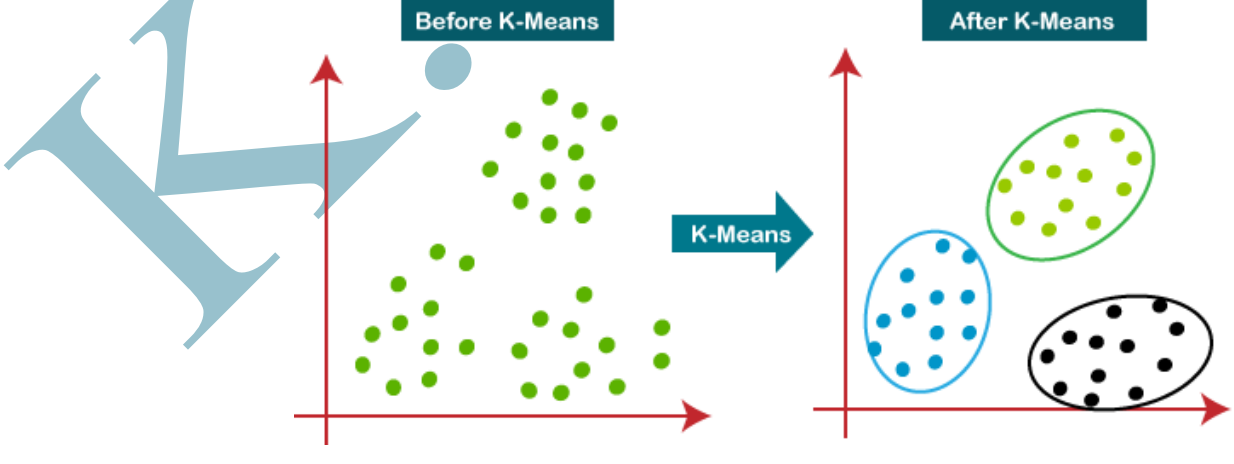
K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.
Step-2: Select random K points or centroids. (It can be other from the input dataset).
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
Step-4: Calculate the variance and place a new centroid of each cluster.
Step-5: Repeat the third step, which means reassign each datapoint to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.

Elbow Method:

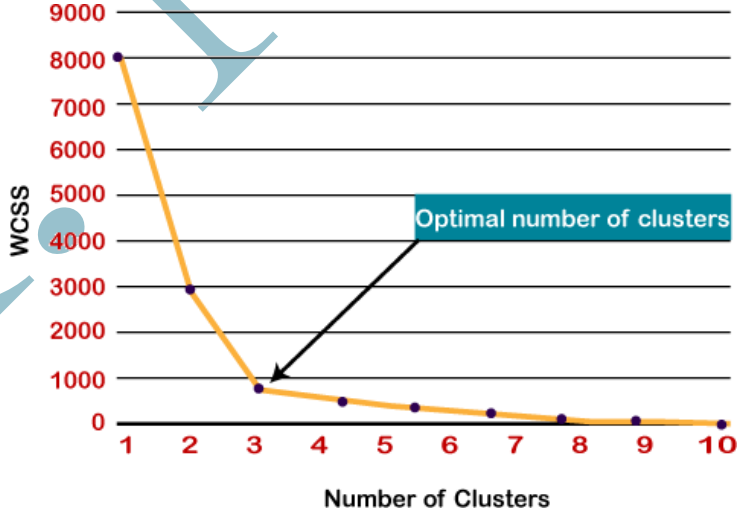
The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$\mathbf{WCSS} = \sum_{Pi \text{ in Cluster1}} \mathbf{distance}(P_i\ C_1)^2 + \sum_{Pi \text{ in Cluster2}} \mathbf{distance}(P_i\ C_2)^2 + \sum_{Pi \text{ in CLuster3}} \mathbf{distance}(P_i\ C_3)^2$$

In the above formula of WCSS,
$\sum_{Pi \text{ in Cluster1}} \text{distance}(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.
To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.
To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Conclusion: We have studied the K-Means clustering algorithm and elbow method to find the optimal number of clusters. Also implemented the K-Means clustering algorithm using python language.

Assignment No.11

Aim:- Installation of Meta Mask and study spending Ether per transaction.

Objectives: Student will be able to learn

1. Concept of Meta mask
2. Installation and study of Meta mask and Ether

Theory:-

ETAMASK

- The existing browsers on our systems are centralized. In order to create a de-centralized system we add a plug-in called ―Metamask‖
- We need it because we need ―ether‖ (currency for blockchain)
- Installed Metamask(Gives a virtual Ethereum wallet)
- Created a simple smart contract through MetaMask(Using fake ether i.e currency of blockchain)
- In-depth study of state-of-art on smart contract implementation.

REMIX IDE

Platform to create and deploy smart contract, supports solidity .

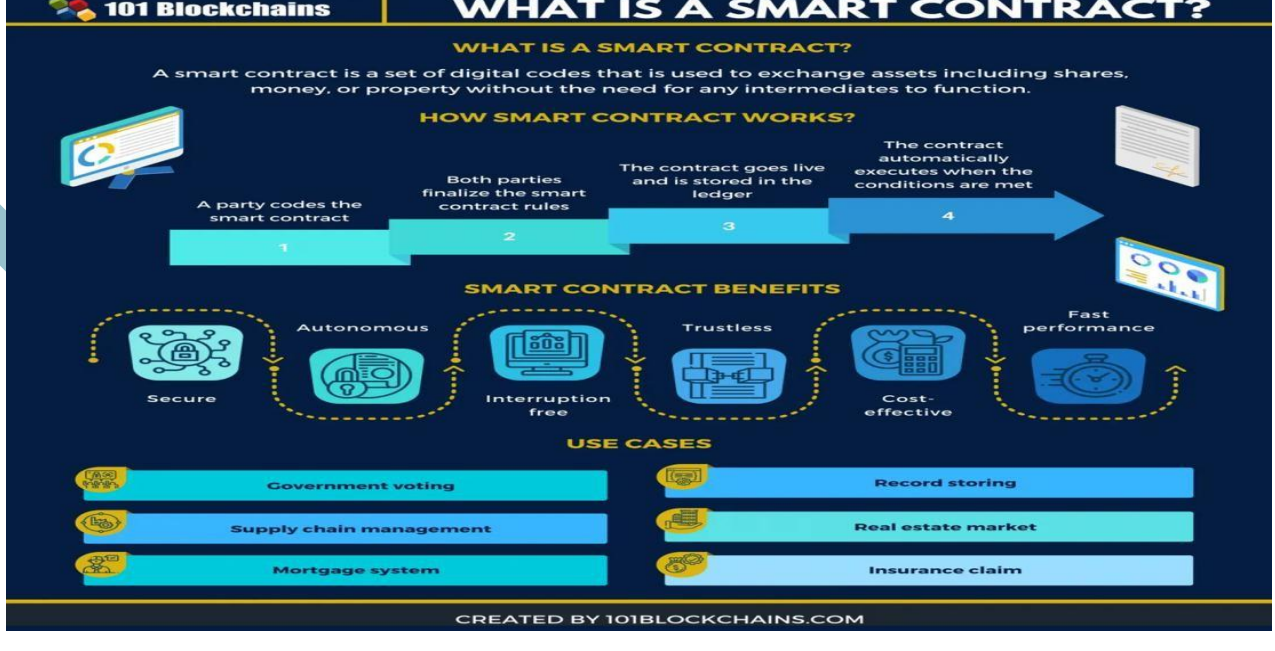
SOLIDITY

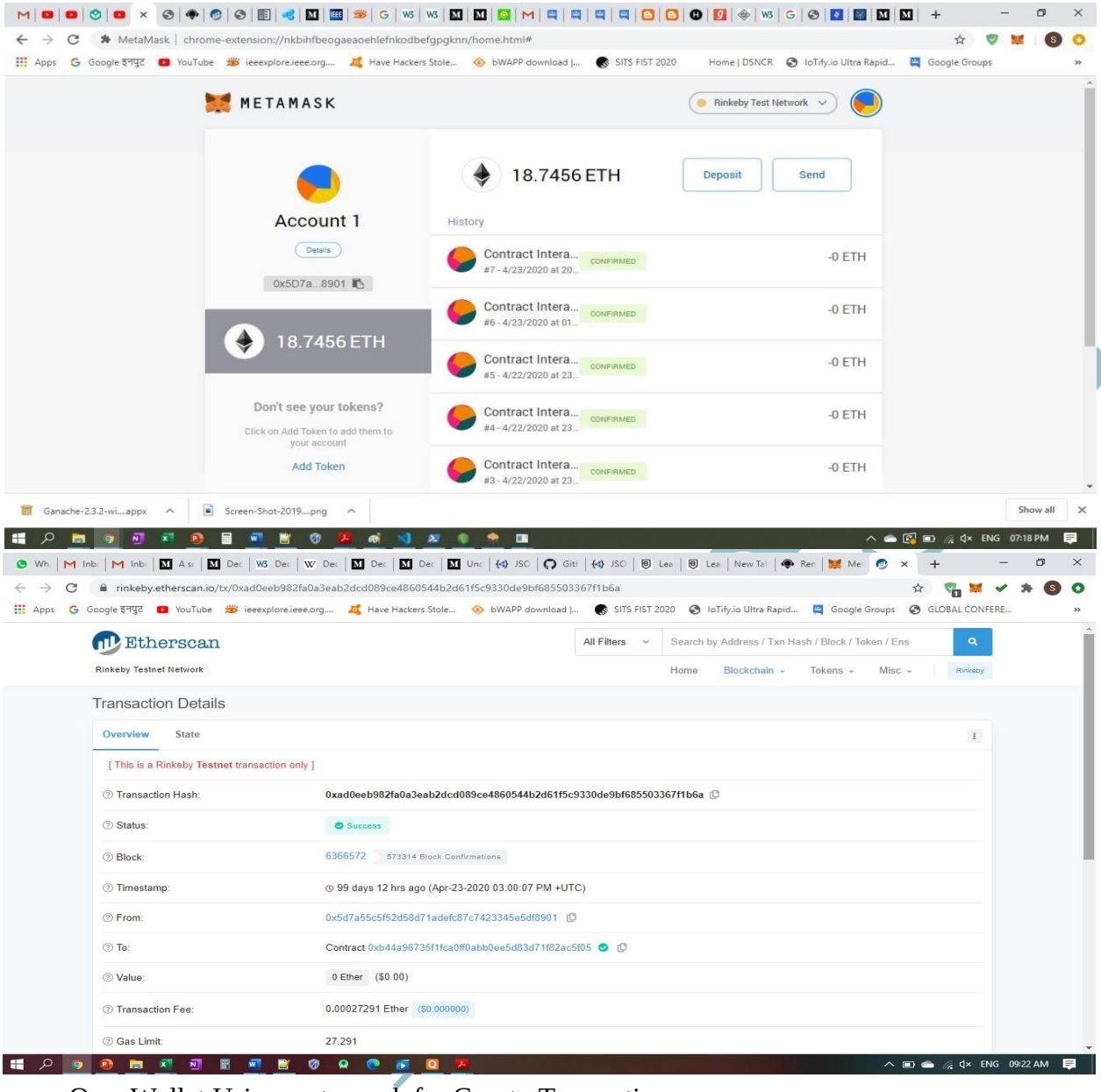
A Language to create smart contracts, similar to JavaScript. reating a De-centralized platform for testing a smart contract

Pre-requisites for creating a Block chain environment:

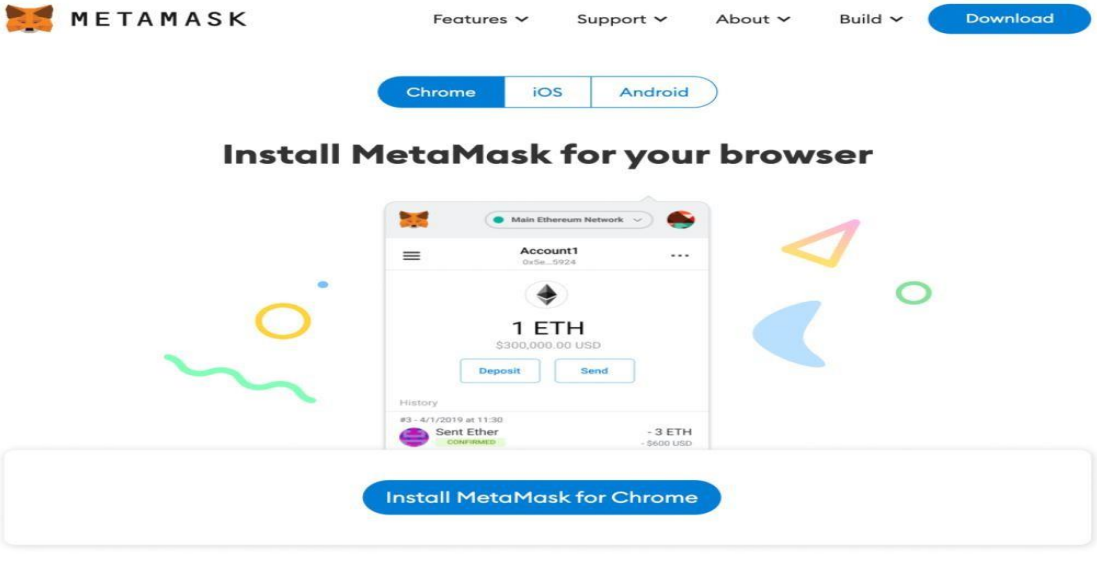
Adding Metamask extension to default browser Creating a wallet through test network Rinkeby. Currency needed for block chain transaction – ether

Rinkeby gives this ether – 18.75 ethers / 3 days Smart contract in solidity language on IDE Remix

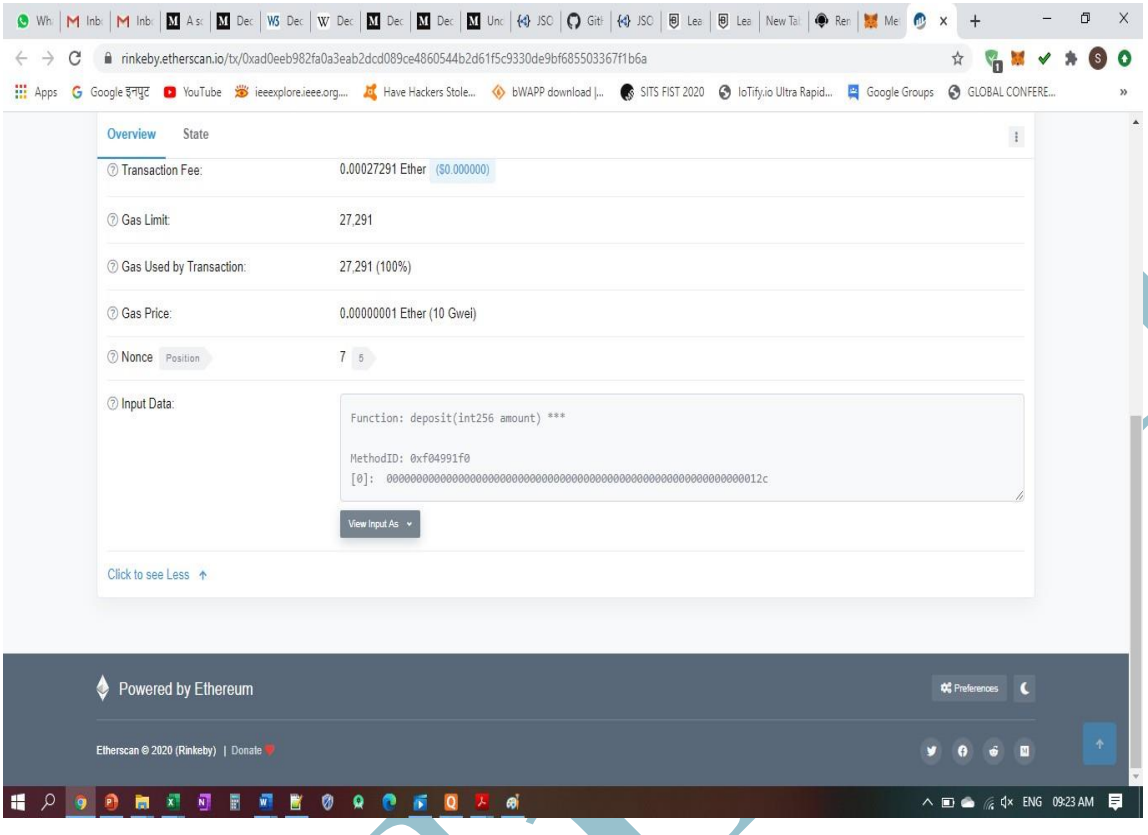




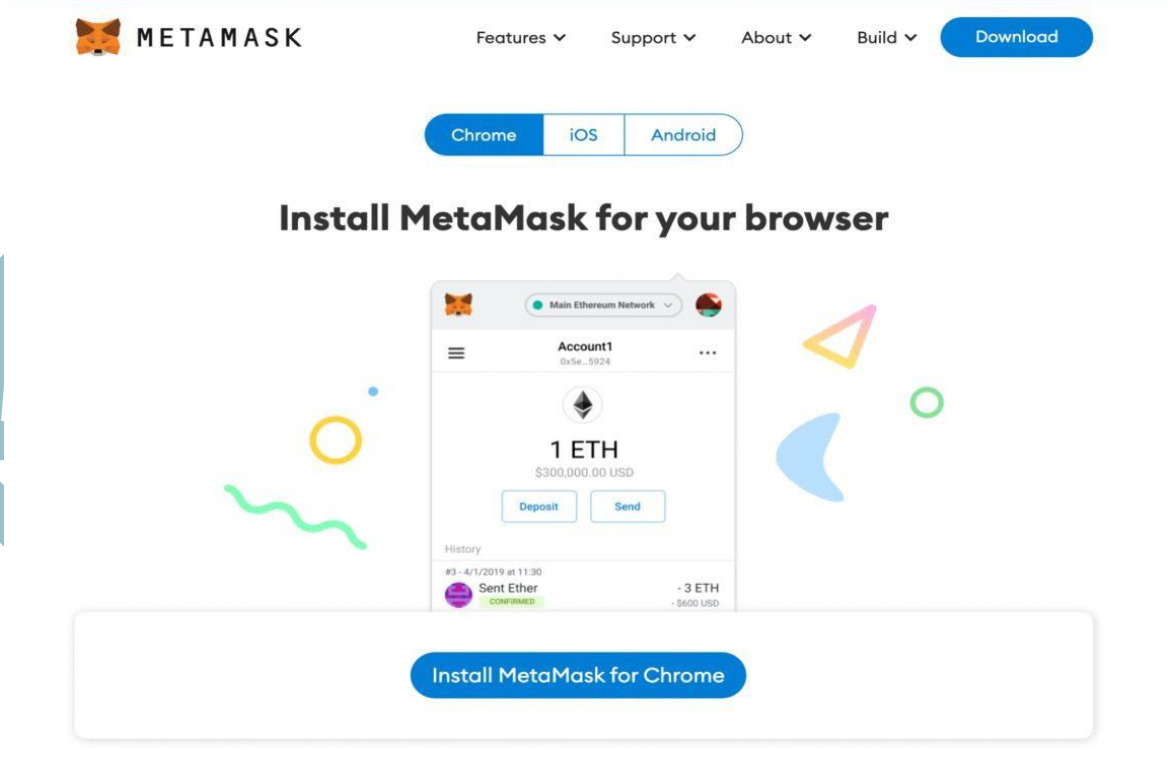
Create your Own Wallet Using meta mask for Crypto Transaction



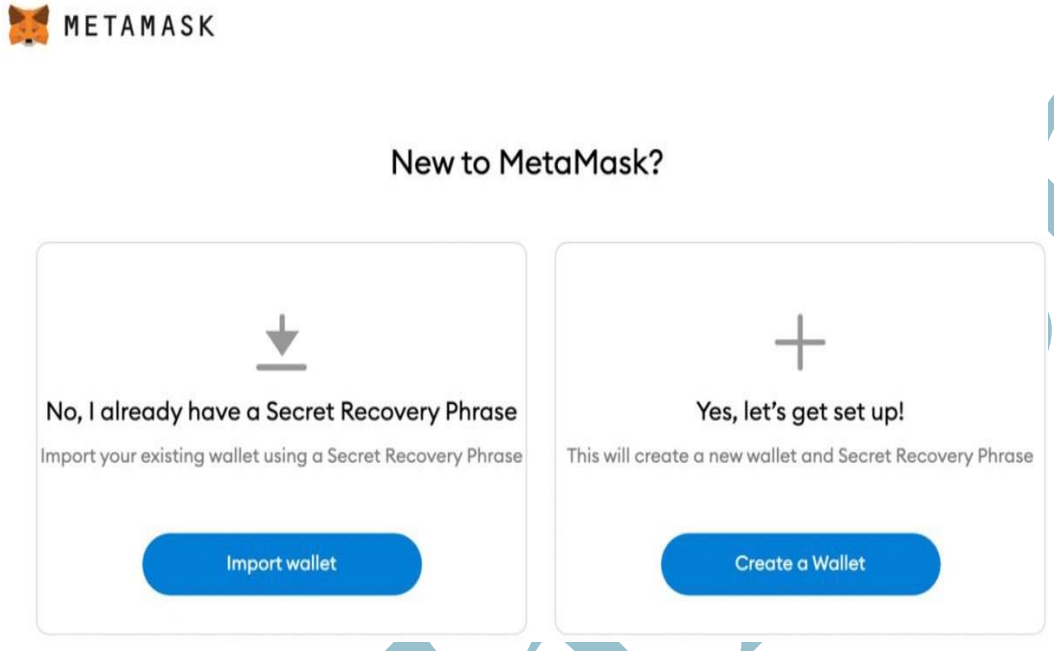
ETHER (TRANSACTION FEE SPENT ON THE SMART CONTRACT



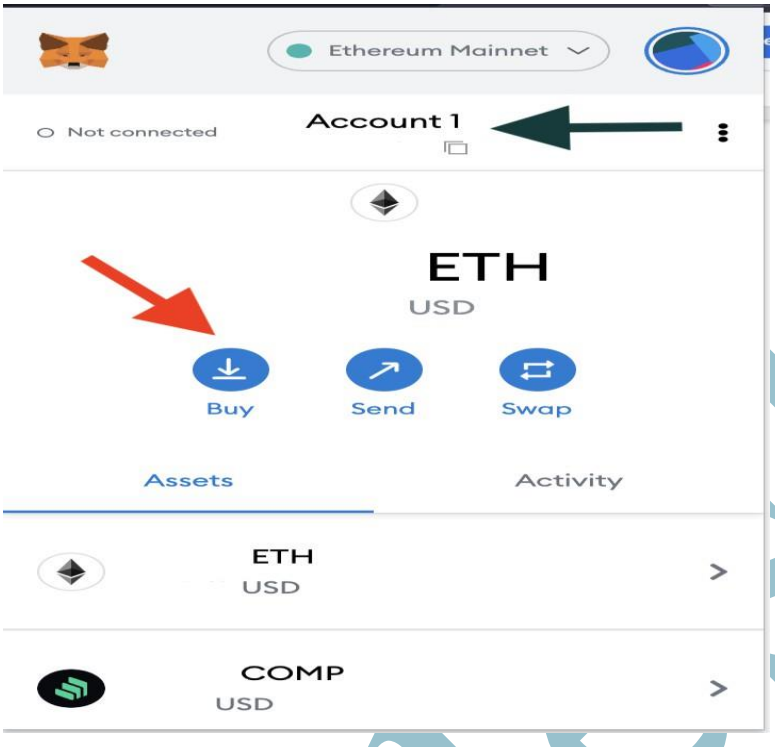
Create Your Own Wallet Using Meta mask For Crypto Transaction



Navigate to the extension icon in the top right corner of your web browser and find the MetaMask option, once you've successfully downloaded the software. Click the ―Get Started‖ button and you'll be taken to the next page and presented with two options (see below.)



Once you've completed the above steps, you'll be able to access your new MetaMask wallet. There are two main components you'll need to familiarize yourself with so that you can begin using the software:
Identifying your public address: This is the address you can freely share with people or platforms like exchanges in order to receive cryptocurrency into your wallet. Think of it as your home address that you share with people to receive inbound mail. It's always advisable, however, to check to make sure any inbound tokens are compatible with MetaMask first before receiving them, otherwise, they might be lost forever.
How to fund/buy and send: These are the core functions of MetaMask.



You can locate your unique MetaMask public address by clicking the ―Account 1″ button (black arrow). Finally, in order to begin interacting with any Ethereum platform, you‘ll first need to fund your MetaMask wallet with an amount of ether – the native cryptocurrency of Ethereum. All actions on the blockchain cost a fee, whether that‘s moving tokens from A to B or creating an NFT collection. This fee, known as a ―gas‖ fee, is denominated in ether

How much you choose to fund your wallet depends on how much you intend to interact with various platforms. For moderate use, $100 worth of ether is usually a good starting point to cover any initial fees.

Conclusion: We have successfully studied the installation of Metamask.

Assignment No.12

Aim:- Create your own wallet using Metamask for crypto transactions.

Objectives: Student will be able to learn

1. Concept of Meta mask
2. own wallet using Metamask for crypto transactions

Theory:-

A Brief introduction to MetaMask

MetaMask is an open-source, straightforward, and easy-to-use cryptocurrency wallet. It functions as a web browser extension available for Chrome, Firefox, Brave, or a mobile application for iOS or Android. Initially, this wallet supported only Ether and ERC-20 tokens, and now it is compatible with ERC-721 and ERC-1155 token standards. Furthermore, MetaMask benefits include interaction with websites; hence, it can function as a connection node for various DApps on Ethereum.

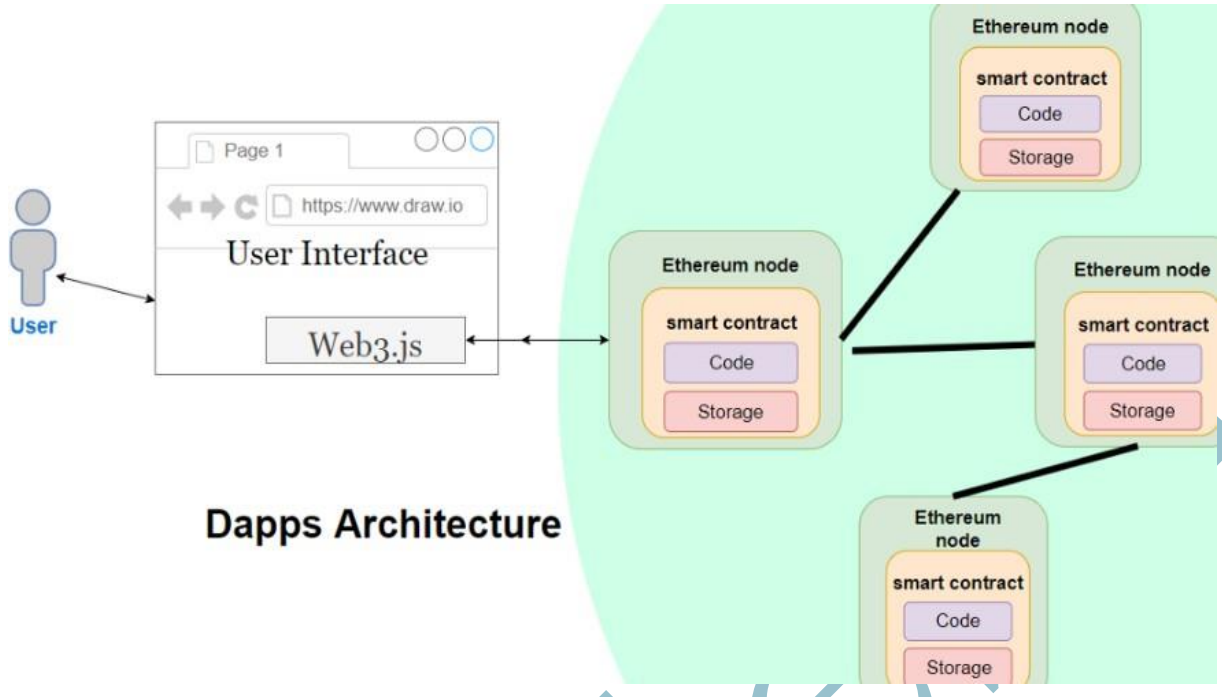
Adrian Devis and Dan Finlay are the MetaMask developers. Their idea was revolutionary and straightforward; they intended to create a web browser extension that would allow managing cryptocurrency and using the browser for fast and secure access with DApps. ConsenSys Software Inc. — a development company, focusing on applications that use Ethereum‘s blockchain, implemented the idea in 2016.

The solution used Ethereum‘s interface and a web API called web3.js. This Ethereum library is the fundament of MetaMask since it allows the browser to interact with the local or remote blockchain nodes via HTTP, IPC, and WebSocket; also, it gained the ability to record and read data from smart contracts, transfer tokens, etc. In another way, web3.js allowed the blockchain developers to create proxy and communication bridges between MetaMask, DApps, and the user.

Adrian Devis and Dan Finlay admit that their idea was great. Yet, the technical implementation was super complicated, especially in providing security for the users (web wallets are considered the most vulnerable to hacker attacks). Nonetheless, ConsenSys succeeded, and on the 14th of July in 2016, they offered the first version of MetaMask web browser cryptocurrency wallet for Chrome. Later, they presented a version for Firefox, Brave, and other popular browsers. In 2019 they also launched the mobile version of the MetaMask cryptocurrency wallet.

How does the MetaMask wallet function?

As we mentioned above, the MetaMask cryptocurrency wallet employs the web3.js library to function. This library is a part of the official Ethereum product. The library was developed focusing on the requirements of web applications that could interact with the Ethereum blockchain and take advantage of all blockchain‘s benefits and functions. MetaMask is a cryptocurrency wallet for Ethereum and an instrument that helps to interact with DApps. MetaMask connects the extension to the DApp so that to fulfill both tasks. When the application identifies the MetaMask, it creates a connection, and the user can start using all the features of a specific application.



For instance, it can assets trading, access to resources or services, or any other task within the capability of a DApp. Each action has its cost (transaction fee) that must be paid in Ethereum or any specified token. MetaMask wallet has all instruments and protocols for this purpose.

Hence, we can state that Metamask also controls the interaction of the user and DApp, and processes the operations required for specific actions, besides the function of a wallet. Reliable and secure cryptography and safe internet connection are the environments for these operations.

Furthermore, MetaMask can generate asymmetric keys, store them on a local device, and manage access to the keys. To sum up, MetaMask is a super-safe extension

Extended functions set for MetaMask clone

To help your MetaMask wallet clone become famous, you should add some advantages that highlight it from the competitors and improve the user experience.

These can be the following:

Linking an account. Your users will find it useful to be able to buy a cryptocurrency and exchange it for fiat within the wallet. This will be possible if you develop a wallet like MetaMask and add the feature of linking bank accounts, credit/debit cards, PayPal, or other online payment systems.

eCommerce integrations. We mean integrating the wallet with exchanges, NFT marketplaces, decentralized applications, shops, and other services that the users might find useful.

Multilingual interface. If you focus on a market where all people speak the same language, you might neglect this aspect. However, your intentions are global, and you should add as many languages as possible to increase the target audience.

Push notifications. The notifications will inform the users of receiving payments, ending transactions, rapid exchange rate changes in the investment account, system updates, suspicious activity, etc.

VIP support. Numerous cryptocurrency trading platforms offer support for an additional fee. This may include 24/7 support, communication with a personal specialist, etc.

QR scanner. This is another useful feature that allows your users to make payments even faster. Moreover, it will decrease the number of transfers done by mistake.

Conclusion: We have successfully created our own wallet using Metamask for crypto transactions.

Assignment No.13

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

- Deposit money
- Withdraw Money
- Show balance

Objectives:

1. Concept of Smart Contract
2. for Bank account of a customer

Theory:-

Writing a Banking Contract

This article will demonstrate how to write a simple, but complete, smart contract in Solidity that acts like a bank that stores ether on behalf of its clients. The contract will allow deposits from any account, and can be trusted to allow withdrawals only by accounts that have sufficient funds to cover the requested withdrawal. This post assumes that you are comfortable with the ether-handling concepts introduced in our post, Writing a Contract That Handles Ether.

That post demonstrated how to restrict ether withdrawals to an —owner‘s‖ account. It did this by persistently storing the owner account‘s address, and then comparing it to the `msg.sender` value for any withdrawal attempt. Here‘s a slightly simplified version of that smart contract, which allows anybody to `deposit` money, but only allows the `owner` to make withdrawals:

```
pragma solidity ^0.4.19;

contract TipJar {

    address owner;    // current owner of the contract

    function TipJar() public {
        owner = msg.sender;
    }

    function withdraw() public {
        require(owner == msg.sender);
        msg.sender.transfer(address(this).balance);
    }

    function deposit(uint256 amount) public payable {
        require(msg.value == amount);
    }

    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }
```

```
}
```

Maintaining Individual Account Balances

I am going to generalize this contract to keep track of ether deposits based on the account address of the depositor, and then only allow that same account to make withdrawals of that ether. To do this, we need a way keep track of account balances for each depositing account—a mapping from accounts to balances. Fortunately, Solidity provides a ready-made mapping data type that can map account addresses to integers, which will make this bookkeeping job quite simple. (This mapping structure is much more general key/value mapping than just addresses to integers, but that's all we need here.)

Here's the code to accept deposits and track account balances:

```
pragma solidity ^0.4.19;

contract Bank {

   mapping(address => uint256) public balanceOf;   // balances, indexed by addresses

   function deposit(uint256 amount) public payable {
      require(msg.value == amount);

      balanceOf[msg.sender] += amount;     // adjust the account's balance
   }
}
```

Here are the new concepts in the code above:

- mapping(address => uint256) public balanceOf; declares a persistent public variable, balanceOf, that is a mapping from account addresses to 256-bit unsigned integers. Those integers will represent the current balance of ether stored by the contract on behalf of the corresponding address.
- Mappings can be indexed just like arrays/lists/dictionaries/tables in most modern programming languages.
- The value of a missing mapping value is 0. Therefore, we can trust that the beginning balance for all account addresses will effectively be zero prior to the first deposit.

It's important to note that balanceOf keeps track of the ether balances assigned to each account, but it does not actually move any ether anywhere. The bank contract's ether balance is the sum of all the balances of all accounts—only balanceOf tracks how much of that is assigned to each account. Note also that this contract doesn't need a constructor. There is no persistent state to initialize other than the balanceOf mapping, which already provides default values of 0.

Withdrawals and Account Balances

Given the balanceOf mapping from account addresses to ether amounts, the remaining code for a fully-functional bank contract is pretty small. I'll simply add a withdrawal function:

bank.sol

```
pragma solidity ^0.4.19;
```

```
contract Bank {

   mapping(address => uint256) public balanceOf;   // balances, indexed by addresses

   function deposit(uint256 amount) public payable {
      require(msg.value == amount);
      balanceOf[msg.sender] += amount;     // adjust the account's balance
   }

   function withdraw(uint256 amount) public {
      require(amount <= balanceOf[msg.sender]);
      balanceOf[msg.sender] -= amount;
      msg.sender.transfer(amount);
   }
}
```

The code above demonstrates the following:

- The require(amount <= balances[msg.sender]) checks to make sure the sender has sufficient funds to cover the requested withdrawal. If not, then the transaction aborts without making any state changes or ether transfers.
- The balanceOf mapping must be updated to reflect the lowered residual amount after the withdrawal.
- The funds must be sent to the sender requesting the withdrawal.

Important: Avoiding the Reentrancy Vulnerability

In the withdraw() function above, it is very important to adjust balanceOf[msg.sender] **before** transferring ether to avoid an exploitable vulnerability. The reason is specific to smart contracts and the fact that a transfer to a smart contract executes code in that smart contract. (The essentials of Ethereum transactions are discussed in How Ethereum Transactions Work.)

Now, suppose that the code in withdraw() did not adjust balanceOf[msg.sender] before making the transfer *and* suppose that msg.sender was a malicious smart contract. Upon receiving the transfer—handled by msg.sender‘s fallback function—that malicious contract could initiate *another* withdrawal from the banking contract. When the banking contract handles this second withdrawal request, it would have already transferred ether for the original withdrawal, but it would not have an updated balance, so it would allow this second withdrawal!

This vulnerability is called a ―reentrancy‖ bug because it happens when a smart contract invokes code in a different smart contract that then calls back into the original, thereby reentering the exploitable contract. For this reason, it‘s essential to always make sure a contract‘s internal state is fully updated before it potentially invokes code in another smart contract. (And, it‘s essential to remember that every transfer to a smart contract executes that contract‘s code.)

Conclusion: We have studied a smart contract on a test network for Bank account of customer operations.

Assignment No.14

Aim: Write a program in solidity to create Student data. Use the following constructs:

- Structures
- Arrays
- Fallback

  Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.

Objectives:

1. Concept of solidity
2. smart contract on Ethereum and Observe the transaction fee and Gas values

Theory:

Introduction

Blockchain is a decentralized, distributed public ledger that lets us collaborate and coordinate the members that do not trust each other to make a secure transaction. Many of you understand blockchain as a bitcoin, but bitcoin is a cryptocurrency that takes the help of blockchain technology to operate.

Blockchain Technology

First, when Windows was launched to create reports and work on any project, we used MS word, where only one person could edit at a time and then send to another, and the process goes on, one after the other method. After some technological evolution, we have seen Google docs, google sheets in the market where online multiple people can work on a single document simultaneously. But the problem here is that it works on centralized architecture where a single server maintains google docs and various nodes are connected. Still, if the significant server crashes or gets corrupt, all the nodes get disconnected, and all the work gets destroyed. So the solution to this is decentralized blockchain technology. Decentralized means that no single server or single node is managing the network. Data is replicated to multiple nodes so that if any node goes down, other nodes operate as it is, and original data can quickly be recovered.

What is Ethereum?

Ethereum is a decentralized blockchain designed to be highly secure, fault-tolerant, and programmable. Ethereum blockchain is a choice for many developers and businesses. As said programmable, the main task of Ethereum is to securely execute and verify the application code known as smart contracts. Ethereum helps to build native scripting language(solidity) and EVM. Ethereum consensus mechanism is proof of work to operate to verify the new transaction. Now we will learn about smart contracts and how it runs on the Ethereum platform.

Overview of Smart Contracts

A smart contract is a small program that runs on an Ethereum blockchain. Once the smart contract is deployed on the Ethereum blockchain, it cannot be changed. To deploy the smart contract to Ethereum, you must pay

the ether (ETH) cost. Understand it as a digital agreement that builds trust and allows both parties to agree on a particular set of conditions that cannot be tampered with.

Learn | Write | Earn

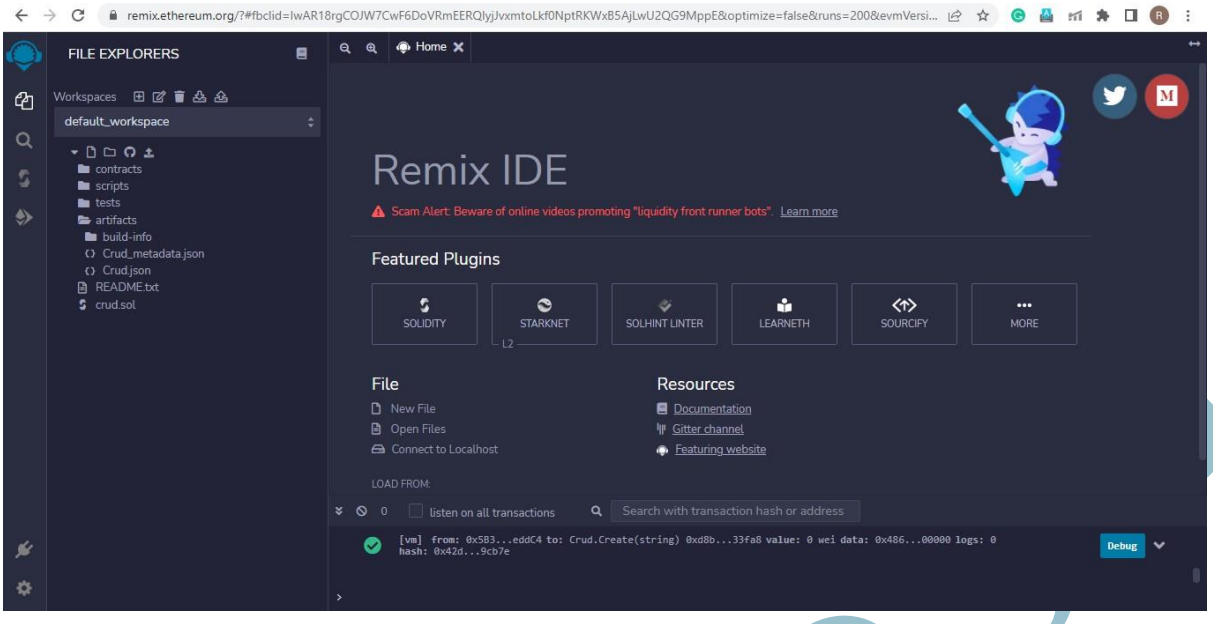
To understand the need for a smart contract, suppose there was one grocery shop, and ram went to buy some groceries. He purchased the groceries for 500 rupees and kept on debt that would pay the money next month when he returned, so the shopkeeper jotted down his purchase in his ledger. In between the period somehow shopkeeper changed 500 to 600 and when next month ram went to pay the money, the shopkeeper has demanded 600 INR and ram has no proof to show that he has only bought 500 INR so in this case, smart contracts play an essential role which prevents both the parties to tamper the agreement and only gets terminate when all the conditions satisfy after the deal. There are a couple of languages to write smart contracts, but the most popular is solidity.

Introduction to Solidity Programming

Solidity is object-oriented, high-level statically-typed programing language used to create smart contracts. Solidity programming looks similar to Javascript, but there are a lot of differences between both languages. In solidity, you need to compile the program first, while in Javascript, you can run the program directly in your browser or by using Node JS. With solidity, you can create contracts for uses such as voting, crowdfunding, blind auctions, and multi-signature wallets. It is also a case-sensitive programming language. Visit the official solidity documentation to read more and be updated about any new functionality release.

What is Remix IDE?

It is an online IDE for creating solid, smart contracts, so you do not need to install or download anything to do any setup. You can develop, deploy, and Administer your solidity smart contract using Remix IDE. Visit this link to access the Remix IDE where you will find multiple options and a window shown below. The window is a little bit similar to VS code where on the left-hand side, you will find some icons to terminate to other options like a compiler, file explorer, search files, deploy, etc.



Using the file explorer, you can create and open any file. We can set the compiler version, run our smart contract, and observe the output using the compiler. Each compiler type provides a different amount of fake ethers used for practicing purposes. **Solidity Compilation Process** Smart contract compilation is a critical process to understand how a smart contract runs when created using solidity. We will understand the process using the below flow chart. We can see that the smart contract written in solidity with sol extension first gets the compiler version. After it goes under the compiler, It gets split into two parts where one is Byte code, and the other is ABI (Abstract Binary Interface) key. Byte code is only executed and deployed on the Ethereum blockchain, not the complete smart contract. Whenever any smart contract wants to communicate with this smart contract, they need the ABI key to call functions and variables.

To observe how ABI and Byte code is generated on Remix IDE, visit IDE, open any contract in the contracts folder, and compile and run it. Scroll down, and you will find two options: ABI and Byte code, where you can copy and paste them into any notepad and observe how your code gets converted to Byte code.

To create a smart contract, the first thing is to define the compiler version to use using the Pragma keyword (you can also determine whether the program supports multiple versions or the version in a particular range); after this, you define the contract using the contract keyword which is same as creating a class in object-oriented programming.

Important points related to smart contract Compilation

1. Contract Bytecode is public in readable form – It means It does not get encrypted because It will run on different nodes of Ethereum. For then, It needs to decrypt again and again not to increase computation time. It is kept in a readable form.
2. The contract doesn‘t have to be public – It does not need to keep contracts public, but most organizations keep them public to maintain the trust.
3. Bytecode is immutable

4. ABI act as a bridge between application and smart contract
5. ABI and bytecode cannot be generated without source code

State and Local Variables in Solidity

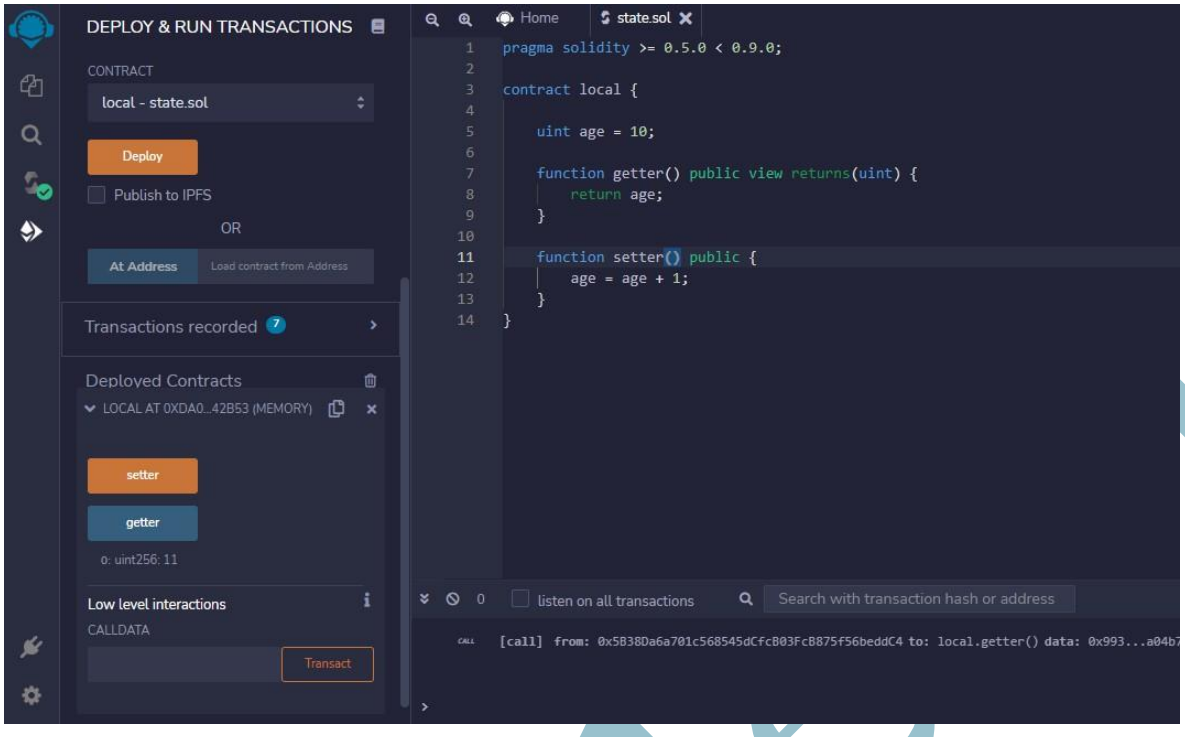
Any variable declared on the contract level is known as a state variable. The critical property of the state variable is that it is permanently stored in the blockchain, so you have to pay some amount of gas and use the state variable with care. Solidity does not have a concept of Null or None; indeed, each data type has one default value which on declaration is assigned to that variable. To define Public before any variable or function, automatically, one get function is set with that variable, and you can access its value. Storage to state variable is not dynamically allocated (To initialize state variable with the value, you need to assign a value at declaration time, use constructor, use getter and setter functions). An instance of a contract variable cannot have another state variable besides those already declared.Local variables are those variables that are declared in the function body and are stored in a stack, not in contract storage. Local variables don‘t cost gas; some types reference the storage by default. Memory keywords cannot be used at the contract level.

Functions in Solidity

Functions are an essential part of any programming language for the reusability of a particular code. We will see the getter and setter function in solidity to learn how to create a function in solidity. The getter function is a function from which we can access the value of our variables. It is a view-only function, so we can define it as a view or Pure, which states that the value of the state variable cannot be changed it returns the variable‘s value, so we define the return type of value. On the other side, the setter function changes the value, so it is a simple public function.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract local {
uint age = 10;
function getter() public view returns(uint) {
return age;
}
function setter() public {
age = age + 1;
}
}
```

After writing the above code on Remix IDE in a new file with sol extension, you can compile the code, visit the deploy section, and deploy the code to observe the deploy section output as shown below. The value will increase as you click the setter and getter function buttons.



Suppose you want to implement the setter function to set the new value of age so we can pass the parameter in the setter function and set the value of age. Thus in the setter function, we change the matter, so we need to pay a certain amount of gas in the setter function, but the getter function is view-only and does not require any amount of gas to be paid. By default, the visibility of a function is private, so to make it public, we define a function as Public.

Pure and View in Solidity

We have seen that we use to view and pure where we are not updating the state variable. But pure, you cannot use where it is also reading the state variable. Pure is used where both reading and writing are not performed. Indeed in View, reading is allowed, but writing is not permitted. When we do not define any one of the following to a function, it simply warns that we can provide one restriction of pure or View to function.

Constructor in Solidity

A constructor is a particular type of function which executes only once when you create your contract. Constructor is used to work with state variables and define smart contracts‘ owners. You can create only one constructor, and it is optional to create. The compiler creates a default constructor if there is no explicitly defined constructor.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract local {
   uint        public        count;
   constructor(uint new_count) {
      count = new_count;
   } }
```

In the above code, we have created a constructor, and before clicking on deploy, you need to define the value of the constructor before it is called only once, so enter the value and click on deploy, and on scrolling, you can observe the value of count.

Control Statements in Solidity

All programming languages have control statements that help us check multiple conditions using loops and an if-else ladder, and solidity also supports loops and if-else statements.

Loops in Solidity

Solidity also supports three loops: a while loop, a Do while loop, and a loop. If you are familiar with any other programming language, you must know about control statements and using loops to run particular code multiple times with different values. In solidity, you cannot write the loops directly in contract storage; instead, you must declare them in any function.

While the loop runs a code snippet multiple times until the condition is proper, the loop terminates when the condition is false. In contrast, the loop runs 0 or multiple times.

Do while loop is a loop that runs even one time when the condition in the while loop is false. So it is used when you need to run a particular code at least once and if certain conditions meet, then run it multiple times.

For loop is a loop that is used when you know the start and end time of the loop and how many intervals you need to take. For loop, the initialization and iterator updating are part of loop syntax.

So let us look after the syntax of each type of loop using a sample program.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract Loops {
    uint [3] public arr;
    uint public count;
    function Whileloop() public {
        while(count < arr.length) {
            arr[count] = count;
            count++;
    }     }
    function Forloop() public {
        for(uint i=count; i<arr.length; i++) {
            arr[count] = count;
            count++;
        }      }
    function doWhileLoop() public {
        do {
            arr[count] = count;
```

```
        count++;
    }while(count < arr.length);
} }
```

If-else Statements in Solidity

If-else statements are an essential part of any programming language that helps compare two or more two types of values to make a particular decision. Below is the sample code snippet denoting the use of if-else in the solidity that you should try and deploy the contract. After deploying, check by entering the different values.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract Array {
    function check(int a) public pure returns(string memory) {
        string memory value;
        if(a > 0) {
            value = "Greater Than zero";
        }
        else if(a == 0) {
            value = "Equal to zero";
        }
        else {
            value = "Less than zero";
        }
        return value;
    }  }
```

Arrays in Solidity

The array is a special data structure used to create a list of similar type values. The array can be of fixed size and dynamic-sized. With the help of index elements can be accessed easily. below is a sample code to create, and access a fixed-sized array in solidity.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract Array {
    uint [4] public arr = [10, 20, 30, 40];
    function setter(uint index, uint value) public {
        arr[index] = value;
    }
    function length() public view returns(uint) {
        return arr.length;
```

```
    }   }
```

You can compile and deploy the code to try changing the array elements with an index and printing the array length.

Creating Dynamic Array

A dynamic array is an array where we can insert any number of elements and delete the details easily using an index. So solidity has functions like push and pops like python, making it easy to create a dynamic array. Below is a code using which you can create a dynamic array. After writing code, compiles and deploy the code by visiting the deploy section in the left-side navigation bar. After that, try inserting and deleting some elements from an array.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract Array {
    uint [] public arr;
    function PushElement(uint item) public {
        arr.push(item);
    }
    function Length() public view returns(uint) {
        return arr.length;
    }
    function PopElement() public {
        arr.pop();
    }   }
```

Structure in Solidity

The structure is a user-defined data type that stores more than one data member of different data types. As in array, we can only store elements of the same data type, but in structure, you can keep elements of different data types used to create multiple collections. The structure can be made outside and inside the contract storage, and the Structure keyword can be used to declare the form. The structure is storage type, meaning we use it in-store only, and if we want to use it in function, then we need to use the memory keyword as we do in the case of a string.

```
pragma solidity >= 0.5.0 < 0.9.0;
struct Student {
    uint  rollNo;
    string name;
}
contract Demo {
    Student public s1;
```

```
    constructor(uint _rollNo, string memory _name) {
        s1.rollNo = _rollNo;
        s1.name = _name;
    }
    // to change the value we have to implement a setter function
    function changeValue(uint _rollNo, string memory _name) public {
        Student memory new_student = Student( {
            rollNo : _rollNo,
            name : _name
        });
        s1 = new_student;
    }   }
```

Create a Smart Contract with CRUD Functionality

We have excellent theoretical and hands-on practical knowledge about solidity, and now you can create a primary smart contract like hello world, getter, and setter contracts. So it's a great time to try making some functional smart contracts, and the best way to try all the things in one code is to create one program that performs all CRUD operations.

```
pragma solidity ^0.5.0;
contract Crud {
    struct User {
        uint id;
        string name;
    }
    User[] public users;
    uint public nextId = 0;
    function Create(string memory name) public {
        users.push(User(nextId, name));
        nextId++;
    }
    function Read(uint id) view public returns(uint, string memory) {
        for(uint i=0; i<users.length; i++) {
            if(users[i].id == id) {
                return(users[i].id, users[i].name);
            }       }       }
    function Update(uint id, string memory name) public {
```

```
    for(uint i=0; i<users.length; i++) {
        if(users[i].id == id) {
            users[i].name =name;
        }       }       }
    function Delete(uint id) public {
        delete users[id];
    }
    function find(uint id) view internal returns(uint) {
        for(uint i=0; i< users.length; i++) {
            if(users[i].id == id) {
                return i;
            }       }
        // if user does not exist then revert back
        revert("User does not exist");
    }   }
```

Conclusion: We have studied program writing in solidity to create Student data by using constructs like structures, arrays, fallback and also deployed this as smart contract on Ethereum and Observe the transaction fee and Gas values.

Assignment No.15

Aim: Write a survey report on types of Blockchains and its real time use cases.

Objectives: Student will be able to learn

1. Concept of types of Blockchains
2. survey report on types of Blockchains and its real time use cases

Theory:

Blockchain & Types

Blockchain technology is being used to carry and transfer the transactions or exchange of information through a secure network. Blockchain technology and distributed ledger technology is used parallel to the digital cryptocurrency to the people. Blockchain is being used for the purpose of private networking and uses too where only the restricted network users can get the authorization and access. Here network administrators are authorized to administrate the activities and any new nodes or users who wish to get permission, need to contact with the system or network administrators. Primarily there are two types of Blockchain technology viz. private Blockchain and public Blockchain. Though based on some other criteria and analysis Blockchain technology can also be noted and called as consortium blockchain technology, and hybrid blockchain technology. It is important to note that every kind of Blockchain basically consists of a cluster of nodes, and this is working on the peer-to-peer (P2P) network system.

Every node in the network has a copy of the shared ledger and further that is timely updated and also being verified the transactions, initiate and receive transactions. Keeping in mind the broad nature, experts classified Blockchain Technology into following three

Public Blockchain,

Private Blockchain, and

Hybrid Blockchain.

1. **Public Blockchain** is a major type of Blockchain, and that is not only open but also decentralized in nature. And in this type of Blockchain technology computer networks are basically accessible to anyone interested in transactions. Here based on validation the validated person basically receives the transaction rewards and furthermore, two kinds of Proof-of-work and Proof-of-stake models are being used. The Public Blockchain is furthermore a non-restrictive and distributed ledger system which is doesn‘t seek any kind of permission, and anyone having access can be authorized one to get the data or part pf the Blockchain. This kind of Blockchain also gives authorization regarding the current and past records verification. Additionally, this is being used for mining and exchanging cryptocurrencies

In this segment most common is Bitcoin and Litecoin blockchains. It is mostly secure upon following strict security rules as well as methods. However, upon non-following the security protocols it may be risky. Some of the examples of this type of Blockchain are - Bitcoin, Ethereum, and Litecoin.

Here given figure depicted some of the features and advantages regarding Public Blockchain Systems and Technology.

Fig. Salient features and functions of the public blockchains

There are two common examples of public blockchains and these are Bitcoin and Ethereum as per the experts This type of Blockchain is concerned with the following type of features viz.

High Security and Privacy,

- Open and Flexible Environment,
- Anonymous Nature,
- No regulations and strict Policies,
- Full Transparency and Systems.
- Distributed, etc.

However, according to the experts, the following are being considered as important advantages and benefits of the Public Blockchain.

Trustable and Faith

Public Blockchainis trusted and here unlike private blockchains, the participants are don‘t need to think of authenticity. In this type of Public Blockchain, they no need of knowing other nodes, and therefore there is no fraud in the transactions. In this category nodes can contact blindly without feeling the needing to trust individual nodes

Secure and Safe

In the Public Blockchain,there are opportunities in connecting with the other participants and nodes in the same public network, and this results in secure, largest, and greater communication and participation. Owing

to this feature, it is difficult for the attackers to enter the systems and here every node will do the mverifications and transactions as per norms. Here thoughtful cryptogenic encrypting methods are being used and therefore it is much safer than the private blockchain according to some experts.

Open and Transparent

Public Blockchain is also having the features of openness and here data is basically transparent to all the nodes and in this mechanism, one blockchain record is normally available to all the authorized nodes.Therefore, here all the nodes become open and transparent and there is an absence of fake transactions or hiding any information. Though there are plenty of advantages and benefits but it is also having a different kind of disadvantages and weaknesses, and some of them are as under.

Lower Transaction per Second

In Public Blockchain System the rate of transaction per second is also very low, and this is due to having a large number of nodes and huge network. Here each node has to verify the transaction and also do proof-of-work is time consuming. Here in public systems seven (07) transactions happen per second and additionally, here Ethereum network has about a 15 TPS rate.

Scalability Matters

Similar to the previously mentioned issue on a lower transaction per second in public blockchain another issue is scalability according to the experts. The huge size basically creates the scalability in this regard and here bitcoins lightning networks are considered as important to overcome the problem according to the experts.

High Energy Consumption

The public blockchain also suffers from higher energy consumption due to the proof-of-work energy consumption. As it needs special algorithms therefore high energy consumption is considered as important in energy, environment, and financial context.

2. Private blockchains

Private Block Chain are restricted and not open, such kind of blockchain also has features of access. This blockchain allows permission for the transaction from the support of the system administrator

Private blockchain solutions develop these platforms having the features of the following—

ˆ ˆ Full of privacy,

ˆ ˆ High efficiency,

ˆ ˆ Faster transaction,

ˆ ˆ Better scalability,

ˆ ˆ Faster and speediness.

This type of blockchain is works on closed systems and networks only and these are usually useful in the organizations, enterprises from which only selected members can be joined. This type of blockchain contains proper security, authorizations, permissions as well as accessibility. According to the experts, private

blockchains are deployed forvoting, regarding supply chain management, for finding and managing digital identity, regarding asset ownership,and so on. There are certain popular private blockchains like Multichain, Hyperledger projects, Corda, etc. Participants are

Fig.: Salient features and functions of the private blockchains

Private blockchains as running with the authorized nodes; therefore no one from the outside of the private network is able in accessing information and transaction related data exchanged between two nodes.

3. **Hybrid Blockchains** is a merger of public blockchain as well as private blockchain and it is required in better control for achieving higher goals. Hybrid Blockchain deals with centralized and decentralized systems and it is not open; however, it has the features of integrity, transparency, as well as security. It has several advantages over traditional blockchains as depicted in Fig. 5. In Hybrid Blockchains maximum customization is being considered as main benefits with private permission-based system as well as a public permission-less system. In this type of blockchain systems users are able in getting access and selected sections and rest can be recorded or keeps safe due to the benefits of the records from the ledger. Hybrid Blockchains is flexible enough so that users can join easily as private blockchain. This type of blockchain is able in enhancing the security and transparency of the blockchain network‘

4. **Consortium Blockchain** is another type of semi-decentralized type of blockchain, and this type of blockchain is able in the organization of managing the blockchain network. This type of blockchain is able in doing activities even from a single organization. Here blockchain is able in exchange information or do the mining and are being used in the areas such as banks, government organizations, etc. Some of the examples of this type of consortium are Energy Web Foundation, R3, etc.

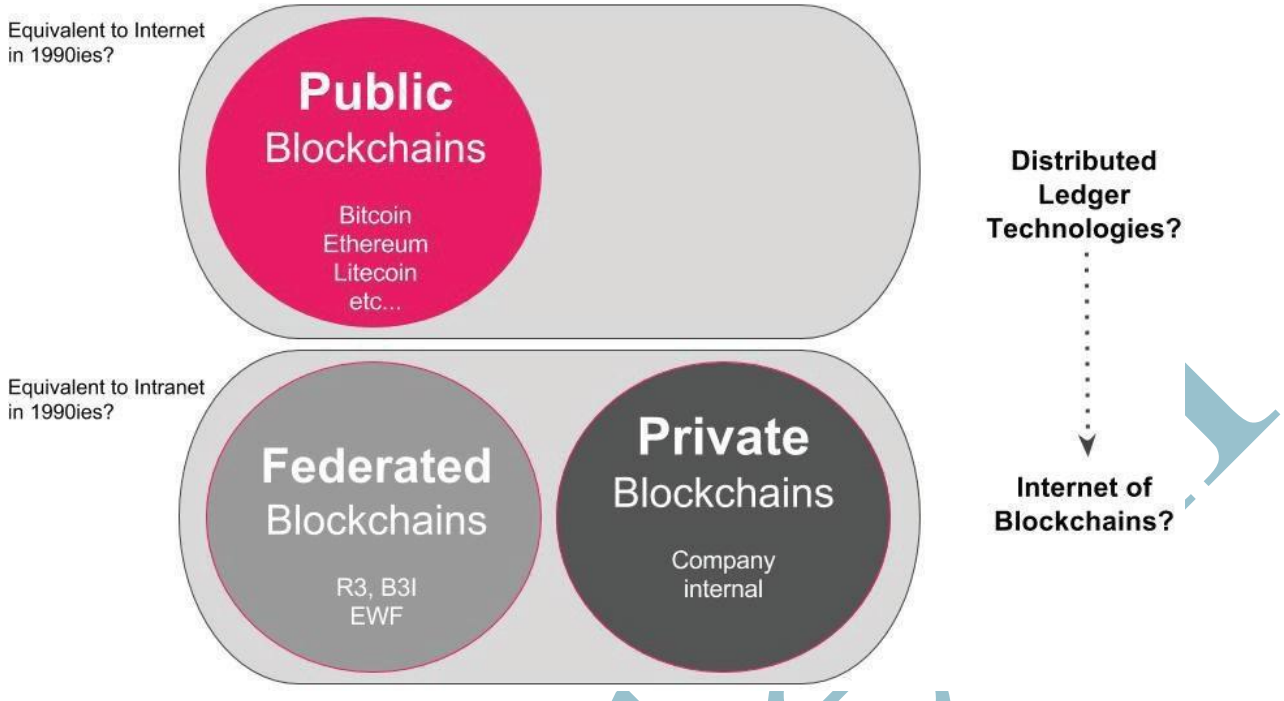


Fig. Types of Blockchains and roadmaps

Therefore, in a nutshell, all the Blockchains are having their own benefits and advantages and as awhole public and private is considered as major or worthy in terms of operations (as depicted in Fig. According to the experts security, scalability, and transparency are considered as worthy and main points in the Blockchains of public and private types. It is important to note that private blockchains are not trustworthy; while the public network is important in proof-of-work based.

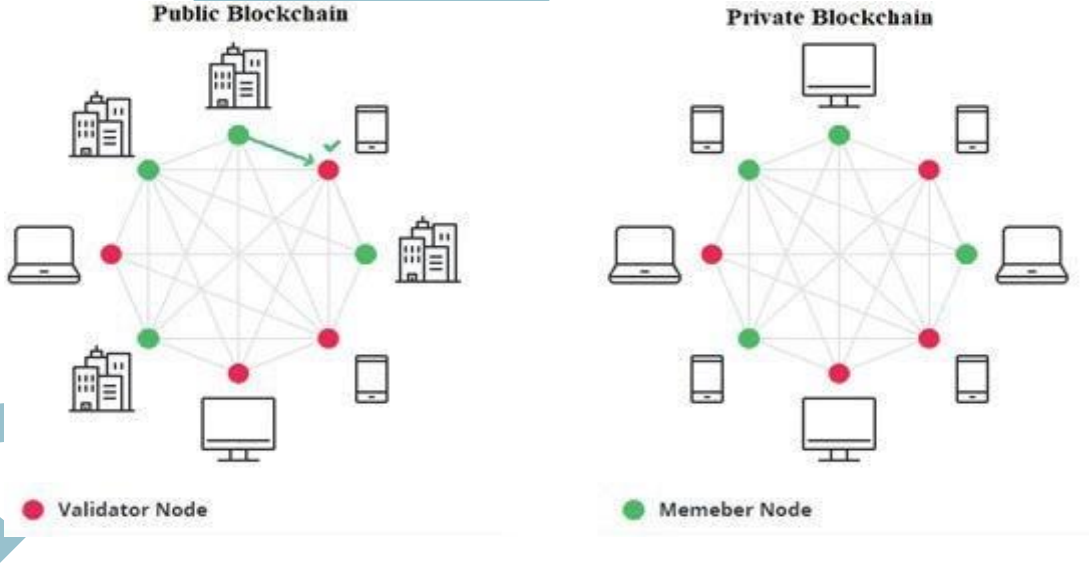


Fig. Architecture wise differences in public and private blockchains

Top blockchain use cases

Blockchain is "a general-purpose technology, which means it is applicable across sectors," said Christos Makridis, a research professor at Arizona State University, senior adviser at Gallup, digital fellow at Stanford University's Digital Economy Lab and CTO at arts and education technology startup Living Opera. "For

example, financial services can use it to write smart contracts between consumers and their banking institution. Similarly, healthcare can use it to write smart contracts between insurers and hospitals, as well as between patients and hospitals. The possibilities are endless." Blockchain use cases continue to expand. Here are some common commercial applications:

1. **Smart contracts.** The primary function of computer programs called "smart contracts" is to automate the execution of contract terms when conditions warrant them. The computer code follows a relatively simple command of "when/if _then__" to ensure that all parties receive the benefits or penalties as the contract stipulates and actions require. Smart contracts are useful to, and used by, most industries today for a variety of uses traditionally governed by paper contracts. The blockchain also makes a permanent record of every action and reaction in the transaction.
2. **Cybersecurity.** Blockchains are highly secure because of their permanency, transparency and distributed nature. With blockchain storage, there's no central entity to attack and no centralized database to breach. Because blockchains are decentralized, including those privately owned, and the data stored in each block is unchangeable, criminals can't access the information. "Essentially, the intruder needs keys to many different locations versus just one," Makridis noted. "The computing requirements for the intruder grow exponentially."
3. **IoT.** Two primary IoT uses of blockchains are in the supply chain sector and for asset tracking and inventory management. A third use is in recording measurements made by machines whether those sensors are in the Artic, the Amazon jungle, a manufacturing plant or on a NASA drone surveying Mars. "Whether it be reports of chemical data regarding oil grades or tracking shipments of electronics across the world through various ports of entry, the blockchain can be utilized anywhere there is data interacting with the real world," explained Aaron Rafferty, CEO of cryptocurrency investment firm R.F. Capital.
4. **Cryptocurrencies.** The blockchain concept was originally developed to manage digital currencies such as bitcoin. While the two technologies still compete against each other in alternative transactions, they've also been separated so blockchains could serve other purposes. Given the anonymity of crypto coins, blockchain is the only way to document transactions with accuracy and privacy for the parties involved.
5. **NFTs.** Nonfungible tokens are units of data certified to be unique and not interchangeable. In short, they are digital assets. According to Rafferty, NFTs are revolutionizing the digital art and collectibles world. "We are using decentralization and the ethereum blockchain to create a music live stream network where artists and streamers can connect with fans directly, sell their NFTs, receive contributions from fans and trade in their rewards and contributions for crypto tokens," said Shantal Anderson, founder and CEO of music and pop culture streaming network Reel Mood.

Conclusion: We have studied the survey report on types of Blockchains and its real time use cases.

Assignment No.16

Aim: Mini-project on Design & Analysis of Algorithm.

Assignment No.17

Aim: Mini-project on Machine Learning.

Assignment No.18

Aim: Mini-project on Blockchain Technology.