Pravara Rural Education Society's

Pravara Rural Engineering College, Loni

Tal: - Rahata, Dist: - Ahmednagar, Loni-413736 (M.S.)

Lab Manual

410246: Laboratory Practice III

Fourth Year of Computer Engineering (2019 Course)

Department of Computer Engineering

LOKNETE DR. BALASAHEB VIKHE PATIL
(PADMA BHUSHAN AWARDEE)
PRAVARA RURAL EDUCATION SOCIETY'S
PRAVARA RURAL ENGINEERING COLLEGE
LONI

Department of Computer Engineering

BE (Computer Engineering 2019 Course) Semester –I (2022-23)

Subject: 410246: Laboratory Practice III

Manual Content

Sr. No.	Particulars
A1	Program to calculate Fibonacci numbers and find its step count.
A2	Implement job sequencing with deadlines using a greedy method.
A3	Fractional Knapsack problem using a greedy method.
A4	0-1 Knapsack prob^{m} using dyn^{c} prog^{g} or branch and bound strategy.
A5	Generate binomial coefficients using dynamic program^{g}.
A6	N-Queens matrix. Use backtracking to place remaining Queen.
A7	Mini Project.
B1	Predict the price of the Uber ride.
B2	Classify the email using the binary classification method.
B3	Build a neural network-based classifier for deter^{e} bank costomer.
B4	Implement Gradient Descent Algorithm to find the local minima.
B5	Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.
B6	Implement K-Means clustering/ hierarchical clustering.
B7	Mini Project.
C1	Installation of Metamask and study spending Ether per transaction.
C2	Create your own wallet using Metamask for crypto transactions.
C3	Smart contract on a test network, for Bank account of a customer.
C4	Program in solidity to create Student data.
C5	Mini Project.

Verified by:

Academic Coordinator HOD

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

About College

Pravara Rural Engineering College, Loni, started on 21"August 1983, is the premier of all the institutions under the technical education complex developed by the Pravara Rural Education Society. The Society was established in 1964 by Late Padmashri Dr.Vitthalrao Vikhe Patil, who was also founder of the Pravara Sahakari Sakhar Karkhana Ltd, Pravaranagar. The Society has been registered under the Societies Registration Act 1960 and the Mumbai Public Trust Act 1950.

Vision of Institute

Enrich the youth with skills and values to enable them to contribute in the development of society: nationally and globally.

Mission of Institute

To provide quality technical education through effective teaching-learning and research to foster youth with skills and values to make them capable of delivering significant contribution in local to global development.

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

About Department

Computer Engineering Department established in 1985 with the vision to produce world class quality computer engineers. PG in Computer Engineering started in 2014. Since its inception the Department has gained reputation in the S P Pune University as a renowned department for its quality in academics and treating students and alumni as Ambassadors of institute. More than 1800 alumni are working in India and abroad in leading multinational companies, government and various research organizations such as IBM, CAPGEMINI, SYNTEL, TCS and renowned top industries. The department has an environmental friendly infrastructure, well equipped laboratories and qualified, experienced staff.

Vision of Department

To develop Computer Engineering graduates capable of exhibiting leadership, creativity and skills for improving the quality of life.

Mission

To provide a platform for self-learning to meet the challenges of changing technology and leadership qualities to succeed in professional career.

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Program Outcomes:

PO1	**Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and computer engineering to the solution of complex engineering problems.
PO2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex computer engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/Development of solutions:** Design solutions for complex computer engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct Investigations of complex problems:** Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool usage:** Create, select, and apply appropriate techniques, and modern engineering and IT tools including prediction and modeling to complex computer engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequent responsibilities relevant to the computer engineering practice.
PO7	**Environment and Sustainability:** Understand the computer engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to computer engineering ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex computer engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of computer engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in broadest context of technological change.

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Program Specific Outcomes:

PSO1:	The ability to understand, analyze and develop computer programs in the areas related to Algorithms, System Software, Machine Learning, Artificial Intelligence, Web Applications, Big Data Analytics and Networking for efficient design of computer based systems of varying complexity.
PSO2:	The ability to apply standard practices and strategies in software / embedded project development using open source programming tools and environment to deliver a quality product for business success.
PSO3:	The ability to employ modern computer language, environment and platforms in creating innovative career paths to be an entrepreneur and path for higher studies.

Program Educational Outcomes (PEOs):

1. To work in a multidisciplinary environment by providing solutions to real time problems.
2. To develop experience computer programmers on design and programming techniques, technologies and tools related to computer engineering.
3. To inculcate, in students, professional and ethical attitude, effective communication skills, team work skill, multidisciplinary approach and ability to relate to engineering issues to boarder social context.

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Course Outcomes and Target Set Level

Academic year: 2022-23 **Class: BE Computer**

Subject: 410246: Laboratory Practice III **2019 Pattern**

Course Outcome's (CO): Course outcomes are the statements of what a student should know, understand and/or be able to demonstrate after completion of a course.

Course Outcome's	Description of COs
410246.1	**CO1: Apply (L3: Apply)** preprocessing techniques on datasets.
410246.2	**CO2: Implement (L3: Apply) and evaluate (L5: Evaluate)** linear regression and random forest regression models.
410246.3	**CO3: Apply (L3: Apply) and evaluate (L5: Evaluate)** classification and clustering techniques.
410246.4	**CO4: Analyze (L4: Analyze) performance of an algorithm.**
410246.5	**CO5: Implement (L3: Apply)** an algorithm that follows one of the following algorithm design strategies: divide and conquer, greedy, dynamic programming, backtracking, branch and bound.
410246.6	**CO6: Interpret (L3: Apply)** the basic concepts in Blockchain technology and its applications

Course Outcome's	PO 1	PO2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
410246.1	3	3	3	1	2	1		1	2		2	3	1	-	-
410246.2	3	3	3	2	2	1		1	2		2	3	1	-	-
410246.3	3	3	3	2	2	2		1	2		2	3	1	-	-
410246.4	3	2	2	-	1	-		1	2		2	2	1	-	-
410246.5	3	2	3	-	1	-		1	2			2	1	-	-
410246.6	3	3	2	2	2	-		1	2			2	1	-	-
410246	3	2.66	2.66	1.75	1.66			1	2		2	2.5	1	-	-

Subject teacher H.O.D

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

List of Laboratory Experiments/Assignments. Assignments from all the Groups (A, B, C) are compulsory.	
Course Contents	
Group A: Design and Analysis of Algorithms	
1	Write a program to calculate Fibonacci numbers and find its step count.
2	Implement job sequencing with deadlines using a greedy method.
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	Write a program to generate binomial coefficients using dynamic programming.
6	Design 8-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen's matrix.
7	**Mini Project** Write a program to implement matrix multiplication. Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell. Analyze and compare their performance. **OR** Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case. **OR** Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching. Observe difference in working of both the algorithms for the same input.
Group B: Machine Learning	
1	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
2	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
3	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

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

	and build the model. Identify the points of improvement and implement the same. 5. Print the accuracy score and confusion matrix (5 points).
4	Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2.
5	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
6	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
7	**Mini Project** Use the following dataset to analyze ups and downs in the market and predict future stock price returns based on Indian Market data from 2000 to 2020. Dataset Link: https://www.kaggle.com/datasets/sagara9595/stock-data **OR** Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.). Dataset Link: https://www.kaggle.com/competitions/titanic/data
Group C: Blockchain Technology	
1	Installation of Metamask and study spending Ether per transaction.
2	Create your own wallet using Metamask for crypto transactions.
3	Write a smart contract on a test network, for Bank account of a customer for following operations: Deposit money Withdraw Money Show balance
4	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.
5	Mini Project: Create a dApp (de-centralized app) for e-voting system

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Group A

Design and Analysis of Algorithms

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Experiment 1.

Write a program to calculate Fibonacci numbers and find its step count.

Aim:

Write a program non-recursive and recursive program to calculate Fibonacci numbers.

Theory:

- **Introduction to Fibonacci numbers**
  - The Fibonacci series, named after Italian mathematician Leonardo Pisano Bogollo, later known as Fibonacci, is a series (sum) formed by Fibonacci numbers denoted as Fn. The numbers in Fibonacci sequence are given as: 0, 1, 1, 2, 3, 5, 8, 13, 21, 38, . . .
  - In a Fibonacci series, every term is the sum of the preceding two terms, starting from 0 and 1 as first and second terms. In some old references, the term '0' might be omitted.
- **What is the Fibonacci Series?**
  - The Fibonacci series is the sequence of numbers (also called Fibonacci numbers), where every number is the sum of the preceding two numbers, such that the first two terms are '0' and '1'.
  - In some older versions of the series, the term '0' might be omitted. A Fibonacci series can thus be given as, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . It can be thus be observed that every term can be calculated by adding the two terms before it.
  - Given the first term, F0 and second term, F1 as '0' and '1', the third term here can be given as,

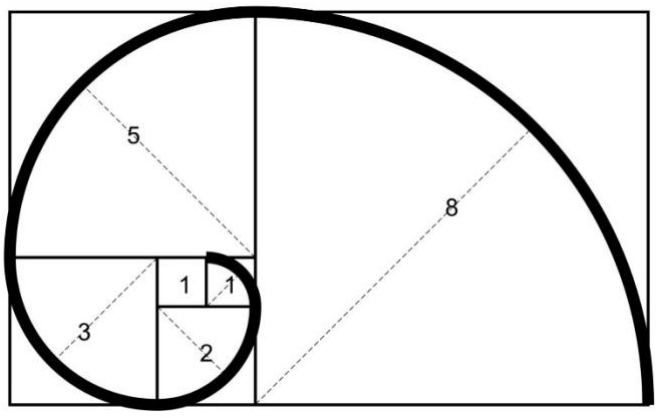
    F2 = 0 + 1 = 1

    Similarly,

    F3 = 1 + 1 = 2

    F4 = 2 + 1 = 3

Given a number n, print n-th Fibonacci Number.



Algorithm:

- **Step 1:** Start
- **Step 2:** Declare variables i, a, b , show
- **Step 3:** Initialize the variables, a=0, b=1, and show =0
- **Step 4:** Enter the number of terms of Fibonacci series to be printed
- **Step 5:** Print First two terms of series
- **Step 6:** Use loop for the following steps

  show=a+b

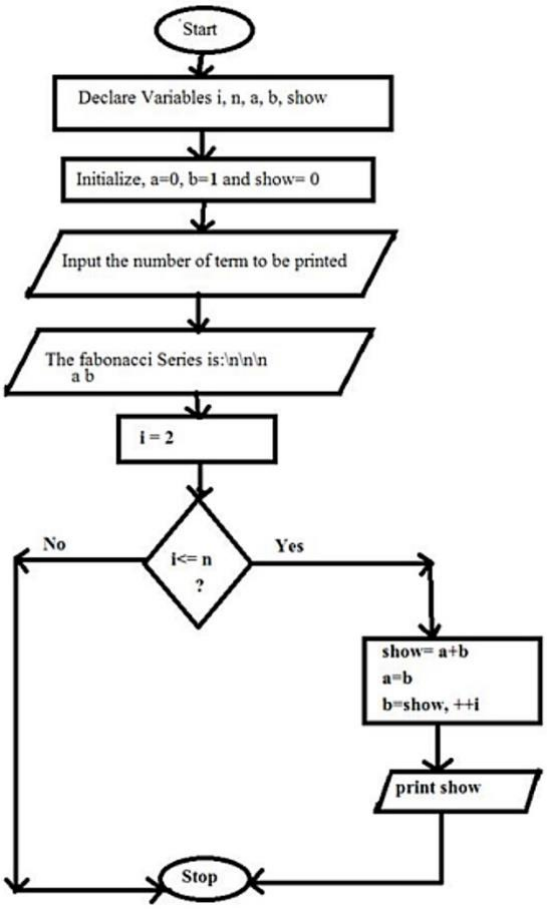
  a=b

  b=show

  increase value of i each time by 1

  print the value of show
- **Step 7:** End

Flow Chart:



Program:

```
import timeit

def fibo(n):
    # Non recursive fibonacci function
    for i in range(2, n + 1):
        fib[i] = fib[i - 1] + fib[i - 2]
    return fib[n]

def fibo_recursive(n):
    # Recursive fibonacci function
    if n == 0:
        return 0
    if n == 1:
        return 1
    fib[n] = fibo_recu(n - 1) + fibo_rec(n - 2)
    return fib[n]

N, RUNS = 20, 1000
print(f"Given N = {N}\n{RUNS} runs")

def callFibo(isRec):
     fib, fib[0], fib[1] =[0] * (N + 1), 0, 1
     print(
         "Fibonacci", "" if isRec else "non", "recursive:",
         fibo_rec(N) if isRec else fibo(N),
     "\tTime:",f'{timeit.timeit( fibo_rec(N) if isRec else fibo(N),
setup=f"from __main__ import {fibo_rec if isRec else fibo}; N={N}",
   number=RUNS
            ):5f}', "O(n)\tSpace: O(", n if else 1, ")",
     )

callFibo(True)
callFibo(False)
```

Output:

```
Given N = 20
1000 runs
Fibonacci non-recursive: 6765   Time: 0.001657 O(n)     Space: O(1)
Fibonacci recursive:     6765   Time: 2.064246 O(2^n)   Space: O(n)
```

Conclusion:

Hence, here we learned that what is Fibonacci series, and performed program on it we findout the complexity of recursive and non recursive function.

.

Experiment 2.

Implement job sequencing with deadlines using a greedy method.

Aim:

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

Theory:

- **Huffman Encoding**

  Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.

  Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.

  - Huffman Coding is a famous Greedy Algorithm.
  - It is used for the lossless compression of data.
  - It uses variable length encoding.
  - It assigns variable length code to all the characters.
  - The code length of a character depends on how frequently it occurs in the given text.
  - The character which occurs most frequently gets the smallest code.
  - The character which occurs least frequently gets the largest code.
  - It is also known as Huffman Encoding.

- **Job Sequencing With Deadlines:**

  The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.

  Here:

  - You are given a set of jobs.
  - Each job has a defined deadline and some profit associated with it.
  - The profit of a job is given only when that job is completed within its deadline.
  - Only one processor is available for processing all the jobs.
  - Processor takes one unit of time to complete a job.

- **The problem states:**

  "How can the total profit be maximized if only one job can be completed at a time?"

- **Approach to Solution:**

  A feasible solution would be a subset of jobs where each job of the subset gets completed within its deadline.

  Value of the feasible solution would be the sum of profit of all the jobs contained in the subset.

  An optimal solution of the problem would be a feasible solution which gives the maximum profit.

Greedy Algorithm:

Greedy Algorithm is adopted to determine how the next job is selected for an optimal solution.

The greedy algorithm described below always gives an optimal solution to the job sequencing problem:

- **Step 01:**

  Sort all the given jobs in decreasing order of their profit.

- **Step 02:**

  Check the value of maximum deadline.

  Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.

- **Step-03:**

  Pick up the jobs one by one.

  Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before its deadline.

Program:

```
class Node:
    """A Huffman Tree Node"""
    def __init__(self, freq_, symbol_, left_=None, right_=None):
        self.freq = freq_          # frequency of symbol
        self.symbol = symbol_      # symbol name (character)
        self.left = left_          # node left of current node
        self.right = right_        # node right of current node
        self.huff = ""             # tree direction (0/1)

def print_nodes(node, val=""):
    new_val = val + str(node.huff)
    if node.left:
        print_nodes(node.left, new_val)
    if node.right:
        print_nodes(node.right, new_val)
    if not node.left and not node.right:
        print(f"{node.symbol} -> {new_val}")

chars = ["a", "b", "c", "d", "e", "f"] # characters for huffman tree
freq = [5, 9, 12, 13, 16, 45] # frequency of characters

nodes = [Node(freq[x], chars[x]) for x in range(len(chars))]

while len(nodes) > 1:
    nodes = sorted(nodes, key=lambda x: x.freq)
    left, right, left.huff, right.huff = nodes[0], nodes[1], 0, 1
    newNode = Node(left.freq + right.freq, left.symbol +
right.symbol, left, right)
    nodes.remove(left)
    nodes.remove(right)
    nodes.append(newNode)

print("Characters :", f'[{", ".join(chars)}]')
print("Frequency                  :",        freq,        "\n\nHuffman
Encoding:")print_nodes(nodes[0])
```

Output:

```
Characters : [a, b, c, d, e, f]
Frequency  : [5, 9, 12, 13, 16, 45]
Huffman Encoding:
f -> 0, c -> 100, d -> 101, a -> 1100, b -> 1101, e -> 111
```

Conclusion:

Hence, here we studied that what is job scheduling and what is deadlock. And we learned that how to handle deadlock using greedy method.

Experiment 3.

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

Aim:

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

Theory:

- **Greedy Method:**
    - A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.
    - The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.
    - This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

- **Advantages of Greedy Approach**
    - The algorithm is easier to describe.
    - This algorithm can perform better than other algorithms (but, not in all cases).

- **Drawback of Greedy Approach**
    - As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm
    - For example, suppose we want to find the longest path in the graph below from root to leaf.

- **Greedy Algorithm**
    1. To begin with, the solution set (containing answers) is empty.
    2. At each step, an item is added to the solution set until a solution is reached.
    3. If the solution set is feasible, the current item is kept.
    4. Else, the item is rejected and never considered again.

- **Knapsack Problem**
  - You are given the following:
    - A knapsack (kind of shoulder bag) with limited weight capacity.
    - Few items each having some weight and value.
- **The problem states:**
  - Which items should be placed into the knapsack such that
    - The value or profit obtained by putting the items into the knapsack is maximum.
    - And the weight limit of the knapsack does not exceed.
- **Knapsack Problem Variants:**
  - Knapsack problem has the following two variants:
    - 1. Fractional Knapsack Problem
    - 2. 0/1 Knapsack Problem
- **Fractional Knapsack Problem:**
  - In Fractional Knapsack Problem,
    - As the name suggests, items are divisible here.
    - We can even put the fraction of any item into the knapsack if taking the complete item is not possible.
    - It is solved using the Greedy Method.
- **Fractional Knapsack Problem Using Greedy Method:**
  - Fractional knapsack problem is solved using greedy method in the following steps-
    - **Step-01:** For each item, compute its value / weight ratio.
    - **Step-02:** Arrange all the items in decreasing order of their value / weight ratio.
    - **Step-03:**

      Start putting the items into the knapsack beginning from the item with the highest ratio.

      Put as many items as you can into the knapsack.
- **Time Complexity:**

The main time taking step is the sorting of all items in decreasing order of their value / 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).

Program:

```
class ItemValue:
    # Item Value DataClass
    def __init__(self, wt_, val_, ind_):
        self.wt, self.val = wt_, val_
        self.ind, self.cost = ind_, val_

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

def fractionalKnapSack(wt, val, capacity):
    # Function to get maximum value
    iVal = [ItemValue(wt[i], val[i], i) for i in range(len(wt))]

    iVal.sort(reverse=True)    # sorting items by value

    totalValue = 0
    for i in iVal:
        curWt, curVal = i.wt, i.val
        if capacity - curWt >= 0:
            capacity -= curWt
            totalValue += curVal
        else:
            fraction = capacity / curWt
            totalValue += curVal * fraction
            capacity = int(capacity - (curWt * fraction))
            break
    return totalValue

if __name__ == "__main__":
    wt, val, capacity = [10, 40, 20, 30], [60, 40, 100, 120], 50

    maxValue = fractionalKnapSack(wt, val, capacity)
    print("Maximum value in Knapsack =", maxValue)
```

Output:

```
Maximum value in Knapsack = 240.0
```

Conclusion:

Hence, here we studied that Greedy Method, Greedy Algorithm, Knapsack, Knapsack Problem.

.

Experiment 4.

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

Aim:

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

Theory:

- **Dynamic Programming:**

  - Dynamic Programming is also used in optimization problems. Like divide-and-conquer method, Dynamic Programming solves problems by combining the solutions of subproblems.
  - Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.
  - Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. These properties are overlapping sub-problems and optimal substructure.
  - Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists.
  - For example, Binary Search does not have overlapping sub-problem. Whereas recursive program of Fibonacci numbers have many overlapping sub-problems.

- **Steps of Dynamic Programming Approach**

  Dynamic Programming algorithm is designed using the following four steps:

  - Characterize the structure of an optimal solution.
  - Recursively define the value of an optimal solution.
  - Compute the value of an optimal solution, typically in a bottom-up fashion.
  - Construct an optimal solution from the computed information.

- **Applications of Dynamic Programming Approach**

  - Matrix Chain Multiplication
  - Longest Common Subsequence
  - Travelling Salesman Problem

- **Knapsack Problem**

  You are given the following:

  - A knapsack (kind of shoulder bag) with limited weight capacity.
  - Few items each having some weight and value.

- **The problem states:**

  Which items should be placed into the knapsack such that:

  - The value or profit obtained by putting the items into the knapsack is maximum.
  - And the weight limit of the knapsack does not exceed.

- **Knapsack Problem Variants**

  Knapsack problem has the following two variants:

  - Fractional Knapsack Problem
  - 0/1 Knapsack Problem

Program:

```
def knapsack_dp(W, wt, val, n):
    """A Dynamic Programming based solution for 0-1 Knapsack problem
    Returns the maximum value that can"""
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    # Build table K[][] in bottom up manner
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w]=max(val[i-1]+K[i-1][w-wt[i-1]],K[i-1][w])
            else:
                K[i][w] = K[i - 1][w]
    return K[n][W]


val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print("Maximum possible profit =", knapsack_dp(W, wt, val, n))
```

Output:

```
Maximum possible profit = 220
```

Conclusion:

Hence, we solved a 0-1 Knapsack problem using dynamic programming or branch and bound strategy in the program.

.

Experiment 5.

Write a program to generate binomial coefficients using dynamic programming.

Aim:

Write a program to generate binomial coefficients using dynamic programming.

Theory:

- **Binomial Coefficient using Dynamic Programming**

  Computing binomial coefficient is very fundamental problem of mathematics and computer science. Binomial coefficient C(n, k) defines coefficient of the term xn in the expansion of (1 + x)n. C(n, k) also defines the number of ways to select any k items out of n items. Mathematically it is defined as,

$$C(n, k) = \frac{n!}{(n-k)!\,k!} \quad (0 \le k \le n)$$

- **Binomial Coefficient using Divide and Conquer**
  - Divide and conquer divides the problem of size C(n, k), in two sub problems, each of size C(n – 1, k – 1) and C(n – 1, k) respectively. Solution of larger problem is build by adding solution of these two sub problems.
  - Structure of binomial coefficient problem using divide and conquer approach is described as :

$$C(n, k) = C\binom{n-1}{k-1} + C\binom{n-1}{k} = C(n-1, k-1) + C(n-1, k)$$

$$C(n, 0) = C(n, n) = 1$$

Where $n > k > 0$

```
Algorithm BINOMIAL_DC (n, k)
// n is total number of items
// k is the number of items to be selected from n

if k == 0 or k == n  then
     return 1
else
     return DC_BINOMIAL(n - 1, k - 1) + DC_BINOMIAL(n - 1, k)
end
```

- **Binomial Coefficient using Dynamic Programming**
  - Many sub problems are called again and again, since they have an overlapping sub problems property. Re-computations of the same sub problems is avoided by storing their results in the temporary array C[i, j] in a bottom up manner.
  - The optimal substructure for using dynamic programming is stated as,

$$C[i, j] = \begin{cases} 1 & , \text{if } i = j \text{ or } j = 0 \\ C[i-1, j-1], + C[i-1, j] & \text{otherwise} \end{cases}$$

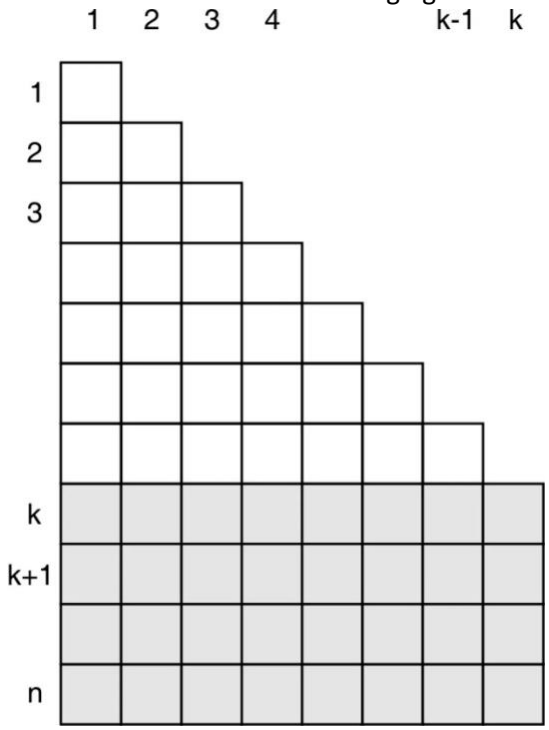
In Table, index i indicates row and index j indicates column

n\k	0	1	2	3	4	.	.	.	(k – 1)	k
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
.	.									
.	.									
k	1									1
.	.									
.	.									
n – 1	1								C(n – 1, k – 1)	C(n – 1, k)
n	1									C(n, k)

  - This tabular representation of binomial coefficients is also known as Pascal's triangle.
  - Algorithm to solve this problem using dynamic programming is shown below

```
Algorithm BINOMIAL_DC (n, k)
// n is total number of items
// k is the number of items to be selected from n

if k == 0 or k == n then
     return 1
else
     return DC_BINOMIAL(n - 1, k - 1) + DC_BINOMIAL(n - 1, k)
end
```

- **Complexity analysis**
  - The cost of the algorithm is cost of filing out the table. Addition is the basic operation. Because k ≤ n, the sum needs to be split into two parts, only the half of the table needs to be filled out for i < k and the remaining part of the table is filled out across the entire row.
  - The growth pattern of these numbers is shown in the following figure.



  - T(n, k) = sum for upper triangle + sum for the lower rectangle

$$T(n,k) = \sum_{i=1}^{k}\sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n}\sum_{j=1}^{k} 1 = \sum_{i=1}^{k}(i-1) + \sum_{i=k+1}^{n} k$$

$$= (1+2+3+\ldots+k-1) + k\sum_{i=k+1}^{n} 1 = \frac{(k-1)\,k}{2} + k\,(n-k)$$

$$= \frac{k^2-k}{2} + nk - k^2 = \frac{k^2-k+2nk-2k^2}{2} = nk - \frac{k^2}{2} - \frac{k}{2}$$

$$= nk \qquad (k << n)\ldots$$

$$T(n,k) = O(nk)$$

Conclusion:

Here, by using Dynamic programming we generate binomial coefficients.

.

Experiment 6.

Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen's matrix.

Aim:

Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen's matrix.

Theory:

- **Backtracking:**

  Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken.

  For example, in a maze problem, the solution depends on all the steps you take one-by-one. If any of those steps is wrong, then it will not lead us to the solution. In a maze problem, we first choose a path and continue moving along it. But once we understand that the particular path is incorrect, then we just come back and change it. This is what backtracking basically is.

  In backtracking, we first take a step and then we see if this step taken is correct or not i.e., whether it will give a correct answer or not. And if it doesn't, then we just come back and change our first step. In general, this is accomplished by recursion. Thus, in backtracking, we first start with a partial sub-solution of the problem (which may or may not lead us to the solution) and then check if we can proceed further with this sub-solution or not. If not, then we just come back and change it.

  Thus, the general steps of backtracking are:

  - start with a sub-solution
  - check if this sub-solution will lead to the solution or not
  - If not, then come back and change the sub-solution and continue again

- **Applications of Backtracking:**
  - N Queens Problem
  - Sum of subsets problem
  - Graph coloring
  - Hamiltonian cycles.

- **N queens on NxN chessboard**

  One of the most common examples of the backtracking is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen. A queen can attack horizontally, vertically, or diagonally. The solution to this problem is also attempted in a similar way. We first place the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left. If no safe place is left, then we change the position of the previously placed queen.

  - **N-Queens Problem:**

    A classic combinational problem is to place n queens on a n*n chess board so that no two attack, i.,e no two queens are on the same row, column or diagonal.

- **What is the N Queen Problem?**

  N Queen problem is the classical Example of backtracking. N-Queen problem is defined as,
  "given N x N chess board, arrange N queens in such a way that no two queens attack each other by being in the same row, column or diagonal".

  For N = 1, this is a trivial case.

  For N = 2 and N = 3, a solution is not possible.

  So we start with N = 4 and we will generalize it for N queens.

  - If we take n=4 then the problem is called the 4 queens problem.
  - If we take n=8 then the problem is called the 8 queens problem.

Algorithm:

- **Step 1:** Start in the leftmost column
- **Step 2:** If all queens are place return true
- **Step 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 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.

- **Step 4:** If all rows have been tried and nothing worked, return false to trigger backtracking.

Program:

```
class NQBacktracking:
    def __init__(self, x_, y_):
        self.ld, self.rd, self.cl = [0] * 30
        self.x, self.y = x_, y_ # Initial position of 1st queen
    def printSolution(self, board):
        """A utility function to print solution"""
        print(  "N Queen Backtracking Solution:",
                "Given initial position of 1st queen at row:",
                self.x, "column:", self.y, "\n",
        )
        for line in board:
            print(" ".join(map(str, line)))
    def solveNQUtil(self, board, col):
        if col >= N:
            return True
        if col == self.y:
            return self.solveNQUtil(board, col + 1)
        for i in range(N):
            if i == self.x:
                continue
            if( self.ld[i-col+N-1]!=1 and
                      self.rd[i+col]!=1) and self.cl[i]!=1 :
                board[i][col] = 1
                self.ld[i - col + N - 1]=self.rd[i + col]=self.cl[i]=1
                if self.solveNQUtil(board, col + 1):
                    return True
                board[i][col] = 0  # BACKTRACK
                self.ld[i - col + N - 1]=self.rd[i + col]=self.cl[i]=0
        return False
    def solveNQ(self):
        board = [[0 for _ in range(N)] for _ in range(N)]
        board[self.x][self.y] = 1
        self.ld[self.x-self.y+N-
1]=self.rd[self.x+self.y]=self.cl[self.x]=1
        if not self.solveNQUtil(board, 0):
            print("Solution does not exist")
            return False
        self.printSolution(board)
        return True
if __name__ == "__main__":
    N = 8
    x, y = 3, 2

    NQBt = NQBacktracking(x, y)
    NQBt.solveNQ()
```

Output:

```
N Queen Backtracking Solution:
Given initial position of 1st queen at row: 3 column: 2

1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
```

Conclusion:

Hence, we learn backtracking, and create a N Queen Problem solver. For example we tested 8 queens chess board.

.

Experiment 7.

Mini Project

Aim:

Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.

Theory:

- **Merge Sort Algorithm**

  The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

- **Merge Sort Working Process:**

  Think of it as a recursive algorithm continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

- **Merge Sort using Multi-threading**

  Merge Sort is a popular sorting technique which divides an array or list into two halves and then start merging them when sufficient depth is reached. Time complexity of merge sort is O(n log n).

  Threads are lightweight processes and threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

  Multi-threading is way to improve parallelism by running the threads simultaneously in different cores of your processor. In this program, we'll use 4 threads but you may change it according to the number of cores your processor has.

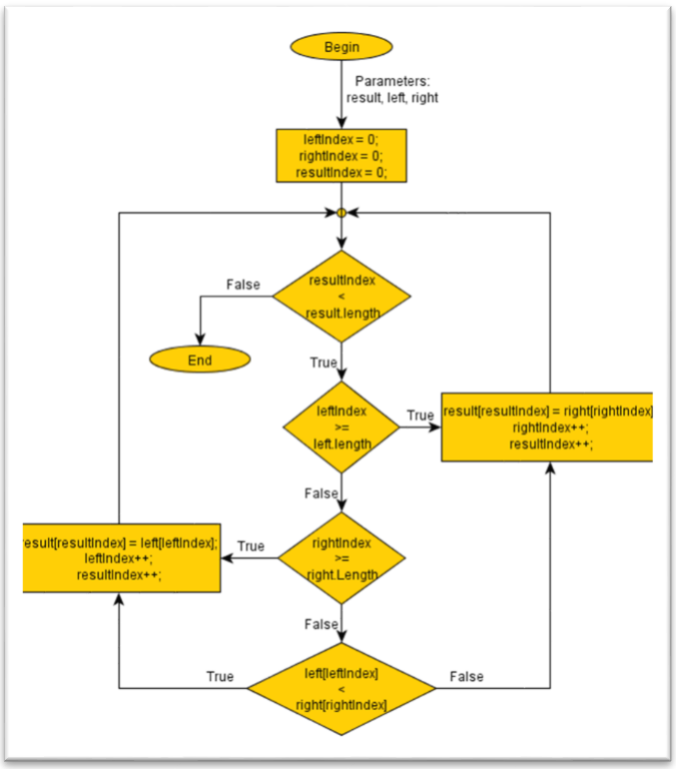
Algorithm:

- **Step 1:** `Start`
- **Step 2:** `declare array and left, right, mid variable`
- **Step 3:** `perform merge function.`

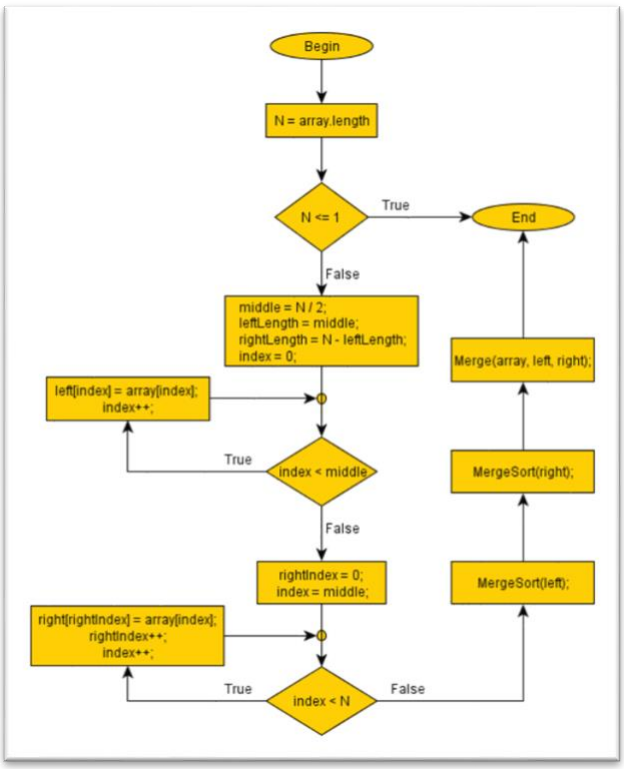
```
        if left > right
            return
        mid= (left+right)/2
    mergesort(array, left, mid)
    mergesort(array, mid+1, right)
    merge(array, left, mid, right)
```

- **Step 4:** `Stop`

Flow Chart:



*** Merge routine ***



*** Merge sort ***

Program:

```
import random
import sys
from contextlib import contextmanager
from multiprocessing import Manager, Pool
from timeit import default_timer as time


class Timer:
    """
    Record timing information.
    """

    def __init__(self, *steps):
        self._time_per_step = dict.fromkeys(steps)

    def __getitem__(self, item):
        return self.time_per_step[item]

    @property
    def time_per_step(self):
        return {
            step: elapsed_time
            for step, elapsed_time in self._time_per_step.items()
            if elapsed_time is not None and elapsed_time > 0
        }

    def start_for(self, step):
        self._time_per_step[step] = -time()

    def stop_for(self, step):
        self._time_per_step[step] += time()


def merge_sort_multiple(results, array):
    """Async parallel merge sort."""
    results.append(merge_sort(array))


def multMerge(results, array_part_left, array_part_right):
    """Merge two sorted lists in parallel."""
    results.append(merge(array_part_left, array_part_right))
```

```
def merge_sort(array):
    """Perform merge sort."""
    array_length = len(array)

    if array_length <= 1:
        return array

    middle_index = array_length // 2
    left = array[:middle_index]
    right = array[middle_index:]
    left = merge_sort(left)
    right = merge_sort(right)
    return merge(left, right)


def merge(left, right):
    """Merge two sorted lists."""
    sorted_list = [0] * (len(left) + len(right))
    i = j = k = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_list[k] = left[i]
            i += 1
        else:
            sorted_list[k] = right[j]
            j += 1
        k += 1

    while i < len(left):
        sorted_list[k] = left[i]
        i += 1
        k += 1

    while j < len(right):
        sorted_list[k] = right[j]
        j += 1
        k += 1

    return sorted_list
```


Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

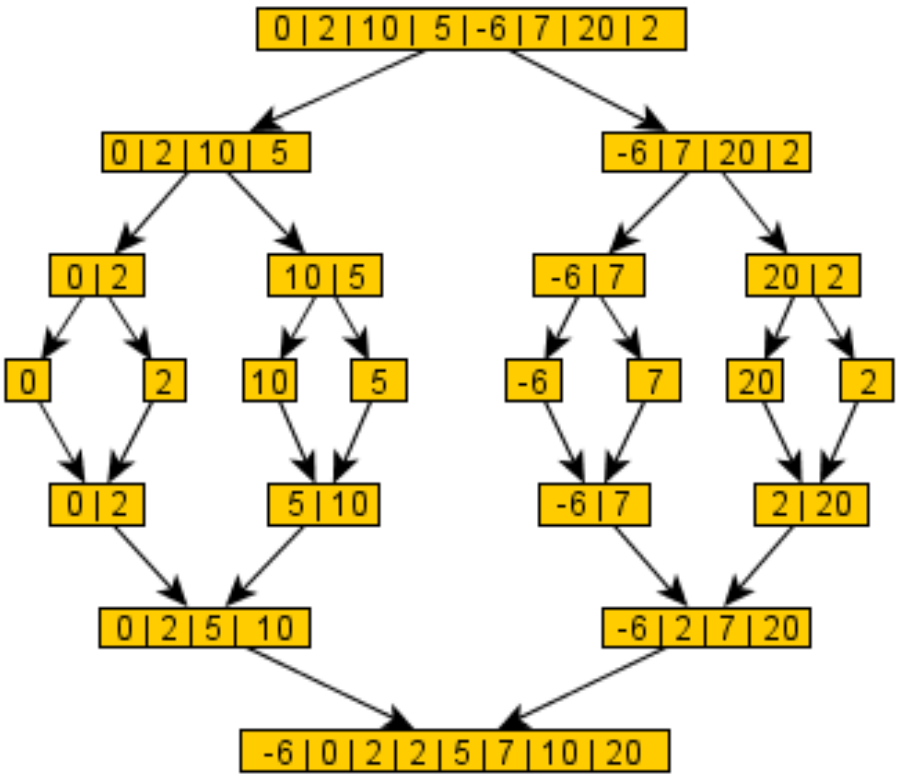
```
@contextmanager
def process_pool(size):
    """Create a process pool and block until all processes have
completed."""
    pool = Pool(size)
    yield pool
    pool.close()
    pool.join()

def para_merge_sort(array, ps_count):
    """Perform parallel merge sort."""
    timer = Timer("sort", "merge", "total")
    timer.start_for("total")
    timer.start_for("sort")

    step = int(length / ps_count) # Divide the list in chunks

    # Instantiate a multiprocessing.Manager obj to store the output
    manager = Manager()
    res = manager.list()

    with process_pool(size=ps_count) as pool:
        for i in range(ps_count):
            # We create a new Process object and assign the
            # merge_sort_multiple function to it,
            # using as input a sublist
            if i < ps_count - 1:
                chunk = array[i * step : (i + 1) * step]
            else:
                # Get the remaining elements in the list
                chunk = array[i * step :]
            pool.apply_async(merge_sort_multiple, (res, chunk))

    timer.stop_for("sort")
    print("Performing final merge.")
    timer.start_for("merge")

    # For a core count greater than 2, we can use multiprocessing
    # again to merge sub-lists in parallel.
    while len(res) > 1:
        with process_pool(size=ps_count) as pool:
            pool.apply_async(multMerge, (res, res.pop(0), res.pop(0)))

    timer.stop_for("merge")
    timer.stop_for("total")
    final_sorted_list = res[0]
    return timer, final_sorted_list
```

```
def get_command_line_parameters():
    """Get the process count from command line parameters."""
    if len(sys.argv) > 1:
        total_processes = int(sys.argv[1])
        if total_processes > 1:
            # Restrict process count to even numbers
            if total_processes % 2 != 0:
                print("Process count should be an even number.")
                sys.exit(1)
        print(f"Using {total_processes} cores")
    else:
        total_processes = 1
    return {"pc": total_processes}

if __name__ == "__main__":
    parameters = get_command_line_parameters()

    pc = parameters["pc"]
    main_timer = Timer("single_core", "list_generation")
    main_timer.start_for("list_generation")

    length = random.randint(3 * 10**6, 4 * 10**6)

    randArr=[random.randint(0,i*100) for i in range(length)]
    main_timer.stop_for("list_generation")

    print(f"List length: {length}")
    print(f"Random generated in {main_timer['list_generation']:.6f}")

    main_timer.start_for("single_core")
    single = merge_sort(randArr)
    main_timer.stop_for("single_core")

    randArr_sorted = randArr[:] # Create a copy first due to mutation
    randArr_sorted.sort()

    print("Verification of sorting algo:", randArr_sorted == single)
    print(f"Single Core elapsed time: {main_timer['single_core']:.6f} sec")
    print("Starting parallel sort.")
    para_timer, para_sorted_list = para_merge_sort(
        randArr, pc
    )

    print(f"Final merge duration: {para_timer['merge']:.6f} sec")
    print("Sorted arrays equal:", para_sorted_list == randArr_sorted)
    print(f"{pc}-Core elapsed time: {para_timer['total']:.6f} sec")
```

Output:

```
Using 16 cores
List length: 3252321
Random list generated in 1.787607
Verification of sorting algorithm: True
Single Core elapsed time: 14.204555 sec
Starting parallel sort.
Performing final merge.
Final merge duration: 4.764258 sec
Sorted arrays equal: True
16-Core elapsed time: 6.984671 sec
```

Idea Chart:



Conclusion:

Hense, we learned in this project are Sort, Merge Sort, Multi thread, Multi treated merge sort.

We also calculated time required.

.

Group B

Machine Learning

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Experiment 1.

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

Aim:

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.**

Theory:

- **Dataset :**

  A dataset in machine learning is, quite simply, a collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes.

- **Pre-Process:**

  Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model.

- **Outliers:**

  Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called Outliers.

- **Correlation:**

  Correlation explains how one or more variables are related to each other.

- **Linear Regression:**

  Linear regression uses the relationship between the data-points to draw a straight line through all them.

- **Random Forest Regression:**

  Random forest is a supervised learning algorithm that uses an ensemble learning method for classification and regression. Random forest is a bagging technique and not a boosting technique. The trees in random forests run in parallel, meaning is no interaction between these trees while building the trees.

Algorithm:

- **Step 1:** Importing required libraries
- **Step 2:** Read the Dataset
- **Step 3:** Cleaning
- **Step 4:** Finding the Outliers
- **Step 5:** Standardization
- **Step 6:** Splitting the Dataset
- **Step 7:** Random Forest Regression
- **Step 8:** Visualisation

Database: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing

df = pd.read_csv('uber.csv')

df.info()
df.head()
df.describe()

df = df.drop(['Unnamed: 0', 'key'], axis=1)

df.isna().sum()

df.dropna(axis=0,inplace=True)

df.dtypes

df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')

df= df.assign(
    second = df.pickup_datetime.dt.second,
    minute = df.pickup_datetime.dt.minute,
    hour = df.pickup_datetime.dt.hour,
    day= df.pickup_datetime.dt.day,
    month = df.pickup_datetime.dt.month,
    year = df.pickup_datetime.dt.year,
```

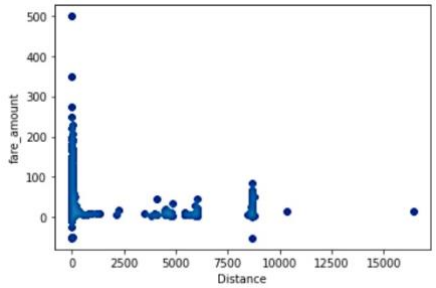
```
    dayofweek = df.pickup_datetime.dt.dayofweek
)
df = df.drop('pickup_datetime',axis=1)

incorrect_coordinates = df.loc[
    (df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
    (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
    (df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
    (df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')

def distance_transform(longitude1, latitude1, longitude2, latitude2):
    long1, lati1, long2, lati2 = map(np.radians, [longitude1, latitude1,
longitude2, latitude2])
    dist_long = long2 - long1
    dist_lati = lati2 - lati1
    a = np.sin(dist_lati/2)**2 + np.cos(lati1) * np.cos(lati2) *
np.sin(dist_long/2)**2
    c = 2 * np.arcsin(np.sqrt(a)) * 6371
return c

df['Distance'] = distance_transform(
    df['pickup_longitude'],
    df['pickup_latitude'],
    df['dropoff_longitude'],
    df['dropoff_latitude']
)

plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")

plt.figure(figsize=(20,12))
sns.boxplot(data = df)

df.drop(df[df['Distance'] >= 60].index, inplace = True)
df.drop(df[df['fare_amount'] <= 0].index, inplace = True)

df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace = True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index, inplace=True )

plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")

corr = df.corr()

corr.style.background_gradient(cmap='BuGn')

X = df['Distance'].values.reshape(-1, 1)        #Independent Variable
y = df['fare_amount'].values.reshape(-1, 1)     #Dependent Variable

from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(y)
print(y_std)

x_std = std.fit_transform(X)
```

```
print(x_std)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2,
random_state=0)

from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))

y_pred = l_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result[['Predicted']] = y_pred

result.sample(10)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test,
y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
y_pred)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred)))

plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")

plt.tight_layout()
plt.show()

cols = ['Model', 'RMSE', 'R-Squared']

# create a empty dataframe of the columns: specifies the columns to be selected
result_tabulation = pd.DataFrame(columns = cols)

# compile the required information
linreg_metrics = pd.DataFrame([[
     "Linear Regresion model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
     np.sqrt(metrics.r2_score(y_test, y_pred))
]], columns = cols)

result_tabulation = pd.concat([result_tabulation, linreg_metrics],
ignore_index=True)
```

```
result_tabulation


rf_reg = RandomForestRegressor(n_estimators=100, random_state=10)

# fit the regressor with training dataset
rf_reg.fit(X_train, y_train)

# predict the values on test dataset using predict()
y_pred_RF = rf_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result['Predicted'] = y_pred_RF

result.sample(10)


print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_RF))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test,
y_pred_RF))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_RF))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
y_pred_RF)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred_RF)))


# Build scatterplot
plt.scatter(X_test, y_test, c = 'b', alpha = 0.5, marker = '.', label = 'Real')
plt.scatter(X_test, y_pred_RF, c = 'r', alpha = 0.5, marker = '.', label =
'Predicted')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.grid(color = '#D3D3D3', linestyle = 'solid')
plt.legend(loc = 'lower right')


plt.tight_layout()
plt.show()


# compile the required information
random_forest_metrics = pd.DataFrame([[
     "Random Forest Regressor model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)),
     np.sqrt(metrics.r2_score(y_test, y_pred_RF))
]], columns = cols)


result_tabulation = pd.concat([result_tabulation, random_forest_metrics],
ignore_index=True)


result_tabulation
```

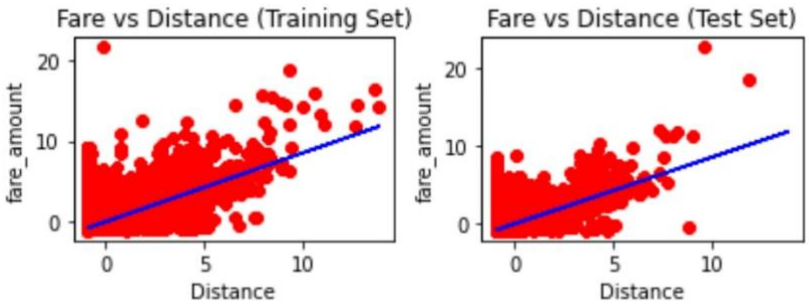
Output:

Outliers:



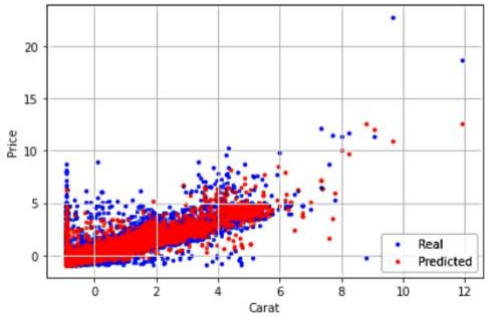
Linear Regression:

Mean Absolute Error: 0.26621298757938955
Mean Absolute % Error: 1.983074763340738
Mean Squared Error: 0.2705243510778542
Root Mean Squared Error: 0.5201195546005305
R Squared (R^2): 0.8567653080822022



Random Forests Regression:

Mean Absolute Error: 0.3077087698385678
Mean Absolute % Error: 2.161623761570947
Mean Squared Error: 0.33297733033643484
Root Mean Squared Error: 0.5770418791876677
R Squared (R^2): 0.8201518783882692



Conclusion:

Hence, we here we studied that what is the dataset, correlation, outliers, linear regression and random forest regression with programmatic visualisation.

.

Experiment 2.

Classify the email using the binary classification method.

Aim:

Analyze their performance.

a) **Normal State – Not Spam**

b) **Abnormal State – Spam.**

Use K-Nearest Neighbors and Support Vector Machine for classification.

Theory:

- **Binary Classification:**
  In machine learning, binary classification is a supervised learning algorithm that categorizes new observations into one of two classes.
- **K-Nearest Neighbors:**
  The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.
- **Support Vector Machine:**
  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.

Algorithm:

- **Step 1:** Importing required libraries
- **Step 2:** Read the Dataset
- **Step 3:** Cleaning
- **Step 4:** Separating the features and the labels
- **Step 5:** Splitting the Dataset
- **Step 6: Machine Learning models**
  - The following 5 models are used:
    - K-Nearest Neighbors
    - Linear SVM
    - Polynomial SVM
    - RBF SVM
    - Sigmoid SVM
- **Step 7:** Fit and predict on each model

Database: https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

Program:

```
import pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics, preprocessing

df = pd.read_csv('emails.csv')

df.info()
df.head()

df.drop(columns=['Email No.'], inplace=True)
df.isna().sum()
df.describe()

X=df.iloc[:, :df.shape[1]-1]        #Independent Variables
y=df.iloc[:, -1]                    #Dependent Variable
X.shape, y.shape

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
random_state=8)

models = {
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=2),
    "Linear SVM":LinearSVC(random_state=8, max_iter=900000),
    "Polynomical SVM":SVC(kernel="poly", degree=2, random_state=8),
    "RBF SVM":SVC(kernel="rbf", random_state=8),
    "Sigmoid SVM":SVC(kernel="sigmoid", random_state=8)
}

for name, model in models.items():
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for {name} model \t: {metrics.accuracy_score(y_test,
y_pred)}")
```

Output:

Accuracy for K-Nearest Neighbors model : 0.8878865979381443
Accuracy for Linear SVM model : 0.9755154639175257
Accuracy for Polynomical SVM model : 0.7615979381443299
Accuracy for RBF SVM model : 0.8182989690721649
Accuracy for Sigmoid SVM model : 0.6237113402061856

Conclusion:

Hence, we studied here, Classification of email using K-Nearest Neighbors, Linear SVM, Polynomial SVM, RBF SVM, Sigmoid SVM, for Analyze their performance..

Experiment 3.

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Aim:

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).**

Theory:

- **Feature:**

  Features are nothing but the independent variables in machine learning models. What is required to be learned in any specific machine learning problem is a set of these features (independent variables), coefficients of these features, and parameters for coming up with appropriate functions or models (also termed hyper parameters).

- **Training Set:**

  Training data is also known as training dataset, learning set, and training set. It's an essential component of every machine learning model and helps them make accurate predictions or perform a desired task.

- **Test Set:**

  The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

- **Normalisation:**

  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.

- **Model:**

  A "model" in machine learning is the output of a machine learning algorithm run on data.

  A model represents what was learned by a machine learning algorithm.

Algorithm:

- **Step 1:** Importing required libraries
- **Step 2:** Read the Dataset
- **Step 3:** Cleaning
- **Step 4:** Separating the features and the labels
- **Step 5:** Splitting the Dataset
- **Step 6:** Initialize & build the model
- **Step 7:** Predict the results using 0.5 as a threshold
- **Step 8:** Print the Accuracy score and confusion matrix

Database: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

Program:

```
import pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics, preprocessing

df = pd.read_csv('churn_modelling.csv')
df.info()
df.head()

df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], inplace=True)
df.isna().sum()
df.describe()

X=df.iloc[:, :df.shape[1]-1].values       #Independent Variables
y=df.iloc[:, -1].values                   #Dependent Variable
X.shape, y.shape

print(X[:8,1], '... will now become: ')

label_X_country_encoder = LabelEncoder()
X[:,1] = label_X_country_encoder.fit_transform(X[:,1])
print(X[:8,1])

print(X[:6,2], '... will now become: ')

label_X_gender_encoder = LabelEncoder()
X[:,2] = label_X_gender_encoder.fit_transform(X[:,2])
print(X[:6,2])
```

```
transform = ColumnTransformer([("countries", OneHotEncoder(), [1])],
remainder="passthrough") # 1 is the country column
X = transform.fit_transform(X)

X = X[:,1:]
X.shape

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

sc=StandardScaler()
X_train[:,np.array([2,4,5,6,7,10])] =
sc.fit_transform(X_train[:,np.array([2,4,5,6,7,10])])
X_test[:,np.array([2,4,5,6,7,10])] =
sc.transform(X_test[:,np.array([2,4,5,6,7,10])])

sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train

from tensorflow.keras.models import Sequential

# Initializing the ANN
classifier = Sequential()

from tensorflow.keras.layers import Dense

# The amount of nodes (dimensions) in hidden layer should be the average of input
and output layers, in this case 6.
# This adds the input layer (by specifying input dimension) AND the first hidden
layer (units)
classifier.add(Dense(activation = 'relu', input_dim = 11, units=256,
kernel_initializer='uniform'))

# Adding the hidden layer
classifier.add(Dense(activation = 'relu', units=512,
kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=256,
kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=128,
kernel_initializer='uniform'))

# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output
(stay with the bank or not)
# We use the sigmoid because we want probability outcomes
classifier.add(Dense(activation = 'sigmoid', units=1,
kernel_initializer='uniform'))

# Create optimizer with default learning rate
# sgd_optimizer = tf.keras.optimizers.SGD()
# Compile the model
```

```
classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

classifier.summary()

classifier.fit(
    X_train, y_train,
    validation_data=(X_test,y_test),
    epochs=20,
    batch_size=32
)

y_pred = classifier.predict(X_test)

# To use the confusion Matrix, we need to convert the probabilities that a
customer will leave the bank into the form true or false.
# So we will use the cutoff value 0.5 to indicate whether they are likely to exit
or not.
y_pred = (y_pred > 0.5)

from sklearn.metrics import confusion_matrix,classification_report

cm1 = confusion_matrix(y_test, y_pred)

print(classification_report(y_test, y_pred))

accuracy_model1 =
((cm1[0][0]+cm1[1][1])*100)/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0])
print (accuracy_model1, '% of testing data was classified correctly')
```

Output:

```
                  precision    recall  f1-score   support

               0       0.88      0.96      0.92      1595
               1       0.74      0.49      0.59       405

        accuracy                           0.86      2000
       macro avg       0.81      0.72      0.75      2000
    weighted avg       0.85      0.86      0.85      2000


86.15 % of testing data was classified correctly
```

Conclusion:

Hence, we studied that, how is the training & test data sets are. What is model and how to built it. Classified testing data set.

.

Experiment 4.

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

Aim:

Find the local minima of the function $y=(x+3)^2$ starting from the point x=2.

Theory:

Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model.

Types of gradient Descent:

- **Batch Gradient Descent:**

  This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.

- **Stochastic Gradient Descent:**

  This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

- **Mini Batch gradient descent:**

  This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b<m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

Algorithm:

- **Gradient Descent:**

Algorithm 1: Pseudo-code for GD

Function `GD`**:**
- Set epsilon as the limit of convergence
- **for** $j = 1$ *and* $j = 0$ **do**
  - **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
    - $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
  - **end**
- **end**

- **Stochastic Gradient Descent**

Algorithm 2: Pseudo-code for SGD

Function `SGD`**:**
- Set epsilon as the limit of convergence
- **for** $i = 1, \ldots, m$ **do**
  - **for** $j = 0, \ldots, n$ **do**
    - **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
      - $\omega_{j+1} := \omega_j - \alpha \cdot (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
    - **end**
  - **end**
- **end**

- **Mini Batch Gradient Descent**

Algorithm 3: Pseudo-code for Mini Batch Gradient Desecent

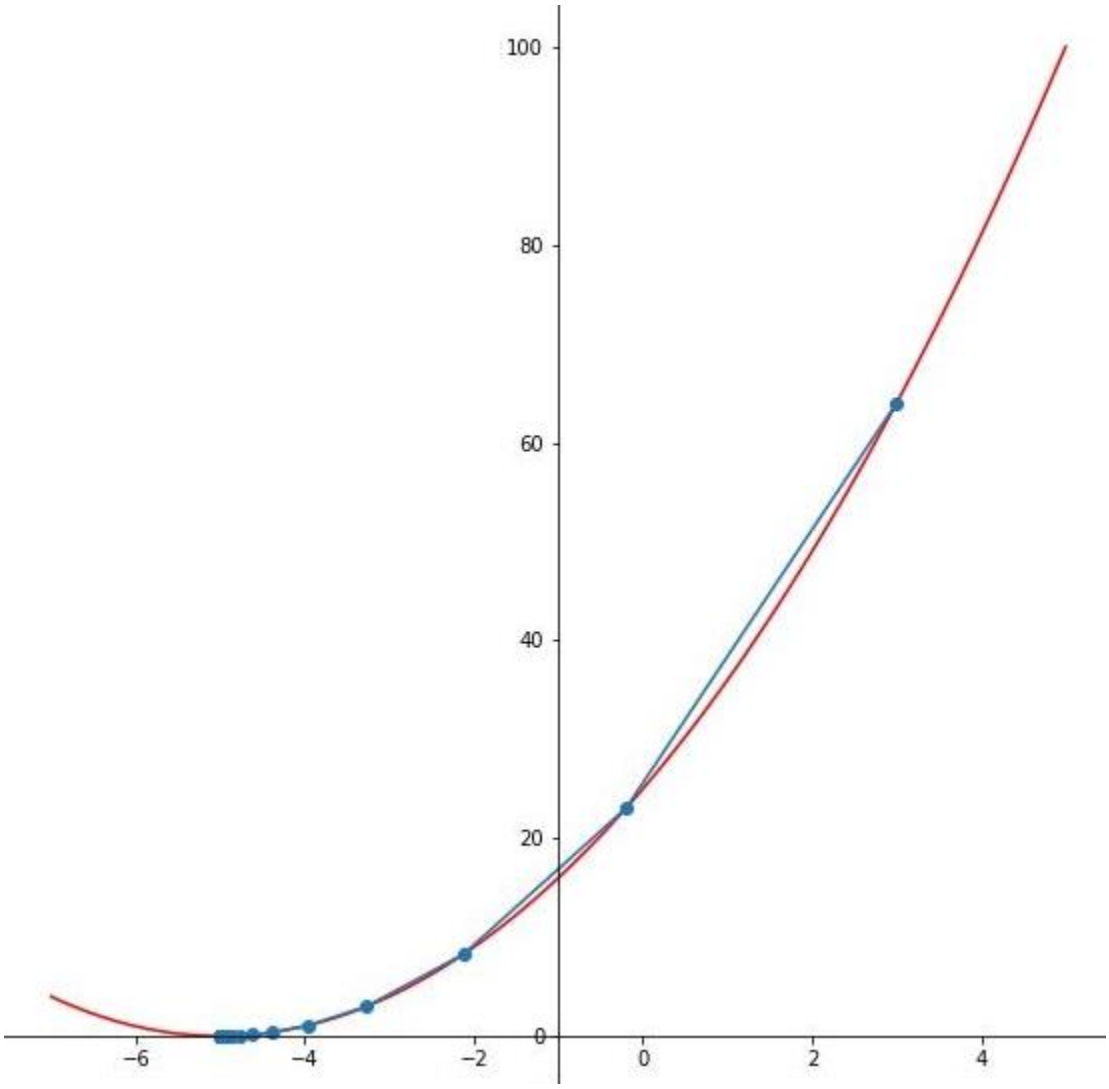
Function `SGD`**:**
- Set epsilon as the limit of convergence
- **for** $i = 1, \ldots, b$ **do**
  - **for** $j = 0, \ldots, n$ **do**
    - **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
      - $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{b} \sum_{k=i}^{i+b-1} (h_\omega(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$;
    - **end**
  - **end**
- **end**

Program:

```
from sympy import Symbol, lambdify
import matplotlib.pyplot as plt
import numpy as np

x = Symbol('x')

def gradient_descent(
    function, start, learn_rate, n_iter=10000, tolerance=1e-06, step_size=1 ):
    gradient = lambdify(x, function.diff(x))
    function = lambdify(x, function)
    points = [start]
    iters = 0                              #iteration counter

    while step_size > tolerance and iters < n_iter:
        prev_x = start                     #Store current x value in prev_x
        start = start - learn_rate * gradient(prev_x) #Grad descent
        step_size = abs(start - prev_x) #Change in x
        iters = iters+1                    #iteration count
        points.append(start)
    print("The local minimum occurs at", start)

    # Create plotting array
    x_ = np.linspace(-7,5,100)
    y = function(x_)

    # setting the axes at the centre
    fig = plt.figure(figsize = (10, 10))
    ax = fig.add_subplot(1, 1, 1)
    ax.spines['left'].set_position('center')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.yaxis.set_ticks_position('left')

    # plot the function
    plt.plot(x_,y, 'r')
    plt.plot(points, function(np.array(points)), '-o')

    # show the plot
    plt.show()

function=(x+5)**2

gradient_descent(
     function=function, start=3.0, learn_rate=0.2, n_iter=50
)
```

Output:

The local minimum occurs at -4.999998938845185



Conclusion:

Hence, we learn Gradient Descent Algorithm and it's types. And program for calculating local minima.

.

Experiment 5.

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.

Aim:

Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

Theory:

- **Nearest Neighbors 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 suite category by using K- NN algorithm.

  - K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

  - 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.

Algorithm:

- **Step 1:** Importing libraries
- **Step 2:** Loading the Dataset
- **Step 3:** Cleaning
- **Step 4:** Visualization
- **Step 5:** Separating the features and the labels
- **Step 6:** Splitting the Dataset
- **Step 7:** Machine Learning model
- **Step 8:** Hyper parameter tuning

Database: https://www.kaggle.com/datasets/abdallamahgoub/diabetes

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from sklearn import preprocessing

df = pd.read_csv('diabetes.csv')
df.info()
df.head()

df.corr().style.background_gradient(cmap='BuGn')
df.isna().sum()
df.describe()

hist = df.hist(figsize=(20,16))

X=df.iloc[:, :df.shape[1]-1]        #Independent Variables
y=df.iloc[:, -1]                    #Dependent Variable
X.shape, y.shape

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=8)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
def knn(X_train, X_test, y_train, y_test, neighbors, power):
    model = KNeighborsClassifier(n_neighbors=neighbors, p=power)
    # Fit and predict on model
    # Model is trained using the train set and predictions are made based on the
test set. Accuracy scores are calculated for the model.
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for K-Nearest Neighbors model \t: {accuracy_score(y_test,
y_pred)}")

    cm = confusion_matrix(y_test, y_pred)
    print(f'''Confusion matrix :\n
| Positive Prediction\t| Negative Prediction
---------------+------------------------+----------------------
Positive Class | True Positive (TP) {cm[0, 0]}\t| False Negative (FN) {cm[0, 1]}
---------------+------------------------+----------------------
Negative Class|False Positive(FP){cm[1,0]}\t| True Negative (TN) {cm[1,1]}\n''')
    cr = classification_report(y_test, y_pred)
    print('Classification report : \n', cr)

param_grid = {
    'n_neighbors': range(1, 51),
    'p': range(1, 4)
}
grid=GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
grid.best_estimator_, grid.best_params_, grid.best_score_

knn(X_train, X_test, y_train, y_test, grid.best_params_['n_neighbors'],
grid.best_params_['p'])
```

Output:

```
Accuracy for K-Nearest Neighbors model  : 0.7987012987012987
Confusion matrix :
    | Positive Prediction       | Negative Prediction
    ---------------+------------------------+----------------------
    Positive Class | True Positive (TP) 91     | False Negative (FN) 11
    ---------------+------------------------+----------------------
    Negative Class | False Positive (FP) 20    | True Negative (TN) 32
Classification report :
               precision    recall  f1-score   support
           0       0.82      0.89      0.85       102
           1       0.74      0.62      0.67        52
    accuracy                           0.80       154
   macro avg       0.78      0.75      0.76       154
weighted avg       0.79      0.80      0.79       154
```

Conclusion:

Hence, here we learned that, confusion matrix, accuracy, error rate, precision and recall, by performing it in program.

.

Experiment 6.

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset.

Aim:

Determine the number of clusters using the elbow method.

Theory:

- **K-Means Clustering:**

  The most common clustering covered in machine learning for beginners is K-Means. The first step is to create c new observations among our unlabelled data and locate them randomly, called centroids. The number of centroids represents the number of output classes. The first step of the iterative process for each centroid is to find the nearest point (in terms of Euclidean distance) and assign them to its category. Next, for each category, the average of all the points attributed to that class is computed. The output is the new centroid of the class. With every iteration, the observations can be redirected to another centroid. After several reiterations, the centroid's change in location is less critical as initial random centroids converge with real ones—the process ends when there is no change in centroids' position. Many methods can be employed for the task, but a common one is 'elbow method'. A low level of variation is needed within the clusters measured by the within-cluster sum of squares. The number of centroids and observations are inversely proportional. Thus, setting the highest possible number of centroids would be inconsistent.

- **Hierarchical Clustering:**

  Two techniques are used by this algorithm- Agglomerative and Divisive. In HC, the number of clusters K can be set precisely like in K-means, and n is the number of data points such that n>K. The agglomerative HC starts from n clusters and aggregates data until K clusters are obtained. The divisive starts from only one cluster and then splits depending on similarities until K clusters are obtained. The similarity here is the distance among points, which can be computed in many ways, and it is the crucial element of discrimination. It can be computed with different approaches:

  Min: Given two clusters C1 and C2 such that point a belongs to C1 and b to C2. The similarity between them is equal to the minimum of distance

  Max: The similarity between points a and b is equal to the maximum of distance

  Average: All the pairs of points are taken, and their similarities are computed. Then the average of similarities is the similarity between C1 and C2.

Algorithm:

- **Step 1:** Importing Libraries
- **Step 2:** Reading the dataset
- **Step 3:** Creating Datalist
- **Step 4:** Applying K-mean algorithm and findout elbow points
- **Step 5:** Creating Clusters
- **Step 6:** Plotting the elbow plot

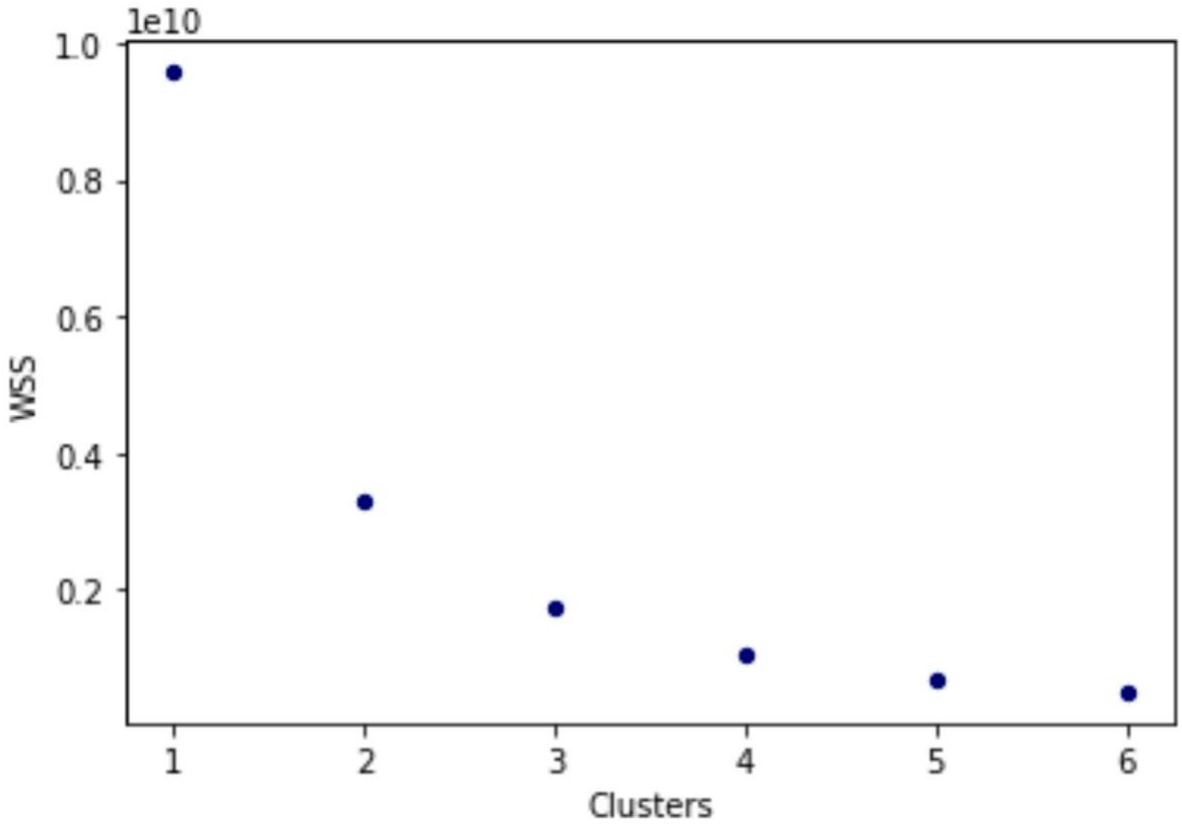
Database: https://www.kaggle.com/datasets/kyanyoga/sample

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.cluster as cluster

%matplotlib inline

df = pd.read_csv('sales_data_sample.csv', encoding='Latin-1')
df.head()

list = ["PRICEEACH", "SALES"]
data = df[list]
data.head()

K=range(1, 7)
wss = []
for k in K:
    kmeans=cluster.KMeans(n_clusters=k,init="k-means++")
    kmeans=kmeans.fit(data)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)

mycenters = pd.DataFrame({'Clusters' : K, 'WSS' : wss})
mycenters

sns.scatterplot(x = 'Clusters', y = 'WSS', data = mycenters)
```

Output:

Clusters		WSS
0	1	9.574691e+09
1	2	3.322310e+09
2	3	1.729502e+09
3	4	1.040298e+09
4	5	7.126630e+08
5	6	5.119084e+08



Conclusion:

Hence, we studied about K-Means clustering & hierarchical clustering. And also performed program on sales data.

.

Experiment 7. Mini Project.

Aim:

Build a machine Learning model that predicts the type of people who survived the Titanic shipwreck using Passenger data (i.e. name, age, gender, socio-economic class, etc.).

Theory:

- **Machine Learning:**

  Machine learning is a field of computer science that aims to teach computers how to learn and act without being explicitly programmed. More specifically, machine learning is an approach to data analysis that involves building and adapting models, which allow programs to "learn" through experience.

- **Model in Machine Learning:**

  A machine learning model is a program that can find patterns or make decisions from a previously unseen dataset. For example, in natural language processing, machine learning models can parse and correctly recognize the intent behind previously unheard sentences or combinations of words.

- **Prediction:**

  Prediction in machine learning refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome.

Algorithm:

- **Step 1:** Importing Required libraries
- **Step 2:** Reading Training & Test Datasets.
- **Step 3:** Cleaning
- **Step 4:** Splitting Datasets
- **Step 5:** Training the Model
- **Step 6:** Testing the Model
- **Step 7:** Use maodel for Prediction.

Database: https://www.kaggle.com/competitions/titanic/data

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train.info()
test.info()

train.head()
train.head()

train.corr().style.background_gradient(cmap='BuGn')

train.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
test.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

train.isna().sum()
test.isna().sum()

train['Embarked'] = train.Embarked.fillna(train.Embarked.dropna().max())
test['Fare'] = test.Fare.fillna(test.Fare.dropna().mean())

# we will guess the age from Pclass and Sex:

guess_ages = np.zeros((2,3))

combine = [train , test]

# Converting Sex categories (male and female) to 0 and 1:
for ds in combine:
    ds['Sex'] = ds['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

# Filling missed age feature:
for ds in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = ds[(ds['Sex'] == i) & \
                                  (ds['Pclass'] == j+1)]['Age'].dropna()
            age_guess = guess_df.median()

            # Convert random age float to nearest .5 age
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    for i in range(0, 2):
        for j in range(0, 3):
            ds.loc[ (ds.Age.isnull()) & (ds.Sex == i) & (ds.Pclass == j+1),\
                    'Age'] = guess_ages[i,j]

    ds['Age'] = ds['Age'].astype(int)
```

```
train.head()
train.describe()

X_train = pd.get_dummies(train.drop(['Survived'], axis=1))
X_test  = pd.get_dummies(test)
y_train = train['Survived']

def print_scores(model, X_train, Y_train, predictions, cv_splites=10):
    print("The mean accuracy score of the train data is %.5f" %
model.score(X_train, Y_train))
    CV_scores = cross_val_score(model, X_train, Y_train, cv=cv_splites)
    print("The individual cross-validation scores are: \n",CV_scores)
    print("The minimum cross-validation score is %.3f" % min(CV_scores))
    print("The maximum cross-validation score is %.3f" % max(CV_scores))
    print("The mean  cross-validation   score is %.5f Ã‚Â± %0.2f" %
(CV_scores.mean(), CV_scores.std() * 2))


model = RandomForestClassifier(n_estimators= 80 ,max_depth=5 , max_features=8
,min_samples_split=3 ,random_state=7)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print_scores(model, X_train, y_train, predictions)
```

Output:

Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	0	22	1	0	7.2500	S
1	1	1	38	1	0	71.2833	C
1	3	1	26	0	0	7.9250	S
1	1	1	35	1	0	53.1000	S
0	3	0	35	0	0	8.0500	S

Conclusion:

Hence, we learn that, Machine Learning, Machine Learning Model, Training the Model, Testing the model, And Actual Prediction.

.

Group C

Blockchain Technology

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Experiment 1.

Installation of Metamask and study spending Ether per transaction.

Aim:

Installation of Metamask and study spending Ether per transaction.

Theory:

- **Introduction to Blockchain**

  - Blockchain can be described as a data structure that holds transactional records and while ensuring security, transparency, and decentralization. You can also think of it as a chain or records stored in the forms of blocks which are controlled by no single authority.

  - A blockchain is a distributed ledger that is completely open to any and everyone on the network. Once an information is stored on a blockchain, it is extremely difficult to change or alter it.

  - Each transaction on a blockchain is secured with a digital signature that proves its authenticity. Due to the use of encryption and digital signatures, the data stored on the blockchain is tamper-proof and cannot be changed.

  - Blockchain technology allows all the network participants to reach an agreement, commonly known as consensus. All the data stored on a blockchain is recorded digitally and has a common history which is available for all the network participants. This way, the chances of any fraudulent activity or duplication of transactions is eliminated without the need of a third-party.

- **Blockchain Features**

  The following features make the revolutionary technology of blockchain stand out:

  - **Decentralized**

    Blockchains are decentralized in nature meaning that no single person or group holds the authority of the overall network. While everybody in the network has the copy of the distributed ledger with them, no one can modify it on his or her own. This unique feature of blockchain allows transparency and security while giving power to the users.

- **Peer-to-Peer Network**

  With the use of Blockchain, the interaction between two parties through a peer-to-peer model is easily accomplished without the requirement of any third party. Blockchain uses P2P protocol which allows all the network participants to hold an identical copy of transactions, enabling approval through a machine consensus. For example, if you wish to make any transaction from one part of the world to another, you can do that with blockchain all by yourself within a few seconds. Moreover, any interruptions or extra charges will not be deducted in the transfer.

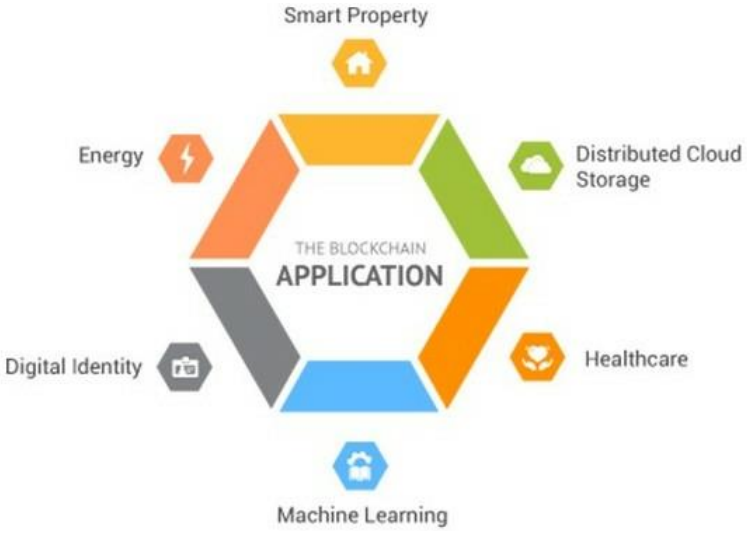
- **Immutable**

  The immutability property of a blockchain refers to the fact that any data once written on the blockchain cannot be changed. To understand immutability, consider sending email as an example. Once you send an email to a bunch of people, you cannot take it back. In order to find a way around, you'll have to ask all the recipients to delete your email which is pretty tedious. This is how immutability works.

- **Tamper-Proof**

  With the property of immutability embedded in blockchains, it becomes easier to detect tampering of any data. Blockchains are considered tamper-proof as any change in even one single block can be detected and addressed smoothly. There are two key ways of detecting tampering namely, hashes and blocks.

- **Popular Applications of Blockchain Technology**



- **Benefits of Blockchain Technology:**
  - **Time-saving:** No central Authority verification needed for settlements making the process faster and cheaper.
  - **Cost-saving:**
    A Blockchain network reduces expenses in several ways. No need for third-party verification. Participants can share assets directly. Intermediaries are reduced. Transaction efforts are minimized as every participant has a copy of shared ledger.
  - **Tighter security:**
    No one can temper with Blockchain Data as it is shared among millions of participants. The system is safe against cybercrimes and Fraud.
  - **In finance market trading, Fibonacci retracement levels are widely used in technical analysis.**

- **How to use MetaMask:**

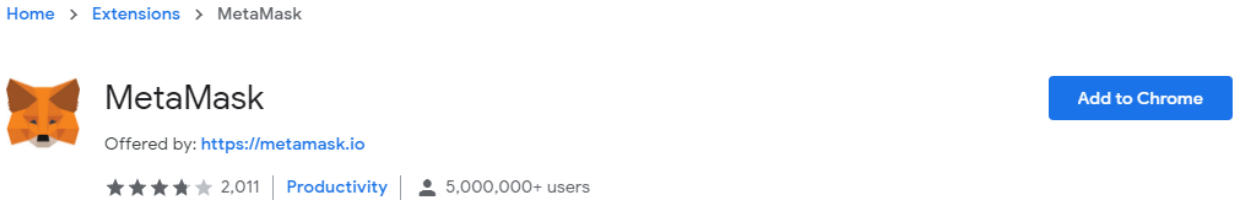
  A step by step guide

  MetaMask is one of the most popular browser extensions that serves as a way of storing your Ethereum and other ERC-20 Tokens. The extension is free and secure, allowing web applications to read and interact with Ethereum's blockchain.

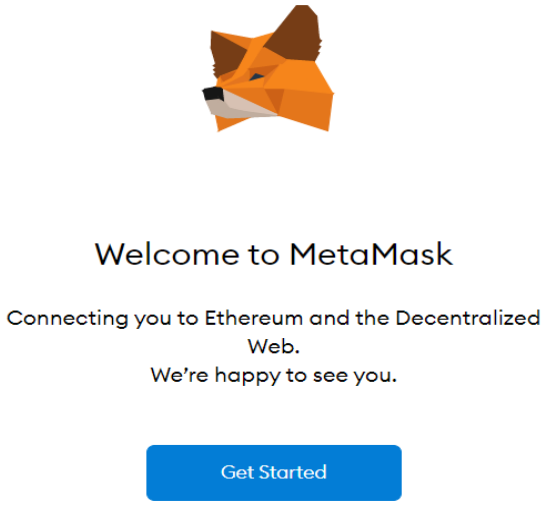
- **Advantages of MetaMask:**
  - Popular - It is commonly used, so users only need one plugin to access a wide range of apps.
  - Simple - Instead of managing private keys, users just need to remember a list of words, and transactions are signed on their behalf.
  - Saves space - Users don't have to download the Ethereum blockchain, as MetaMask sends requests to nodes outside of the user's computer.
  - Integrated - Dapps are designed to work with MetaMask, so it becomes much easier to send Ether in and out.

Istallation:

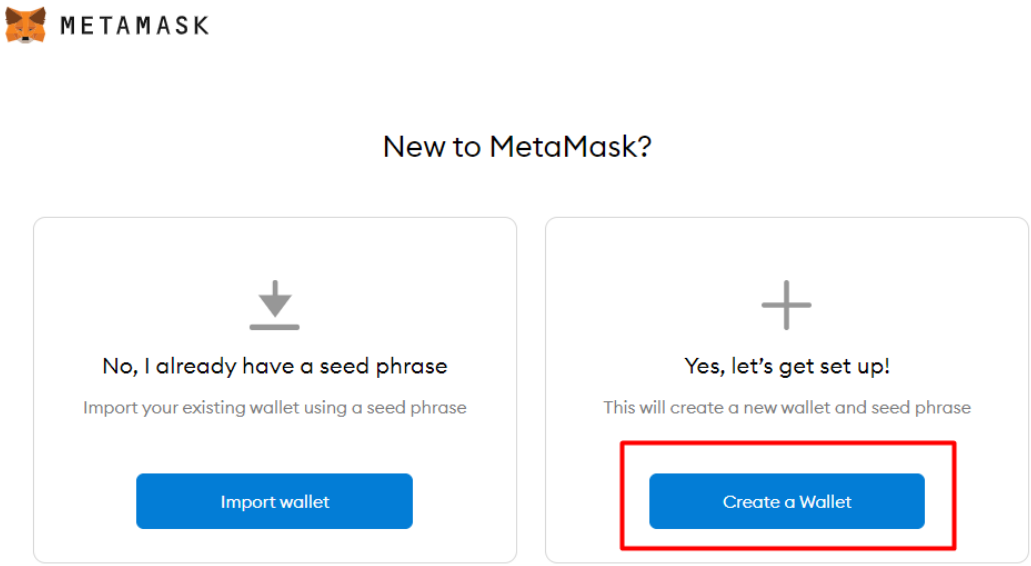
Step 1: Go to Chrome Web Store Extensions Section.
Step 2: Search *MetaMask.*
Step 3: Check the number of downloads to make sure that the legitimate MetaMask is being installed, as hackers might try to make clones of it.



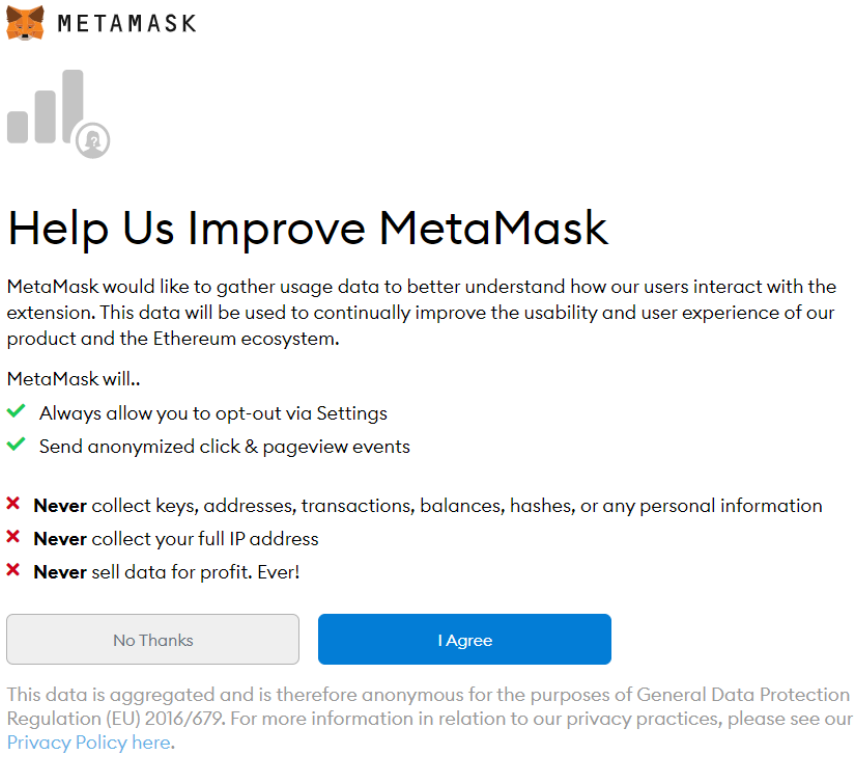
Step 4: Click the *Add to Chrome* button.
Step 5: Once installation is complete this page will be displayed. Click on the *Get Started* button.



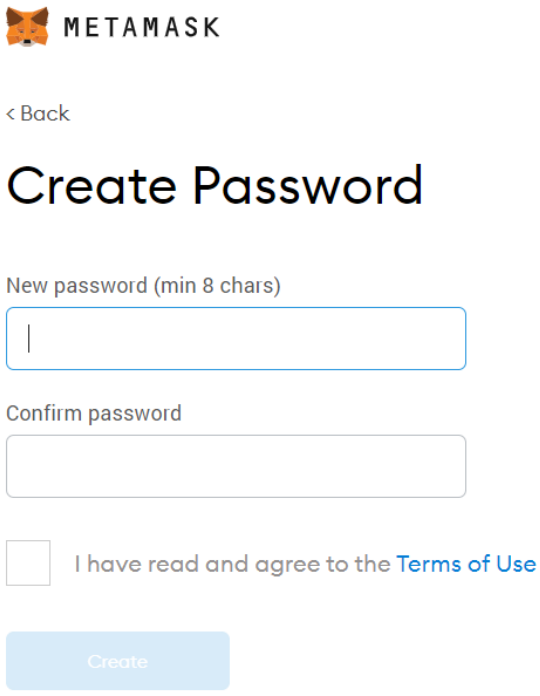
Step 6: This is the first time creating a wallet, so click the *Create a Wallet* button. If there is already a wallet then import the already created using the *Import Wallet* button.



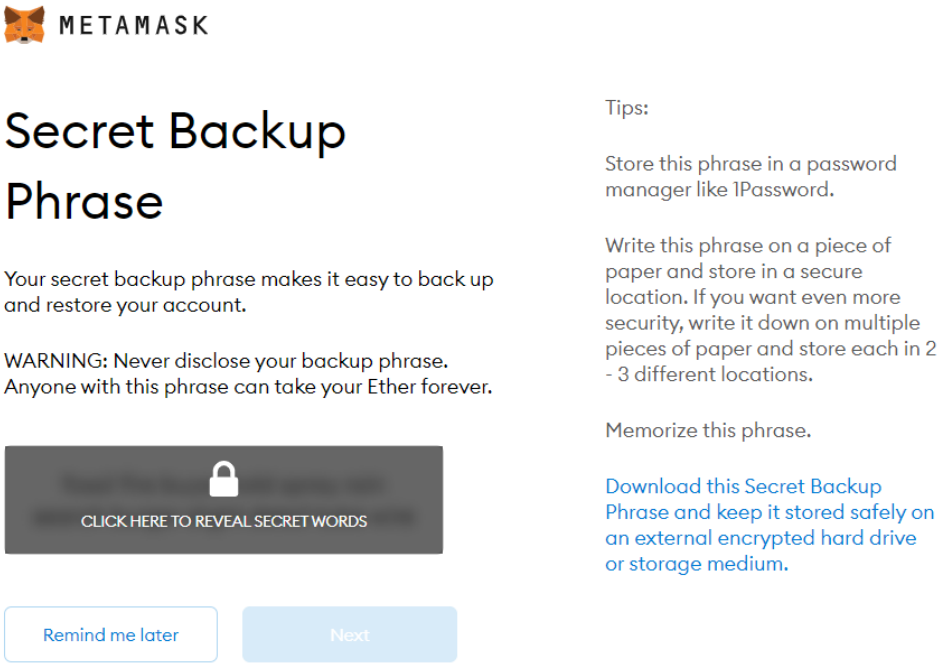
Step 7: Click *I Agree* button to allow data to be collected to help improve MetaMask or else click the *No Thanks* button. The wallet can still be created even if the user will click on the *No Thanks* button.



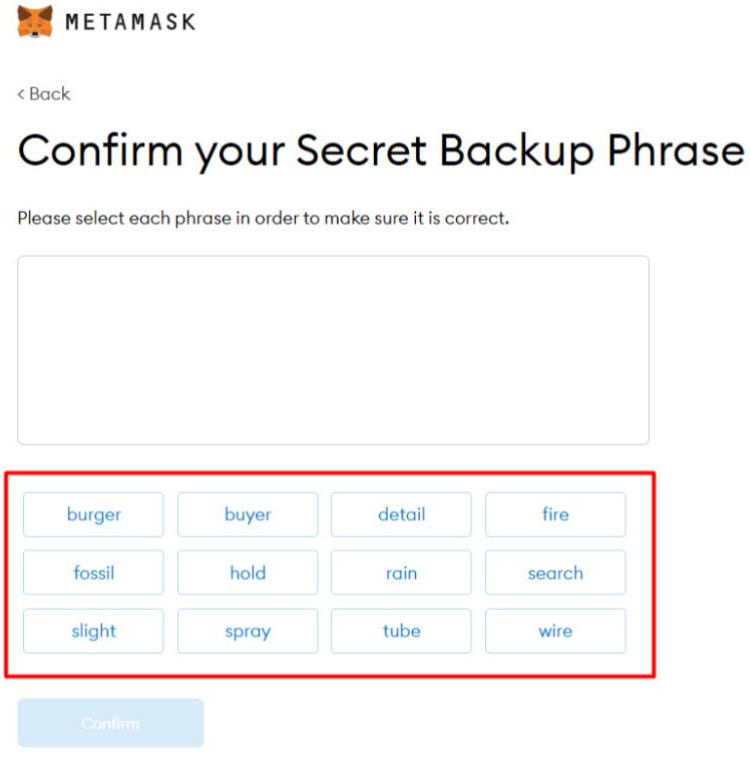
Step 8: Create a password for your wallet. This password is to be entered every time the browser is launched and wants to use MetaMask. A new password needs to be created if chrome is uninstalled or if there is a switching of browsers. In that case, go through the *Import Wallet* button. This is because MetaMask stores the keys in the browser. Agree to *Terms of Use*.



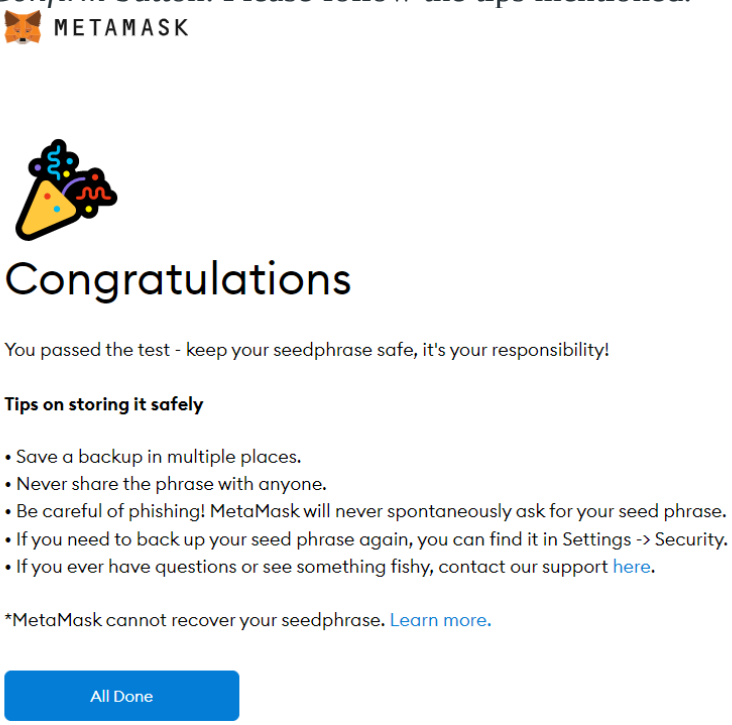
Step 9: Click on the dark area which says *Click here to reveal secret words* to get your secret phrase.
Step 10: This is the most important step. Back up your secret phrase properly. Do not store your secret phrase on your computer. Please read everything on this screen until you understand it completely before proceeding. The secret phrase is the only way to access your wallet if you forget your password. Once done click the *Next* button.



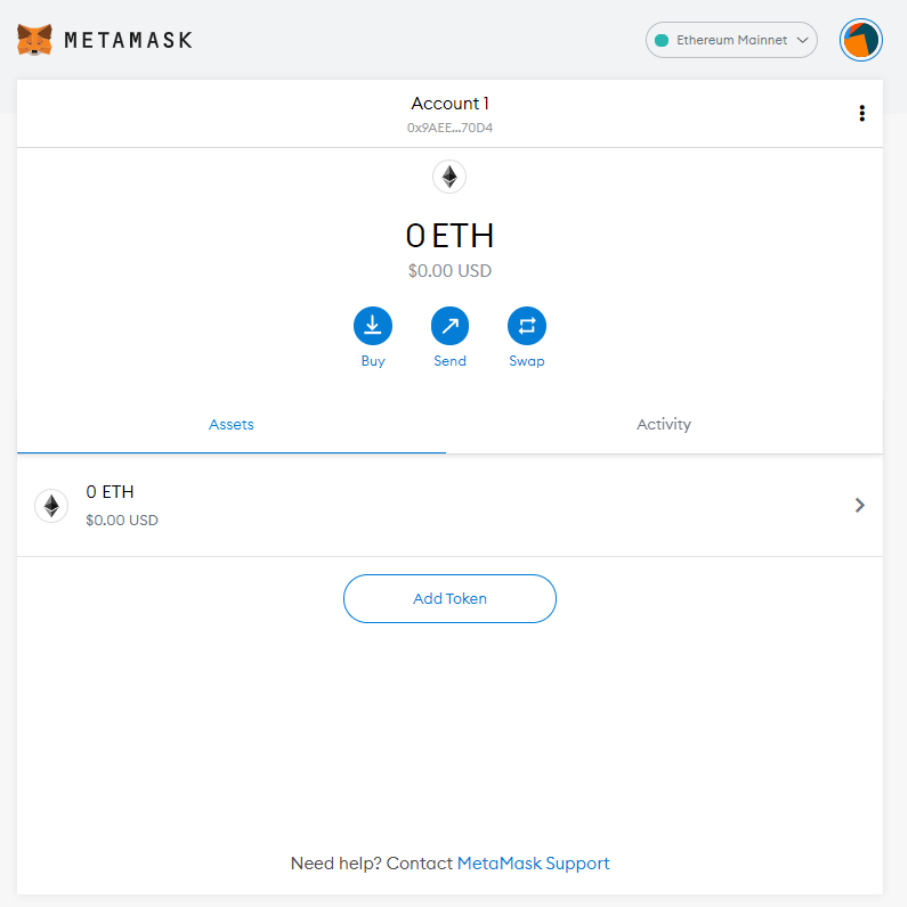
Step 11: Click the buttons respective to the order of the words in your seed phrase. In other words, type the seed phrase using the button on the screen. If done correctly the *Confirm* button should turn blue.



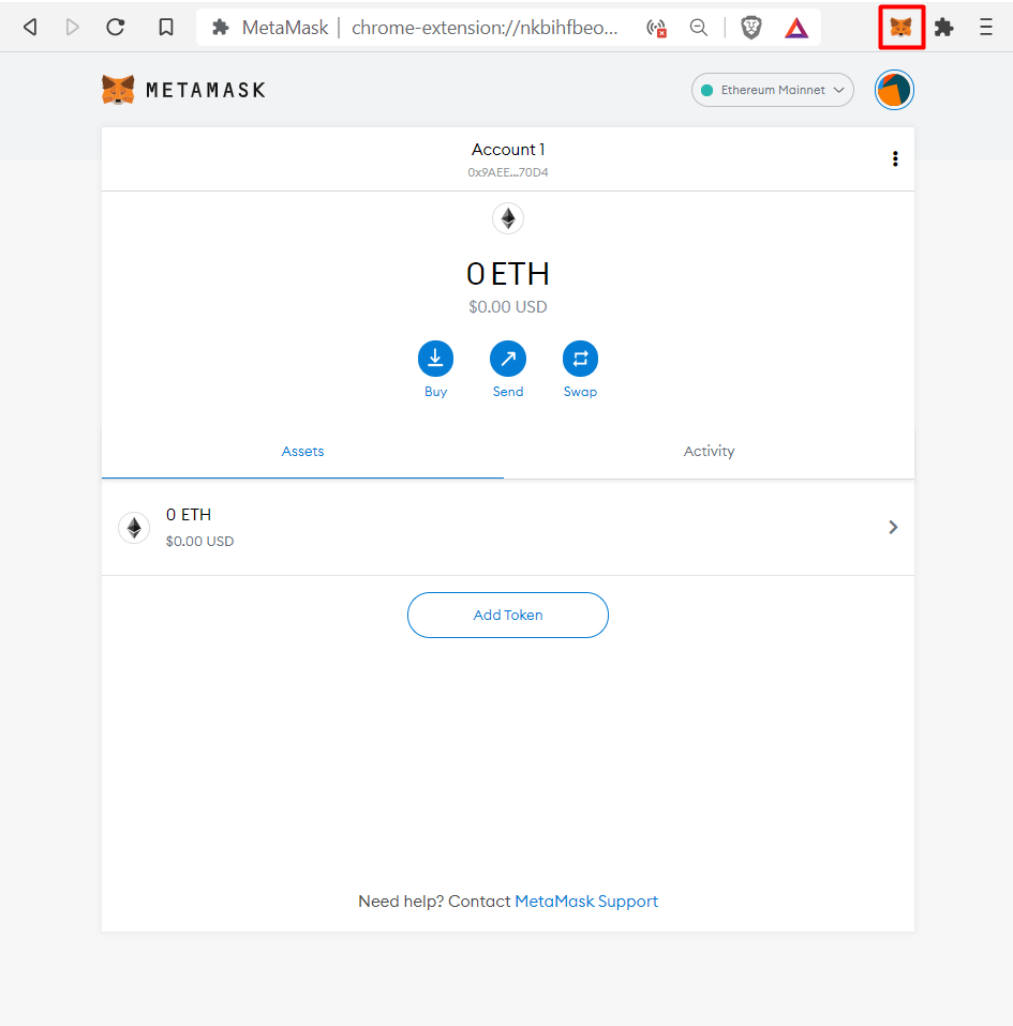
Step 12: Click the *Confirm* button. Please follow the tips mentioned.



Step 13: One can see the balance and copy the address of the account by clicking on the *Account 1* area.



Step 14: One can access MetaMask in the browser by clicking the Foxface icon on the top right. If the Foxface icon is not visible, then click on the puzzle piece icon right next to it.



Conclusion:

In this way we have explored Concept Blockchain and metamat wallet for transaction of digital currency

.

Experiment 2.

Create your own wallet using Metamask for crypto transactions.

Aim:

Create your own wallet using Metamask for crypto transactions.

Theory:

- **Introduction to Cryptocurrency**
  - Cryptocurrency is a digital payment system that doesn't rely on banks to verify transactions. It's a peer-to-peer system that can enable anyone anywhere to send and receive payments. Instead of being physical money carried around and exchanged in the real world, cryptocurrency payments exist purely as digital entries to an online database describing specific transactions. When you transfer cryptocurrency funds, the transactions are recorded in a public ledger. Cryptocurrency is stored in digital wallets.
  - Cryptocurrency received its name because it uses encryption to verify transactions. This means advanced coding is involved in storing and transmitting cryptocurrency data between wallets and to public ledgers. The aim of encryption is to provide security and safety.
  - The first cryptocurrency was Bitcoin, which was founded in 2009 and remains the best known today. Much of the interest in cryptocurrencies is to trade for profit, with speculators at times driving prices skyward.

- **How does cryptocurrency work?**
  - Cryptocurrencies run on a distributed public ledger called blockchain, a record of all transactions updated and held by currency holders.
  - Units of cryptocurrency are created through a process called mining, which involves using computer power to solve complicated mathematical problems that generate coins. Users can also buy the currencies from brokers, then store and spend them using cryptographic wallets.
  - If you own cryptocurrency, you don't own anything tangible. What you own is a key that allows you to move a record or a unit of measure from one person to another without a trusted third party.
  - Although Bitcoin has been around since 2009, cryptocurrencies and applications of blockchain technology are still emerging in financial terms, and more uses are expected in the future. Transactions including bonds, stocks, and other financial assets could eventually be traded using the technology.

- **Cryptocurrency examples**

  There are thousands of cryptocurrencies. Some of the best known include:

  - **Bitcoin:**

    Founded in 2009, Bitcoin was the first cryptocurrency and is still the most commonly traded. The currency was developed by Satoshi Nakamoto – widely believed to be a pseudonym for an individual or group of people whose precise identity remains unknown.

  - **Ethereum:**

    Developed in 2015, Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) or Ethereum. It is the most popular cryptocurrency after Bitcoin.

  - **Litecoin:**

    This currency is most similar to bitcoin but has moved more quickly to develop new innovations, including faster payments and processes to allow more transactions.

  - **Ripple:**

    Ripple is a distributed ledger system that was founded in 2012. Ripple can be used to track different kinds of transactions, not just cryptocurrency. Non-Bitcoin cryptocurrencies are collectively known as "altcoins" to distinguish them from the original.

- **How to store cryptocurrency**

  - Once you have purchased cryptocurrency, you need to store it safely to protect it from hacks or theft. Usually, cryptocurrency is stored in crypto wallets, which are physical devices or online software used to store the private keys to your cryptocurrencies securely. Some exchanges provide wallet services, making it easy for you to store directly through the platform. However, not all exchanges or brokers automatically provide wallet services for you.
  - There are different wallet providers to choose from. The terms "hot wallet" and "cold wallet" are used:
    - **Hot wallet storage:** "hot wallets" refer to crypto storage that uses online software to protect the private keys to your assets.
    - **Cold wallet storage:** Unlike hot wallets, cold wallets (also known as hardware wallets) rely on offline electronic devices to securely store your private keys.

Conclusion:

In this way we have explored Concept Cryptocurrency and learn how transactions are done using digital currency

.

Experiment 3.

Write a smart contract on a test network, for Bank account of a customer for following Operations.

Aim:

Smart Contract on Deposit money Withdraw Money Show balance

Theory:

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;
 }
}
```

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;
 function deposit(uint256 amount) public payable {
     require(msg.value == amount);
      balanceOf[msg.sender] += amount;
 }
}
```

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. Note also that this contract doesn't need a constructor. There is no persistent state to initialize other than the balance Of mapping, which already provides default values of 0. Given the balance Of 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. 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.) To avoid this sort of reentrancy bug, follow the "Checks-Effects-Interactions pattern" as described in the Solidity documentation. The withdraw() function above is an example of implementing this pattern

Experiment 4.

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.

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.

Theory:

- **Overview**

  This article is intended for developers new to Ethereum development. In this article, we will talk about Solidity and smart contracts, What they are and what role they actually play in the ethereum development with the end goal of writing a smart contract using Solidity

- **What is Ethereum?**

  Before getting started with smart contracts or Solidity let us first get an overview of what Ethereum is: Ethereum is a decentralized open-source blockchain with support for a Turing-complete programming language, Solidity. What we normally call computer programs are called smart contracts in Ethereum. Launched in 2015, Ethereum has been gaining significant popularity the last 5 years. It is a permissionless, public blockchain, which means anyone can use the blockchain, though for every write operation you need to pay ETH (Gas).

- **What is a Smart Contract?**

  The smart contract term was coined by Nick szabo in 1997. Smart contracts are nothing but programs that exist on the blockchain. These "smart contracts" can be used by making outside method calls or calls from other smart contracts. These smart contracts execute in the EVM (Ethereum virtual machine). Smart contracts can reduce malicious exceptions, fraudulent losses, and a need for trusted intermediator, when properly written and audited.

  Ethereum supports Turing complete smart contracts, which means you can perform almost any type of operation you want. (Remember, you need to pay ETH for every write operation).

- **What is Solidity?**

As we saw that smart contracts are nothing but programs and you need a programming language to write these programs. Ethereum core contributors invented a programming language called Solidity to write smart contracts (aka computer programs that run on the blockchain). Solidity is a high-level, object-oriented language inspired by JavaScript, C++, and Python - it has syntax very similar to JavaScript. There are other blockchains and Ethereum forks that support Solidity - such as Tron. Solidity is not the only language you can use to write smart contracts though. There are other languages that can be used to write smart contracts, like Vyper, the most popular and adopted smart contract language after solidity.

Your first Smart contract

Now, let's write a simple smart contract. Our contract will allow us to store an unsigned integer and retrieve it.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.0 <0.7.0;
contract SimpleStorage {
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public view returns (uint) {
        return storedData;
    }
}
```

The code snippet above is a smart contract written in Solidity language. Let's take a moment to understand what the code we wrote in our smart contract is doing line by line.

Line 1: Specifying SPDX license type, which is an addition after Solidity ^0.6.8; whenever the source code of a smart contract is made available to the public, these licenses can help resolve/avoid copyright issues. If you do not wish to specify any license type, you can use a special value UNLICENSED or simply skip the whole comment (it won't result in an error, just a warning).

Line 2: On the first line we are declaring which Solidity compiler we want to use. For instance, we are targeting any version between ≥ 0.4.0 and <0.7.0 .

Line 3: We are declaring our contract here and naming it as Simplestorage. It is normal practice to use the same filename as the contract name. For example - this contract will be saved in the file name SimpleStorage.sol (.sol is the file extension for solidity smart contracts).

Line 4: We are declaring a uint (Unsigned Integer) variable named storedData, this variable will be used to store data.

Line 5-7: Next, we will add a set function, using which we will change the value of our variable storeData. Here set function is accepting a parameter x whose value, we are storing into storeData. In addition, the function is marked as public which means that the function can be called by anyone.

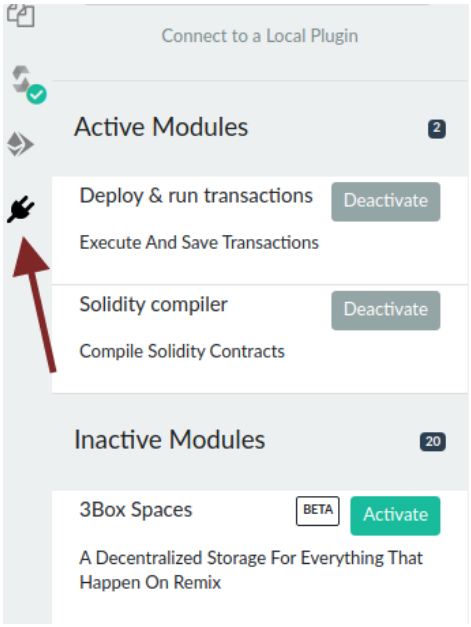
Line 8-10: We will add a get function to retrieve the value of storeData variable. This function is marked as view which tells Solidity compiler that this is a read-only function.

Other than that the get function also has returns (uint), which means that the function will return a uint.

- **Deploying the Smart contract**

After writing a smart contract it needs to be deployed on the ethereum network, we will deploy our smart contract using Remix. There are other ways to deploy smart contracts, but to make it beginner-friendly, we will use Remix. Remix is an online web Ethereum IDE. It's simple and supports many functionalities. So open Remix using this link. Remix has plugins, we need to activate two plugins for compiling and deploying our smart contract which you can see in the image below.

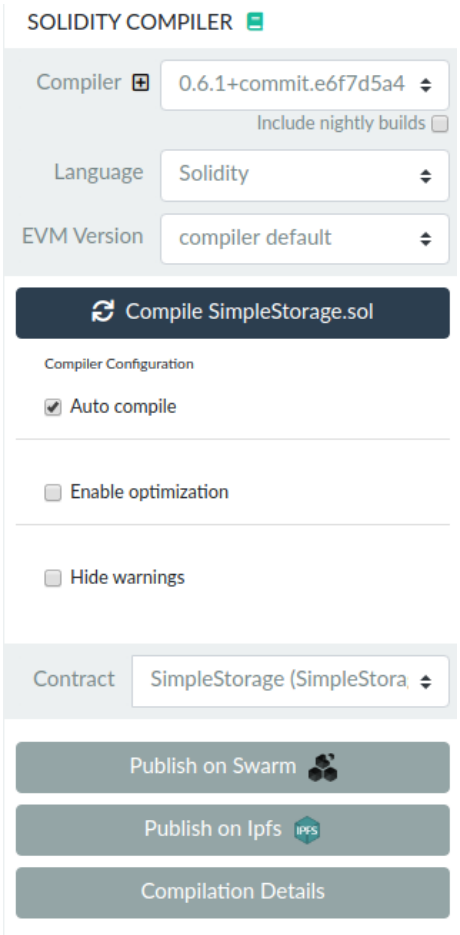
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.0 <0.7.0;
contract SimpleStorage {
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public view returns (uint) {
        return storedData;
    }
}
```



Here, you can see we have activated both plugins

Next, create a file on Remix with the name SimpleStorage.sol and copy/paste our smart contract code.

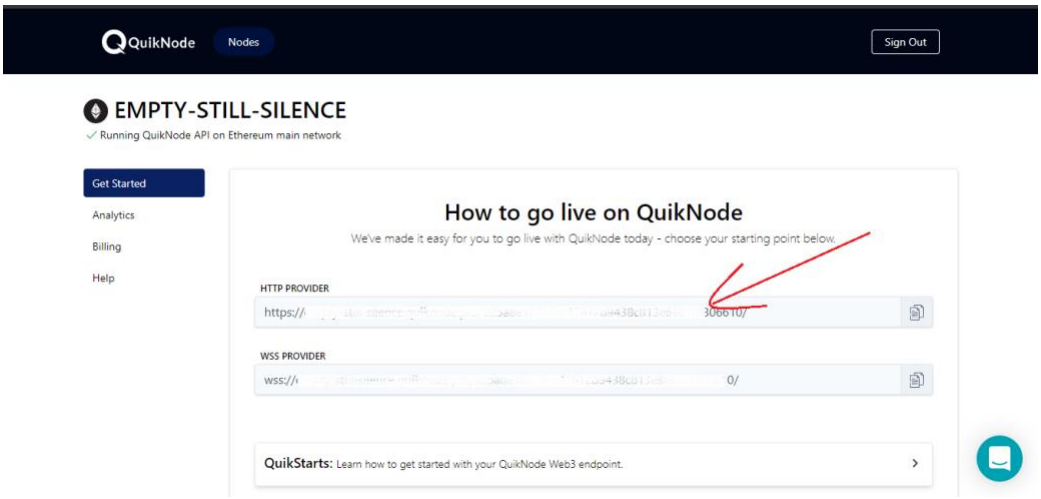
Now, lets compile our smart contract using the Remix plugin.



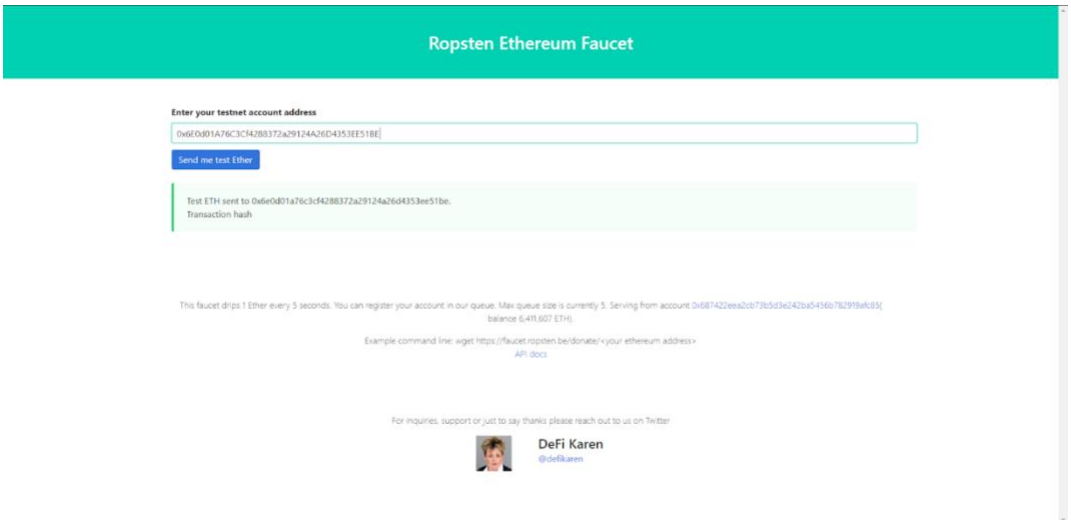
Click on "Compile SimpleStorage.sol" to compile the contract

Finally, we can now deploy our **SimpleStorage** smart contract. We will deploy our smart contract on **Ropsten testnet**. Blockchains have multiple public networks. One is the main public network, which we usually call **mainnet**. Others are for testing purposes, which we usually call **testnets**.

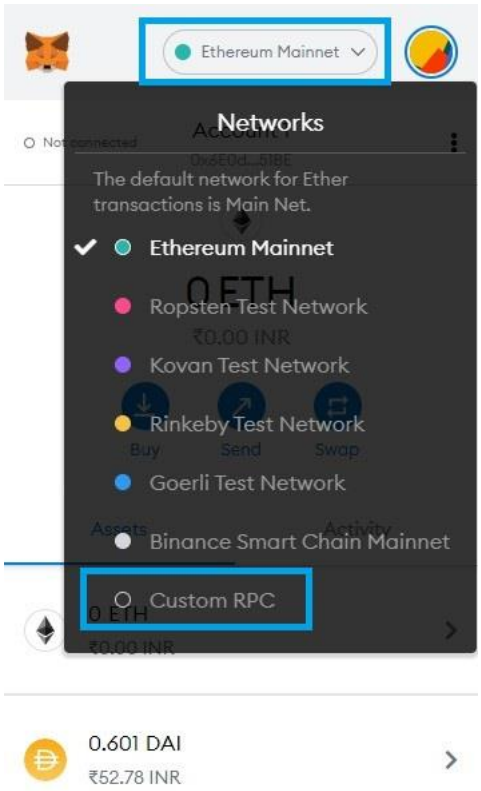
We could use pretty much any Ethereum client, such as Geth or OpenEthereum (fka Parity), for our purposes today. Since that is a bit too involved for fetching logs, we'll just grab a free endpoint from QuickNode to make this easy. We'll need a Ropsten endpoint to get data from the chain as we've deployed our contract on the Ropsten testnet. After you've created your free ethereum endpoint, copy your HTTP Provider endpoint:



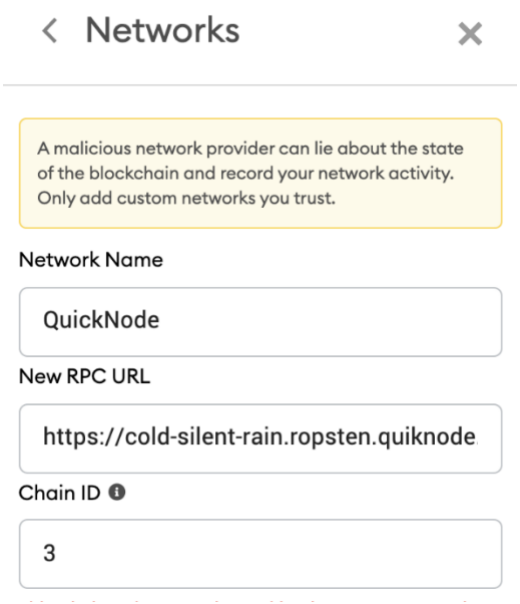
We'll deploy our contract on the Ropsten testnet. To get started, you will need the Metamask browser extension to create an ETH wallet and some test ETH, which you can get by going to the Ropsten faucet. You'll need to select Ropsten Test Network on your Metamask wallet and copy-paste the wallet address into the text field in the faucet, then click Send me test Ether.



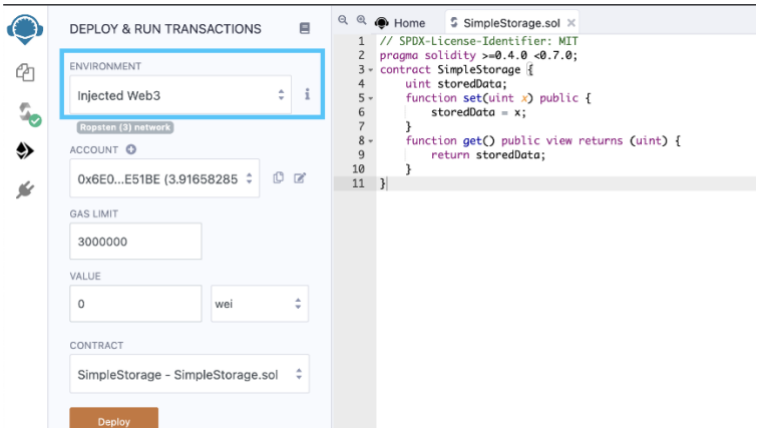
Now, let's add our node to Metamask:
Step 1: Open Metamask, click on the network menu on top, and select custom RPC.



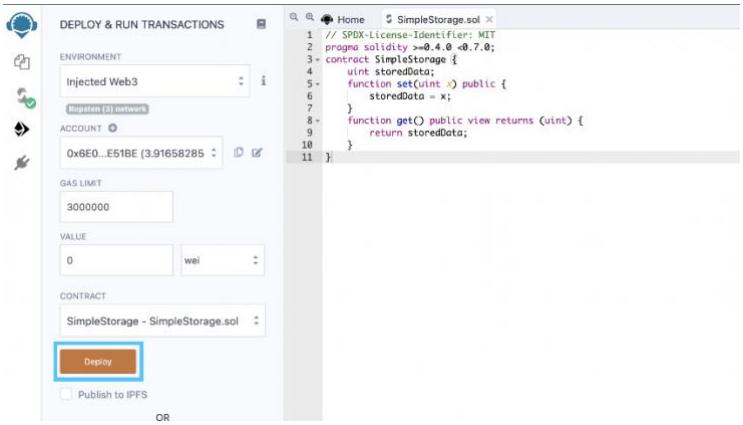
Step 2: Enter the network name (It can be any name of your choice, QuickNode in this example), Paste your QuickNode endpoint URL in the second field that says New RPC URL, Enter the Chain ID, which is also known as the network id (3 here as we are using an Ethereum Ropsten endpoint), and click on save. save.



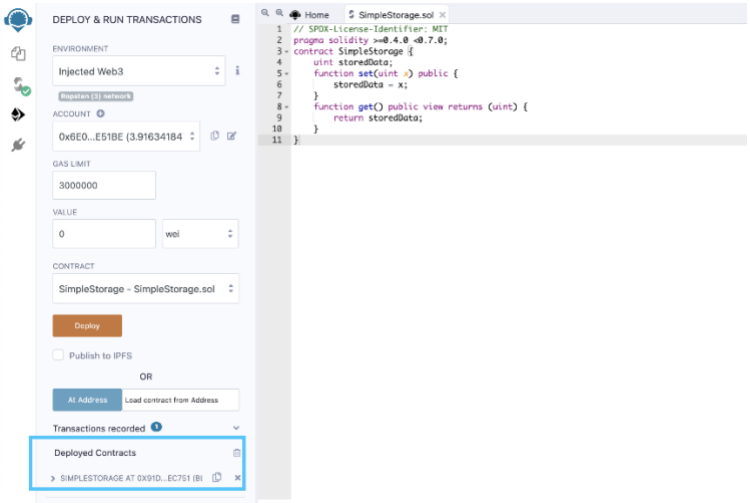
Now, go back to remix, go to the third tab on the left menu, and select Injected Web3 under the ENVIRONMENT option.



Click on deploy, and now a Metamask window must open up to confirm the transaction.



After you click on confirm, it might take some time for the transaction to get approved. Once confirmed, you must see a transaction confirmed message in the remix console and the deployed contract under the Deployed Contracts section.



Conclusion

So we have successfully created and deployed our smart contract written in Solidity to the blockchain, You can refer the Solidity Documentation for more.

Address : A/p. Loni Bk., Tal. Rahata, Dist. Ahmednagar (M.S.) PIN: 413736
Ph No.: (O) +91-2422-273539 / 273203 / (P) 273204 / (R) 273463 |
Website : www.pravara.in | Email - principal.prcenggloni@pravara.in

Experiment 5.

Mini Project

Aim:

Create a dApp (de-centralized app) for e-voting system

Theory:

A simple E-voting Decentralised App using the Ethereum Blockchain, Solidity and the MERN(MongoDB, Express.js, ReactJS, Node.js) stack

Ethereum is an open source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality.

- **D-App:**

  The E-Voting app has 2 main users:

  1. **Admin**
  2. **Voter**

  Admin can create an election and add candidates to the Ethereum Blockchain

  Users(Voters) can select an election and vote for a candidate of their choice

- **Dependencies**
  - **Node.js, Npm, React.js, Web3.js, Ganache-cli, Truffle, Solidity, MongoDB, Metamask**

- **Getting Started**

  To deploy the Smart Contract

  1. Install Ganache and create a workspace.
  2. Install Truffle npm package globally by running **`npm install -g truffle .`**
  3. Run **`truffle migrate --reset`** from the command line to deploy the smart contract to the blockchain.
  4. Download Metamask Chrome extension for the browser to help interaction between the application and the blockchain.

- **To run react development server**

```
cd blockchain
npm start
```

- **To run node server**

```
cd server
npm run dev
```

Source code : https://github.com/sherwyn11/E-Voting-App

Database:

```
const mongoose = require('mongoose')
const conn =
mongoose.connect('mongodb://127.0.0.1:27017/elections',{
    useNewUrlParser: true,
    useCreateIndex: true,
    useUnifiedTopology: true
})
```

Program:

- **Election.sol :**

```
pragma solidity ^0.5.16;

contract Election {

    struct Candidate {
        uint id;
        string name;
        uint voteCount;
        string details;
        string eid;
    }

    mapping(uint => Candidate) public cand;
    mapping(address => bool) public voters;

    uint public cC;
    string public candidate;
    constructor() public {}

    event votedEvent(
        uint indexed _cid
    );

    function addCandidate(string memory _name, string memory
_details, string memory _eid) public { cC++;
        cand[cC] = Candidate(cC, _name, 0, _details, _eid);
    }

    function vote(uint _cid) public {
        require(!voters[msg.sender]);
        require(_cid > 0 && _cid <= cC);
        voters[msg.sender] = true;
        cand[_cid].voteCount++;
        emit votedEvent(_cid);
    }

}
```

Conclusion:

Hence the dApp is created with the help of node js. And Solidity .

.