**Assignment No.1**

#import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv ( 't20\_matches.csv' )

data

# Print first 5 Rows only

data.head(5)

# Print all the columns name

print(data.columns)

# Print the Data Types of Each column

data.dtypes

data.shape

data.describe()

data.boxplot()

data.hist()

data.info()

# prepocessing steps

data.isnull().sum()

col=['round', 'home', 'away', 'winner', 'win\_by\_runs',

       'win\_by\_wickets', 'balls\_remaining', 'innings1', 'innings1\_runs',

       'innings1\_wickets', 'innings1\_overs\_batted', 'innings1\_overs',

       'innings2', 'innings2\_runs', 'innings2\_wickets',

       'innings2\_overs\_batted', 'innings2\_overs', 'D/L\_method',"result","scores"]

data.drop(col,axis=1,inplace=True)

data.head()

data.columns

data.dtypes

data.shape

#check for null values

data.isnull().sum()

data.target.fillna(data["target"].mean(),inplace=True)

data.isnull().sum()

data.describe()

data.boxplot()

data.hist()

data.info()

print(data['date'].mode())

**Assignment 2**

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from sklearn.discriminant\_analysis import StandardScaler

from sklearn.preprocessing import MinMaxScaler

#loading the dataset

data=pd.read\_csv("C:/Users/barve/Downloads/archive (6).zip")

print(data)

#a)find standard deciation and varience of each attribute

std\_dev=data.std()

print("\nStandard deviation is:\n",std\_dev)

varience=data.var()

print("\nVarience is:\n",varience)

#b)Find covariance and perform Correlation analysis using Correlation coefficient.

covarience\_matrix=data.cov()

print("\nCovarience is:\n",covarience\_matrix)

correlation\_matrix=data.corr()

print("\nCorrelation is:\n",correlation\_matrix)

#c) How many independent features are present in the given dataset?

independent\_features=np.linalg.matrix\_rank(data.corr())

print("The number of independent features are:",independent\_features)

#np.linalg is a submodule within the NumPy library that provides functions for linear algebra operations.

#np.linalg.matrix\_rank: Computes the rank of a matrix

#d) Can we identify unwanted features?

unwanted\_features = set()

for i in range(len(correlation\_matrix.columns)):

    for j in range(i):

        if abs(correlation\_matrix.iloc[i, j]) > 0.5:

            colname = correlation\_matrix.columns[i]

            unwanted\_features.add(colname)

print("\nd)Unwanted features:",unwanted\_features)

#e) Perform the data discretization using equi frequency binning method on age attribute.

num\_bins=5

data['age\_bins'] = pd.qcut(data['age'],q=num\_bins,labels=False)

#plotting equi frequency binnning for age

plt.figure(figsize=(5,3))

plt.hist(data['age'], bins=num\_bins,edgecolor='black',alpha=0.7)

plt.title("Equi frequency binning on age attribute")

plt.xlabel("Age")

plt.ylabel("Frequency")

plt.show()

#f) Normalize RestBP, chol, and MaxHR attributes (considering above dataset) using minmax normalization, Z-score normalization, and decimal scaling normalization

attributes\_to\_normalized = ['trestbps','chol','thal']

min\_max\_scaler=MinMaxScaler()

data\_min\_max\_normalized=data.copy()

data\_min\_max\_normalized[attributes\_to\_normalized]=min\_max\_scaler.fit\_transform(data[attributes\_to\_normalized])

print("\nmin max normalized in dataframes\n",data\_min\_max\_normalized.head())

#Z score

z\_score\_scaler=StandardScaler()

data\_z\_score\_normalized=data.copy()

data\_z\_score\_normalized[attributes\_to\_normalized]=z\_score\_scaler.fit\_transform(data[attributes\_to\_normalized])

print("\nZ score normalized is\n", data\_z\_score\_normalized.head())

#decimal scaling

decimal\_scaling\_factor=10\*\*(len(str(int(data[attributes\_to\_normalized].abs().max().max())))-1)

data\_decimal\_scaling=data.copy()

data\_decimal\_scaling[attributes\_to\_normalized]=data[attributes\_to\_normalized]/decimal\_scaling\_factor

print("\nDecimal scaling is\n", data\_decimal\_scaling.head())

**Assignment No.3**

**Construct FP tree**

!pip install mlxtend

!pip install apyori graphviz

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import fpgrowth

import graphviz

#Create a sample dataset:

data = [['Milk', 'Bread', 'Butter'],

        ['Milk', 'Bread'],

        ['Milk', 'Eggs'],

        ['Bread', 'Eggs'],

        ['Milk', 'Bread', 'Eggs', 'Butter'],

        ['Tea', 'Bread', 'Eggs']]

df = pd.DataFrame(data, columns=['Item1', 'Item2', 'Item3', 'Item4'])

df

#Convert the dataset to a transaction format:

te = TransactionEncoder()

te\_ary = te.fit(data).transform(data)

df\_encoded = pd.DataFrame(te\_ary, columns=te.columns\_)

df\_encoded

#Apply the Apriori algorithm:

frequent\_itemsets\_apriori = apriori(df\_encoded, min\_support=0.33, use\_colnames=True)

frequent\_itemsets\_apriori

#Apply the FP-growth algorithm:

frequent\_itemsets\_fpgrowth = fpgrowth(df\_encoded, min\_support=0.33, use\_colnames=True)

frequent\_itemsets\_fpgrowth

#Construction of FP-Tree

class Node:

    def \_\_init\_\_(self, item, count, parent):

        self.item = item

        self.count = count

        self.parent = parent

        self.children = {}

def build\_tree(data, min\_support):

    header\_table = {}

    for index, row in data.iterrows():

        for item in row:

            header\_table[item] = header\_table.get(item, 0) + 1

    for k in list(header\_table):

        if header\_table[k] < min\_support:

            del header\_table[k]

    frequent\_items = list(header\_table.keys())

    frequent\_items.sort(key=lambda x: header\_table[x], reverse=True)

    root = Node("Null", 1, None)

    for index, row in data.iterrows():

        ordered\_items = [item for item in frequent\_items if item in row]

        if ordered\_items:

            insert\_tree(ordered\_items, root, header\_table, 1)

    # Ensure 'Null' is in header\_table

    if 'Null' not in header\_table:

        header\_table['Null'] = (0, None)

    return root, header\_table

def insert\_tree(items, node, header\_table, count):

    if not items:

        return

    if items[0] in node.children:

        node.children[items[0]].count += count

    else:

        node.children[items[0]] = Node(items[0], count, node)

        if header\_table[items[0]][1] is None:

            header\_table[items[0]] = (header\_table[items[0]][0], node.children[items[0]])

        else:

            update\_header(header\_table[items[0]][1], node.children[items[0]])

    if len(items) > 1:

        insert\_tree(items[1:], node.children[items[0]], header\_table, count)

def update\_header(node\_to\_test, target\_node):

    while node\_to\_test.nodeLink is not None:

        node\_to\_test = node\_to\_test.nodeLink

    node\_to\_test.nodeLink = target\_node

# FP-tree construction

root, header\_table = build\_tree(df, min\_support=2)

# Visualize the FP-tree

def visualize\_tree(node, graph, parent\_name, graph\_name):

    if node is not None:

        graph.node(graph\_name, f"{node.item} ({node.count})", shape="box")

        if parent\_name is not None:

            graph.edge(parent\_name, graph\_name)

        for child\_key, child\_node in node.children.items():

            visualize\_tree(child\_node, graph, graph\_name, f"{graph\_name}\_{child\_key}")

# Create a graph using Graphviz

fp\_tree\_graph = graphviz.Digraph('FP\_Tree', node\_attr={'shape': 'box'}, graph\_attr={'rankdir': 'TB'})

visualize\_tree(root, fp\_tree\_graph, None, 'Root')

# Display the FP-tree visualization

fp\_tree\_graph.render(filename='fp\_tree\_visualization', format='png', cleanup=True)

fp\_tree\_graph

#Answer the questions based on the FP-tree:

# a) Find maximum frequent itemset

max\_frequent\_itemset\_fp = frequent\_itemsets\_fpgrowth[frequent\_itemsets\_fpgrowth['support'] == frequent\_itemsets\_fpgrowth['support'].max()]

print("a) Maximum Frequent Itemset (FP-growth):\n", max\_frequent\_itemset\_fp)

max\_frequent\_itemset\_apriori = frequent\_itemsets\_apriori[frequent\_itemsets\_apriori['support'] == frequent\_itemsets\_apriori['support'].max()]

print("a) Maximum Frequent Itemset (Apriori):\n", max\_frequent\_itemset\_apriori)

# b) How many transactions does it contain?

num\_transactions = len(df)

print("b) Number of transactions in the dataset:", num\_transactions)

# c) Simulate frequent pattern enumeration based on the FP-tree constructed.

def mine\_patterns(node, prefix, header\_table, min\_support, patterns):

    if header\_table[node.item][0] >= min\_support:

        patterns.append(prefix + [node.item])

    for child\_key, child\_node in node.children.items():

        mine\_patterns(child\_node, prefix + [node.item], header\_table, min\_support, patterns)

patterns\_apriori = list(frequent\_itemsets\_apriori['itemsets'])

print("c) Frequent Patterns Enumerated (Apriori):\n", patterns\_apriori)

patterns\_fp = []

mine\_patterns(root, [], header\_table, min\_support=2, patterns=patterns\_fp)

print("c) Frequent Patterns Enumerated (FP-growth):\n", patterns\_fp)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv("t20\_matches.csv")

# a) Find Missing Values and replace them with a suitable alternative.

missing\_values = data.isnull().sum()

print("Missing Values:")

print(missing\_values)

# b) Remove inconsistency (if any) in the dataset.

# There are various ways to handle inconsistencies, depending on the nature of the data.

# We'll assume there are no inconsistencies in this dataset for simplicity.

# c) Prepare boxplot analysis for each numerical attribute. Find outliers (if any) in each attribute in the dataset.

# numerical\_attributes = data.select\_dtypes(include=[np.number]).columns.tolist()

# plt.figure(figsize=(12, 8))

# for  i, col in enumerate(numerical\_attributes, 1):

#     plt.subplot(2, 2, i)

#     data.boxplot(col)

#     plt.title(col)

# plt.tight\_layout()

# plt.show()

# d) Draw a histogram for any two suitable attributes (e.g., age and Chol attributes for the above dataset).

plt.figure(figsize=(10, 5))

plt.hist(data['match\_id'], bins=20, alpha=0.7, label='match\_id')

plt.hist(data['series\_id'], bins=20, alpha=0.7, label='series\_id')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.title('Histogram of match\_id and series\_id')

plt.legend()

plt.show()

# e) Find the data type of each column.

data\_types = data.dtypes

print("\nData Types:")

print(data\_types)

# f) Finding out Zeros.

zeros\_count = (data == 0).sum()

print("\nZeros Count:")

print(zeros\_count)

# g) Find the mean age of patients considering the above dataset.

mean\_age = data['match\_id'].mean()

print("\nMean :", mean\_age)

# h) Find the shape of the data.

data\_shape = data.shape

print("\nData Shape:", data\_shape)

**Assignment 6**