EXP 3. Implementation of Data Preprocessing using Python / Scilab:

a. Data Cleaning

b. Handling Missing Data

c. Data Transformation, etc

Code:

import pandas as pd

import numpy as np

# Read

df = pd.read\_csv('exp3.csv')

# Display the original dataset

print("Original Dataset:")

print(df)

print("\n")

df\_cleaned = df.dropna()

# Display the cleaned dataset

print("Cleaned Dataset (Removed Rows with Missing Values):")

print(df\_cleaned)

print("\n")

# b. Handling Missing Data: Filling missing values with mean

numeric\_columns = df.select\_dtypes(include=[np.number]).columns

df\_filled = df.copy()

df\_filled[numeric\_columns] = df\_filled[numeric\_columns].fillna(df\_filled[numeric\_columns].mean())

# Display the dataset with missing values filled

print("Dataset with Missing Values Filled (Using Mean):")

print(df\_filled)

print("\n")

# c. Data Transformation: Adding a new column (e.g., Adjusted Salary)

df\_filled['Adjusted Salary'] = df\_filled['Salary'] \* 1.1

# Display the transformed dataset

print("Transformed Dataset (Added Adjusted Salary Column):")

print(df\_filled)

print("\n")

# Save the cleaned and transformed DataFrames to CSV files

df\_cleaned.to\_csv('cleaned\_data\_experiment.csv', index=False)

df\_filled.to\_csv('transformed\_data\_experiment.csv', index=False)

exp3.csv

name,age,Salary

John,,70689

Jane,45,45814

Michael,28,57621

Emily,59,35312

David,21,

Sophia,50,62975

William,,40996

Olivia,32,

James,27,54771

Emma,49,

Benjamin,29,65692

Isabella,42,77143

Alexander,,34226

Ava,20,47235

Lucas,38,

EXP4:

Implementation of Data Visualization and Statistical Data Analysis using

Python / Scilab:

1. Plotting Bar Charts, Histograms, Scatter Plots, etc.

Code:

import pandas as pd

import matplotlib.pyplot as plt

file\_path = 'exp4a.csv'

logistics\_data = pd.read\_csv(file\_path)

print("Logistics Data:")

print(logistics\_data)

#fig, axes = plt.subplots(2, 2, figsize=(15, 10))

plt.subplot(2,2,1)

#  Bar Chart for Shipment Quantities

plt.bar(logistics\_data['Shipment\_Number'], logistics\_data['Shipment\_Quantity'], color='skyblue')

plt.title('Bar Chart - Shipment Quantities')

plt.xlabel('Shipment Number')

plt.ylabel('Quantity')

#  Histogram for Delivery Times

plt.subplot(2,2,2)

plt.hist(logistics\_data['Delivery\_Time'], bins=10, color='lightcoral', edgecolor='black')

plt.title('Histogram - Delivery Times')

plt.xlabel('Delivery Time (days)')

plt.ylabel('Frequency')

#  Scatter Plot for Delivery Times vs. Transportation Costs

plt.subplot(2,2,3)

plt.scatter(logistics\_data['Delivery\_Time'], logistics\_data['Transportation\_Cost'], color='gold', alpha=0.7)

plt.title('Scatter Plot - Delivery Times vs. Transportation Costs')

plt.xlabel('Delivery Time (days)')

plt.ylabel('Transportation Cost ($)')

#  Box Plot for Shipment Quantities

plt.subplot(2,2,4)

plt.boxplot(logistics\_data['Shipment\_Quantity'])

plt.title('Box Plot - Shipment Quantities')

plt.ylabel('Quantity')

plt.tight\_layout()

plt.show()

EXP4a.csv

Shipment\_Number,Shipment\_Quantity,Delivery\_Time,Transportation\_Cost

1,26,1.624345364,118.396065

2,74,2.100929867,122.9728348

3,13,2.438059164,107.2614127

4,84,2.669315537,112.5069807

5,42,2.133769443,119.7517364

6,80,1.153912474,144.3598846

7,21,2.045444058,136.8729388

8,15,2.193998486,152.5567516

9,37,2.647688538,97.07874804

10,75,2.500643054,147.9533938

11,95,3.761037734,71.20570359

12,23,1.9071052,153.2131083

13,67,1.503185526,104.979822

14,33,2.888557905,179.0616851

15,53,2.954587929,63.09049888

16,59,2.315318645,126.3071803

17,90,1.44625601,169.0919722

18,48,2.536582519,114.1177398

19,74,1.936213958,125.8704679

20,47,2.033697892,148.1742046

21,97,3.864436202,92.4530306

22,35,1.412912161,92.27597177

23,58,1.866231583,178.5919114

24,52,1.442257653,141.8909015

25,39,2.332454159,77.97593686

26,99,1.92909056,121.7216828

27,43,2.83532232,179.1977165

28,45,1.505420813,157.7331485

29,89,2.495640398,77.63359438

30,72,3.58571075,98.20919745

31,17,1.926422013,171.9262488

32,97,2.400878211,56.66749184

33,49,2.232277642,128.6713

34,42,1.721298223,143.9603707

35,13,1.333674327,76.89257192

36,44,1.745056271,167.1371968

37,47,2.74290611,68.90878155

38,49,2.517457482,107.4130143

39,74,1.294493159,179.1711447

40,21,3.282732767,79.21429586

41,35,1.649270169,104.972541

42,62,1.351407251,64.6635422

43,20,1.840704507,180.2417737

44,98,2.342386298,161.9507813

45,27,2.731102775,138.0801168

46,55,1.885433747,76.01253835

47,19,1.426019804,53.73216331

48,45,1.847619282,111.9183686

49,84,3.108547014,69.689496

50,81,3.310537224,69.90142967

51,150,1.885433747,76.01253835

EXP4b:

1. Implementation of Chi-square test, correlation analysis, etc.

Code:  
import numpy as np

import csv

# Read data from CSV file

data = []

with open('chi.csv', newline='') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

data.append(row)

# Convert data to numpy array

data\_array = np.array(data[1:], dtype=int) # Skip header row

# Extract observed frequencies

observed = data\_array[:2, :2]

# Calculate row and column totals

row\_totals = np.sum(observed, axis=1)

col\_totals = np.sum(observed, axis=0)

# Calculate total observations

total\_obs = np.sum(observed)

# Calculate expected frequencies

expected = np.outer(row\_totals, col\_totals) / total\_obs

# Calculate the chi-square statistic

chi\_square = np.sum((observed - expected)\*\*2 / expected)

# Calculate degrees of freedom

df = (observed.shape[0] - 1) \* (observed.shape[1] - 1)

# Print the results

print("Chi-Square Test:")

print("Observed Frequencies:")

print(observed)

print("\nExpected Frequencies:")

print(expected)

print("\nChi-square Statistic:", chi\_square)

print("Degrees of Freedom:", df)

# Define the variables for correlation analysis

right\_handed = data\_array[:, 0]

left\_handed = data\_array[:, 1]

# Calculate means

mean\_right = np.mean(right\_handed)

mean\_left = np.mean(left\_handed)

# Calculate covariance

covariance = np.sum((right\_handed - mean\_right) \* (left\_handed - mean\_left)) / (len(right\_handed) - 1)

# Calculate standard deviations

std\_dev\_right = np.std(right\_handed, ddof=1)

std\_dev\_left = np.std(left\_handed, ddof=1)

# Calculate correlation coefficient

correlation\_coefficient = covariance / (std\_dev\_right \* std\_dev\_left)

# Print the results

print("\nCorrelation Analysis (Pearson):")

print("Correlation Coefficient:", correlation\_coefficient)

exp4.csv chisquere

,Right-handed,Left-handed,Total\_hands

American,236,19,255

Canadian,155,15,170

Total\_person,391,34,425

New code

import pandas as pd

from scipy.stats import chi2\_contingency

from scipy.stats import pearsonr

import numpy as np

# Read the CSV file

df = pd.read\_csv('exp4b.csv')

# Display the DataFrame

print("Original DataFrame:")

print(df)

# Extract the data for analysis (excluding the 'Total\_person' row)

data = df.iloc[:-1, 1:].astype(int)  # Exclude the first column and last row

# Perform Chi-square test for independence

chi2, p\_value, dof, \_ = chi2\_contingency(data, correction=False)

# Calculate Chi-square statistic

chi2\_statistic = np.sqrt(chi2)

# Calculate Pearson correlation coefficient

correlation\_coefficient, \_ = pearsonr(df['Right-handed'], df['Left-handed'])

# Display the results

print("\nChi-square Test Results:")

print(f"Chi-square value: {chi2}")

print(f"Chi-square statistic: {chi2\_statistic}")

print(f"P-value: {p\_value}")

print(f"Degrees of Freedom: {dof}")

# Interpret the results

alpha = 0.05  # Significance level

if p\_value < alpha:

    print("There is a significant association between handedness and nationality.")

else:

    print("There is no significant association between handedness and nationality.")

print("\nPearson Correlation Coefficient between Right-handed and Left-handed:")

print(f"{correlation\_coefficient}")

EXP5  
pip install pandas

pip install scikit-learn

pip install matplotlib

pip install pandas scikit-learn matplotlib

code:

# Importing necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score, confusion\_matrix

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv("try.csv")

# Perform one-hot encoding on categorical variables

data = pd.get\_dummies(data)

# Splitting the dataset into features (X) and target variable (y)

X = data.drop('Decision\_Yes', axis=1) # Adjust column name if needed

y = data['Decision\_Yes'] # Adjust column name if needed

# Splitting the data into training and testing sets (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Creating the Decision Tree classifier with limited depth

clf = DecisionTreeClassifier(max\_depth=3, random\_state=42)

# Training the Decision Tree classifier

clf.fit(X\_train, y\_train)

# Making predictions on the testing set

y\_pred = clf.predict(X\_test)

# Calculating accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

# Plotting the decision tree with proper size and structure

plt.figure(figsize=(15,10))

plot\_tree(clf, feature\_names=X.columns, class\_names=['No Decision', 'Decision'], filled=True, fontsize=10)

plt.show()

try.csv  
Age,Income,Education,Employment,MaritalStatus,Decision

35,50000,Bachelor,Employed,Single,Yes

45,70000,Master,Employed,Married,Yes

28,30000,High School,Unemployed,Single,No

55,80000,PhD,Employed,Married,Yes

40,60000,Bachelor,Employed,Single,Yes

30,40000,Master,Employed,Single,No

50,90000,Bachelor,Employed,Married,Yes

33,55000,Bachelor,Employed,Single,Yes

38,62000,High School,Employed,Married,Yes

48,75000,Master,Employed,Married,Yes

25,35000,High School,Unemployed,Single,No

42,68000,Bachelor,Employed,Married,Yes

31,42000,Master,Employed,Single,Yes

37,58000,Bachelor,Employed,Single,Yes

29,38000,High School,Unemployed,Single,No

34,53000,Bachelor,Employed,Married,No

44,72000,Master,Employed,Married,Yes

39,61000,Bachelor,Employed,Single,Yes

27,32000,High School,Unemployed,Single,No

51,92000,Bachelor,Employed,Married,Yes

Exp7  
Aim:Implement and evaluate any Clustering Algorithm using Python

import numpy as np

import matplotlib.pyplot as plt

class KMeans:

    def \_\_init\_\_(self, n\_clusters, max\_iter=100):

        self.n\_clusters = n\_clusters

        self.max\_iter = max\_iter

    def fit(self, X):

        n\_samples, n\_features = X.shape

        # Initialize cluster centroids randomly

        random\_idx = np.random.choice(n\_samples, self.n\_clusters, replace=False)

        self.centroids = X[random\_idx]

        # Array to store the cluster assignment for each data point

        self.labels = np.zeros(n\_samples)

        # Iterate for a maximum of self.max\_iter times

        for \_ in range(self.max\_iter):

            # Assign each data point to the nearest centroid

            self.labels = self.\_assign\_clusters(X, self.centroids)

            # Update centroids based on the mean of data points in each cluster

            new\_centroids = self.\_update\_centroids(X, self.labels)

            # Check for convergence

            if np.allclose(self.centroids, new\_centroids):

                break

            self.centroids = new\_centroids

    def \_assign\_clusters(self, X, centroids):

        distances = np.sqrt(((X - centroids[:, np.newaxis])\*\*2).sum(axis=2))

        return np.argmin(distances, axis=0)

    def \_update\_centroids(self, X, labels):

        new\_centroids = np.zeros((self.n\_clusters, X.shape[1]))

        for k in range(self.n\_clusters):

            new\_centroids[k] = np.mean(X[labels == k], axis=0)

        return new\_centroids

    def predict(self, X):

        distances = np.sqrt(((X - self.centroids[:, np.newaxis])\*\*2).sum(axis=2))

        return np.argmin(distances, axis=0)

# Example usage

np.random.seed(0)

# Generate random data points for clustering

X = np.random.rand(100, 2)

# Number of clusters

k = 3

# Create a KMeans instance

kmeans = KMeans(n\_clusters=k)

# Fit the data

kmeans.fit(X)

# Get cluster labels

labels = kmeans.predict(X)

# Plotting the clusters

plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7, edgecolors='w')

plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1], c='red', marker='x', s=200, label='Centroids')

plt.title('K-means Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()

k mean with input  
import numpy as np

import matplotlib.pyplot as plt

class KMeans:

    def \_\_init\_\_(self, n\_clusters, max\_iter=100):

        self.n\_clusters = n\_clusters

        self.max\_iter = max\_iter

    def fit(self, X):

        n\_samples, n\_features = X.shape

        # Initialize cluster centroids randomly

        random\_idx = np.random.choice(n\_samples, self.n\_clusters, replace=False)

        self.centroids = X[random\_idx]

        # Array to store the cluster assignment for each data point

        self.labels = np.zeros(n\_samples)

        # Iterate for a maximum of self.max\_iter times

        for \_ in range(self.max\_iter):

            # Assign each data point to the nearest centroid

            self.labels = self.\_assign\_clusters(X, self.centroids)

            # Update centroids based on the mean of data points in each cluster

            new\_centroids = self.\_update\_centroids(X, self.labels)

            # Check for convergence

            if np.allclose(self.centroids, new\_centroids):

                break

            self.centroids = new\_centroids

    def \_assign\_clusters(self, X, centroids):

        distances = np.sqrt(((X - centroids[:, np.newaxis])\*\*2).sum(axis=2))

        return np.argmin(distances, axis=0)

    def \_update\_centroids(self, X, labels):

        new\_centroids = np.zeros((self.n\_clusters, X.shape[1]))

        for k in range(self.n\_clusters):

            new\_centroids[k] = np.mean(X[labels == k], axis=0)

        return new\_centroids

    def predict(self, X):

        distances = np.sqrt(((X - self.centroids[:, np.newaxis])\*\*2).sum(axis=2))

        return np.argmin(distances, axis=0)

def generate\_random\_data(n\_samples, n\_features):

    return np.random.rand(n\_samples, n\_features)

if \_\_name\_\_ == "\_\_main\_\_":

    # Input parameters

    n\_samples = int(input("Enter the number of data points: "))

    n\_clusters = int(input("Enter the number of clusters: "))

    # Generate random data points

    X = generate\_random\_data(n\_samples, 2)  # Assuming 2 features for simplicity

    # Create a KMeans instance

    kmeans = KMeans(n\_clusters=n\_clusters)

    # Fit the data

    kmeans.fit(X)

    # Get cluster labels

    labels = kmeans.predict(X)

    # Plotting the clusters

    plt.figure(figsize=(8, 6))

    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7, edgecolors='w')

    plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1], c='red', marker='x', s=200, label='Centroids')

    plt.title('K-means Clustering')

    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')

    plt.legend()

    plt.show()