# Data Science

1. Write program to check whether number is Harshad number or Not.

A **Harshad number** (or Niven number) is an integer that is divisible by the sum of its digits. Here's a Python program to check whether a given number is a Harshad number:

```python
def is_harshad_number(num):
    if num <= 0:
        return False  # Harshad numbers are positive integers

    # Calculate the sum of the digits
    digit_sum = sum(int(digit) for digit in str(num))

    # Check if the number is divisible by the sum of its digits
    return num % digit_sum == 0

# Input from user
number = int(input("Enter a number: "))

# Check and display the result
if is_harshad_number(number):
    print(f"{number} is a Harshad number.")
else:
    print(f"{number} is not a Harshad number.")
```

2. Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot on given iris dataset.

### Implementation

Below is the Python code to achieve this:

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Load the income.csv dataset
income_data = pd.read_csv("income.csv")

# Assuming 'income.csv' has numerical features like 'Income' and 'Spending_Score'
# Normalize the data
scaler = StandardScaler()
normalized_income_data = scaler.fit_transform(income_data.select_dtypes(include=[np.number]))

# Apply k-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
income_clusters = kmeans.fit_predict(normalized_income_data)
```

```
# Load the Iris dataset
iris = load_iris()
iris_data = iris.data
iris_features = iris.feature_names

# Take the first two features for plotting
x = iris_data[:, 0]  # Sepal Length
y = iris_data[:, 1]  # Sepal Width

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(x, y, c=income_clusters, cmap='viridis', s=50)
plt.title("Clustering from Income Dataset Visualized on Iris Dataset")
plt.xlabel(iris_features[0])
plt.ylabel(iris_features[1])
plt.colorbar(label='Cluster Label')
plt.show()
```

3. Write a python program to count the occurrence of each word in a given sentence

```
def count_word_occurrences(sentence):
    # Convert the sentence to lowercase and split into words
    words = sentence.lower().split()

    # Create a dictionary to store word counts
    word_counts = {}

    for word in words:
        # Remove punctuation from words
        word = ''.join(char for char in word if char.isalnum())
        # Increment the count of the word in the dictionary
        if word in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1

    return word_counts

# Input sentence from the user
sentence = input("Enter a sentence: ")

# Get the word occurrences
occurrences = count_word_occurrences(sentence)

# Display the results
print("Word occurrences:")
for word, count in occurrences.items():
    print(f"{word}: {count}")
```

4. Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the income.csv dataset
# Replace 'income.csv' with the path to your dataset
data = pd.read_csv("income.csv")

# Inspect the first few rows of the dataset
print("Dataset Head:")
print(data.head())

# Select numerical columns (e.g., 'Income', 'Age')
# Replace these column names with appropriate ones from your dataset
numerical_features = ['Income', 'Spending_Score']
income_data = data[numerical_features]

# Handle missing values if necessary
income_data = income_data.dropna()

# Standardize the data
scaler = StandardScaler()
income_data_scaled = scaler.fit_transform(income_data)

# Apply k-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(income_data_scaled)

# Add cluster labels to the original data
data['Cluster'] = clusters

# Scatter plot of the clusters
plt.figure(figsize=(10, 6))
plt.scatter(
    income_data_scaled[:, 0], income_data_scaled[:, 1],
    c=clusters, cmap='viridis', s=50
)
plt.title("Clustering of Income Dataset")
plt.xlabel("Income (Standardized)")
plt.ylabel("Spending Score (Standardized)")
plt.colorbar(label="Cluster")
plt.show()
```

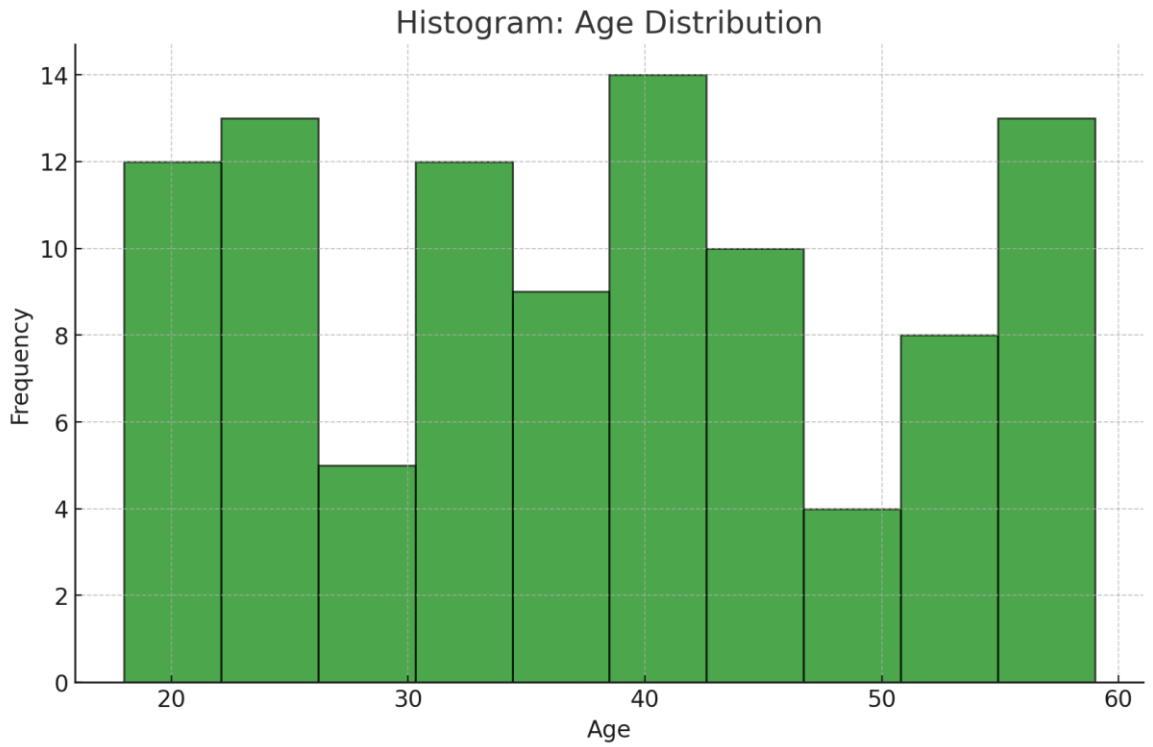
5. Write a python program to accept input string from user and display number of vowels and consonant in string.

```
def count_vowels_and_consonants(input_string):
    # Define vowels
    vowels = "aeiou"
    # Initialize counters
    vowel_count = 0
    consonant_count = 0
```

```
    # Convert the string to lowercase for uniformity
    input_string = input_string.lower()

    # Loop through each character in the string
    for char in input_string:
        if char.isalpha():  # Check if the character is a letter
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return vowel_count, consonant_count

# Input string from the user
user_input = input("Enter a string: ")

# Get counts
vowels, consonants = count_vowels_and_consonants(user_input)

# Display the results
print(f"Number of vowels: {vowels}")
print(f"Number of consonants: {consonants}")
```

6.Implement Data Visualization Kindly refers your own data.

1. Draw scatter plot diagram.
2. Draw Histogram.




Here are the visualizations:

1. **Scatter Plot**: Shows the relationship between Age and Income. Each point represents an individual's data, with Age on the x-axis and Income on the y-axis.
2. **Histogram**: Displays the distribution of Ages in the dataset, grouped into bins. It helps visualize how the ages are spread across the dataset.

7. Write a python program to accept input from user and check whether number is Armstrong or not.

## Definition of an Armstrong Number:

An **Armstrong number** (or narcissistic number) is a number that is equal to the sum of its own digits raised to the power of the number of digits. For example:

- 153=13+53+33153 = 1^3 + 5^3 + 3^3153=13+53+33 (3 digits)
- 9474=94+44+74+449474 = 9^4 + 4^4 + 7^4 + 4^49474=94+44+74+44 (4 digits)

```
def is_armstrong_number(num):
    # Convert the number to a string to get digits
    digits = str(num)
    num_digits = len(digits)

    # Calculate the sum of digits raised to the power of num_digits
    armstrong_sum = sum(int(digit) ** num_digits for digit in digits)

    # Check if the number is an Armstrong number
    return armstrong_sum == num

# Input number from the user
number = int(input("Enter a number: "))
```

```
# Check and display result
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

8. Implement Classification algorithm KNN classifier Data Analysis on given iris dataset.

The K-Nearest Neighbors (KNN) algorithm is a simple, supervised machine learning algorithm commonly used for classification. Below is an implementation of KNN for the Iris dataset using Python.

Steps to Implement KNN Classifier on Iris Dataset:

Load the Dataset:

Use the Iris dataset from sklearn.datasets.

Preprocess the Data:

Split the dataset into training and testing sets.

Standardize the features for better performance.

Implement the KNN Algorithm:

Use KNeighborsClassifier from sklearn.neighbors.

Evaluate the Model:

Measure the accuracy using the test set.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (Sepal Length, Sepal Width, Petal Length, Petal Width)
y = iris.target  # Target labels (Setosa, Versicolor, Virginica)

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3)

# Train the model
knn.fit(X_train, y_train)

# Predict on the test set
y_pred = knn.predict(X_test)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of KNN Classifier: {accuracy:.2f}\n")

# Detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

9. Implement Classification algorithm Decision tree Data Analysis on given iris dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (Sepal Length, Sepal Width, Petal Length, Petal Width)
y = iris.target  # Target labels (Setosa, Versicolor, Virginica)

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the model
dt_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = dt_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Decision Tree Classifier: {accuracy:.2f}\n")

# Detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.title("Decision Tree Visualization", fontsize=16)
plt.show()
```

10. Write a Python program to print factorial of number using Recursion.

```
def factorial(n):
    # Base case: if n is 0 or 1, return 1
    if n == 0 or n == 1:
        return 1
    else:
        # Recursive case: n * factorial(n-1)
        return n * factorial(n - 1)
```

```
# Input number from user
num = int(input("Enter a number: "))

# Ensure the number is non-negative
if num < 0:
    print("Factorial does not exist for negative numbers.")
else:
    # Calculate and display the factorial
    result = factorial(num)
    print(f"Factorial of {num} is {result}")
```

11. Implement Data Visualization Kindly refers “income.csv” dataset.
    1. Draw a scatter plot for Age and Income.
    2. Draw a bar graph for Age and Income.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset (Replace 'income.csv' with the path to your dataset)
data = pd.read_csv('income.csv')

# Check the first few rows of the dataset to ensure it's loaded correctly
print(data.head())

# 1. Scatter Plot: Age vs Income
plt.figure(figsize=(10, 6))
plt.scatter(data['Age'], data['Income'], color='blue', alpha=0.6)
plt.title('Scatter Plot: Age vs Income', fontsize=16)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Income', fontsize=12)
plt.grid(True)
plt.show()

# 2. Bar Graph: Age vs Income (Bar plot)
plt.figure(figsize=(10, 6))
plt.bar(data['Age'], data['Income'], color='green', alpha=0.6)
plt.title('Bar Graph: Age vs Income', fontsize=16)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Income', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```

12. Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot on given iris dataset.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the income.csv dataset
```

```
data = pd.read_csv("income.csv")

# Inspect the dataset
print(data.head())

# Assuming the dataset has 'Age' and 'Income' columns, we use them for clustering
X = data[['Age', 'Income']]

# Handle missing values (if any) by dropping rows with missing values
X = X.dropna()

# Standardize the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply KMeans clustering (let's choose 3 clusters for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
data['Cluster'] = kmeans.fit_predict(X_scaled)

# Plotting the clusters using scatter plot
plt.figure(figsize=(10, 6))

# Scatter plot for Age vs Income with different colors for each cluster
plt.scatter(data['Age'], data['Income'], c=data['Cluster'], cmap='viridis', s=50)
plt.title('K-Means Clustering: Age vs Income', fontsize=16)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Income', fontsize=12)
plt.colorbar(label='Cluster')  # Show color legend for clusters
plt.show()
```

13. Write a python program to count the occurrence of each word in a given sentence

```
from collections import Counter

def count_word_occurrences(sentence):
    # Split the sentence into words (using space as delimiter)
    words = sentence.split()

    # Count occurrences of each word using Counter
    word_count = Counter(words)

    return word_count

# Input sentence from the user
sentence = input("Enter a sentence: ")

# Get the word count
word_occurrences = count_word_occurrences(sentence)

# Display the word occurrences
print("Word occurrences:")
for word, count in word_occurrences.items():
    print(f"'{word}': {count}")
```

14. Implement Classification algorithm SVM Classifier Data Analysis on given iris dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target labels (Setosa, Versicolor, Virginica)

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the SVM Classifier
svm_classifier = SVC(kernel='linear', random_state=42)  # Linear kernel for simplicity

# Train the model
svm_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = svm_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of SVM Classifier: {accuracy:.2f}\n")

# Detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Optional: Visualizing the decision boundaries for the first two features
# We'll use only the first two features for 2D visualization
X_train_2d = X_train[:, :2]
X_test_2d = X_test[:, :2]

# Train the SVM Classifier again with only 2 features for visualization
svm_classifier.fit(X_train_2d, y_train)

# Plotting the decision boundary
h = .02  # Step size in the mesh
x_min, x_max = X_train_2d[:, 0].min() - 1, X_train_2d[:, 0].max() + 1
y_min, y_max = X_train_2d[:, 1].min() - 1, X_train_2d[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_train_2d[:, 0], X_train_2d[:, 1], c=y_train, edgecolors='k', marker='o', cmap=plt.cm.coolwarm)
plt.title("SVM Decision Boundary (First Two Features)")
plt.xlabel('Sepal Length')
```

```
plt.ylabel('Sepal Width')
plt.show()
```

15. Write a program to print length of String using Recursion.

```
def string_length(s):
    # Base case: if the string is empty, return 0
    if s == "":
        return 0
    else:
        # Recursive case: 1 + length of the substring excluding the first character
        return 1 + string_length(s[1:])

# Input string from the user
input_string = input("Enter a string: ")

# Call the function to get the length of the string
length = string_length(input_string)

# Print the length of the string
print(f"The length of the string is: {length}")
```

16. Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot on given iris dataset.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load the income.csv dataset (Replace with the actual file path)
income_data = pd.read_csv("income.csv")

# Inspect the first few rows of the dataset
print(income_data.head())

# Assuming the dataset has 'Age' and 'Income' columns
X_income = income_data[['Age', 'Income']]

# Standardize the data (optional but recommended for K-Means)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_income)

# Apply K-Means clustering (let's choose 3 clusters for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
income_data['Cluster'] = kmeans.fit_predict(X_scaled)

# Load the Iris dataset for visualization
iris = load_iris()
X_iris = iris.data[:, :2]  # Using only the first two features: Sepal Length and Sepal Width
y_iris = iris.target  # Target labels for the Iris dataset
```

```
# Visualize the clusters (we will plot the clusters using the first two features of Iris dataset)
plt.figure(figsize=(10, 6))

# Scatter plot for Iris data (for the first two features)
plt.scatter(X_iris[:, 0], X_iris[:, 1], c=income_data['Cluster'], cmap='viridis', edgecolor='k', s=50)

plt.title('K-Means Clustering on Income Data Visualized with Iris Dataset', fontsize=14)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.colorbar(label='Cluster')
plt.show()
```

17. Write a python program to print prime number between 1 to 100.

```
def is_prime(num):
    # Check if a number is prime
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):  # Loop up to the square root of the number
        if num % i == 0:
            return False
    return True

# Print prime numbers between 1 and 100
print("Prime numbers between 1 and 100 are:")
for num in range(1, 101):
    if is_prime(num):
        print(num, end=" ")
```

18. Implement Classification algorithm Decision Tree Classifier Data Analysis on given iris dataset

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn import tree
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target labels (Setosa, Versicolor, Virginica)

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the model
dt_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = dt_classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Decision Tree Classifier: {accuracy:.2f}\n")

# Detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Visualizing the Decision Tree
plt.figure(figsize=(12, 8))
tree.plot_tree(dt_classifier, filled=True, feature_names=iris.feature_names, class_names=iris.target_names,
fontsize=10)
plt.title("Decision Tree Classifier on Iris Dataset")
plt.show()
```

19. Write a python program to Implement Data Visualization Kindly refers your own data.
    1. Draw a horizontal bar graph.
    2. Draw scatter plot diagram.

```
import matplotlib.pyplot as plt
import numpy as np

# Custom data for demonstration
categories = ['A', 'B', 'C', 'D', 'E']
values = [15, 30, 45, 10, 25]

# 1. Draw a horizontal bar graph
plt.figure(figsize=(10, 6))
plt.barh(categories, values, color='skyblue')
plt.xlabel('Values')
plt.ylabel('Categories')
plt.title('Horizontal Bar Graph')
plt.show()

# 2. Draw a scatter plot
# Let's create another custom dataset for scatter plot
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 5, 7, 11])

plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='red', marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')
plt.grid(True)
plt.show()
```

20. Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot on given iris dataset

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

```
# Load the 'income.csv' dataset
# (replace 'income.csv' with the actual path to the file)
income_data = pd.read_csv("income.csv")

# Inspect the first few rows of the dataset
print(income_data.head())

# Assuming 'Age' and 'Income' are columns in the income dataset
X_income = income_data[['Age', 'Income']]  # Select features for clustering

# Standardize the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_income)

# Apply K-Means clustering (let's choose 3 clusters for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
income_data['Cluster'] = kmeans.fit_predict(X_scaled)

# Load the Iris dataset for visualization
iris = load_iris()
X_iris = iris.data[:, :2]  # Using only the first two features: Sepal Length and Sepal Width
y_iris = iris.target  # Target labels for the Iris dataset

# Visualize the clusters (we will plot the clusters using the first two features of the Iris dataset)
plt.figure(figsize=(10, 6))

# Scatter plot for Iris data (for the first two features)
plt.scatter(X_iris[:, 0], X_iris[:, 1], c=income_data['Cluster'], cmap='viridis', edgecolor='k', s=50)

# Adding plot title and labels
plt.title('K-Means Clustering on Income Data Visualized Using Iris Dataset', fontsize=14)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.colorbar(label='Cluster')
plt.show()
```

21. Implement Data Visualization Kindly refers “Salary_Data.csv” dataset.
    1. Draw a Scatter plot for Age and Salary.
    2. Draw Histogram for Salary.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the "Salary_Data.csv" dataset
# Replace 'Salary_Data.csv' with the actual path to your CSV file
salary_data = pd.read_csv("Salary_Data.csv")

# Inspect the first few rows of the dataset to ensure it contains 'Age' and 'Salary'
print(salary_data.head())

# 1. Scatter plot for Age and Salary
plt.figure(figsize=(10, 6))
plt.scatter(salary_data['Age'], salary_data['Salary'], color='b', edgecolor='k', alpha=0.7)
plt.title('Scatter Plot of Age vs Salary', fontsize=14)
plt.xlabel('Age')
```

```
plt.ylabel('Salary')
plt.grid(True)
plt.show()

# 2. Histogram for Salary
plt.figure(figsize=(10, 6))
plt.hist(salary_data['Salary'], bins=10, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Histogram of Salary', fontsize=14)
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

22. Write a Python program to print factorial of number using Recursion.

```
# Function to calculate factorial using recursion
def factorial(n):
    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    # Recursive case: n * factorial of (n-1)
    else:
        return n * factorial(n - 1)

# Accept user input
number = int(input("Enter a number to find its factorial: "))

# Call the recursive function and display the result
result = factorial(number)
print(f"The factorial of {number} is {result}")
```

23. Write python program to accept number and check whether number is Armstrong or not.

```
# Function to check whether the number is Armstrong or not
def is_armstrong(num):
    # Convert the number to string to easily get digits
    num_str = str(num)
    num_digits = len(num_str)  # Number of digits in the number

    # Calculate the sum of each digit raised to the power of number of digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

    # Check if the sum equals the original number
    if sum_of_powers == num:
        return True
    else:
        return False

# Accept user input
number = int(input("Enter a number to check if it is Armstrong: "))

# Check and print the result
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
```

```
    print(f"{number} is not an Armstrong number.")
```

24. Write a Python program to print factorial of number using Recursion.

```
# Function to calculate factorial using recursion
def factorial(n):
    # Base case: if n is 0 or 1, return 1
    if n == 0 or n == 1:
        return 1
    # Recursive case: n * factorial of (n-1)
    else:
        return n * factorial(n - 1)

# Accept user input
number = int(input("Enter a number to find its factorial: "))

# Call the recursive function and display the result
result = factorial(number)
print(f"The factorial of {number} is {result}")
```

25. Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot on given iris dataset.

```
    import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Load the 'income.csv' dataset
# Make sure to replace 'income.csv' with the correct path to your CSV file
income_data = pd.read_csv('income.csv')

# Inspect the first few rows of the dataset to ensure it contains 'Age' and 'Income'
print(income_data.head())

# Assuming the columns in income.csv are 'Age' and 'Income' for clustering
X_income = income_data[['Age', 'Income']]

# Standardize the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_income)

# Apply K-Means clustering (let's choose 3 clusters for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
income_data['Cluster'] = kmeans.fit_predict(X_scaled)

# Load the Iris dataset for visualization
iris = load_iris()
X_iris = iris.data[:, :2]  # Using only the first two features: Sepal Length and Sepal Width
y_iris = iris.target  # Target labels for the Iris dataset

# Visualize the clusters (we will plot the clusters using the first two features of the Iris dataset)
```

```
plt.figure(figsize=(10, 6))

# Scatter plot for Iris data (for the first two features)
plt.scatter(X_iris[:, 0], X_iris[:, 1], c=income_data['Cluster'], cmap='viridis', edgecolor='k', s=50)

# Adding plot title and labels
plt.title('K-Means Clustering on Income Data Visualized Using Iris Dataset', fontsize=14)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.colorbar(label='Cluster')
plt.show()
```