//SLIP NO:-2///

Q1. Write an R program to calculate the multiplication table using afunction.

====>

```
# Define a function to generate a multiplication table
multiplication_table <- function(number, range) {
  cat("Multiplication Table for", number, "\n")
  for (i in 1:range) {
    result <- number * i
    cat(number, "*", i, "=", result, "\n")
  }
}

# Set the number and range for the multiplication table
number <- 5  # You can change this to any number
range <- 10  # You can change this to set the range

# Call the function
multiplication_table(number, range)
```

O/P:-

```
Multiplication Table for 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Q2. Write a python program to implement k-means algorithms on asynthetic dataset.

====>

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate synthetic dataset
n_samples = 300
n_features = 2
n_clusters = 4
random_state = 42

X, y = make_blobs(n_samples=n_samples, centers=n_clusters, n_features=n_features,
random_state=random_state)

# Apply K-means algorithm
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
kmeans.fit(X)

# Get cluster centers and labels
centers = kmeans.cluster_centers_
labels = kmeans.labels_
```

```
# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')  # Cluster centers
plt.title('K-means Clustering on Synthetic Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.grid()
plt.show()
```

//SLIP NO:-3//

Q1. Write a R program to reverse a number and also calculate the sum ofdigits of that number.

====>

```
# Function to reverse a number and calculate the sum of its digits
reverse_and_sum_digits <- function(number) {
  # Convert the number to a string to facilitate reversal
  num_str <- as.character(number)

  # Reverse the string
  reversed_str <- rev(strsplit(num_str, "")[[1]])
  reversed_number <- as.numeric(paste(reversed_str, collapse = ""))

  # Calculate the sum of the digits
  digit_sum <- sum(as.numeric(reversed_str))

  # Return the reversed number and the sum of digits
  return(list(reversed_number = reversed_number, digit_sum = digit_sum))
}
# Example usage
number <- 12345  # You can change this number
result <- reverse_and_sum_digits(number)

# Print the results
cat("Original Number:", number, "\n")
cat("Reversed Number:", result$reversed_number, "\n")
cat("Sum of Digits:", result$digit_sum, "\n")
```

O/P:- Original Number: 12345
    Reversed Number: 54321
    Sum of Digits: 15

Q2. Consider the following observations/data. And apply simple linear regression and find out estimated coefficients b0 and b1.(use numpypackage)
x=[0,1,2,3,4,5,6,7,8,9,11,13]
y = ([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,16, 18]

====>

```
import numpy as np
# Given data
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12, 16, 18])

# Number of observations
```

```
n = len(x)

# Calculate the necessary sums
sum_x = np.sum(x)
sum_y = np.sum(y)
sum_xy = np.sum(x * y)
sum_x_squared = np.sum(x**2)

# Calculate coefficients b1 and b0
b1 = (n * sum_xy - sum_x * sum_y) / (n * sum_x_squared - sum_x**2)
b0 = (sum_y - b1 * sum_x) / n

# Print the results
print("Estimated coefficients:")
print(f"b0 (Intercept): {b0}")
print(f"b1 (Slope): {b1}")
```

O/P:-
Estimated coefficients:
b0 (Intercept): 0.8857142857142857
b1 (Slope): 1.2857142857142858
//SLIP NO:-4//
Q1. Write a R program to calculate the sum of two matrices of given size.
====>

```
# Function to calculate the sum of two matrices
sum_of_matrices <- function(matrix1, matrix2) {
  return(matrix1 + matrix2)
}

# Define the size of the matrices
rows <- as.integer(readline(prompt = "Enter the number of rows: "))
cols <- as.integer(readline(prompt = "Enter the number of columns: "))

# Initialize matrices
cat("Enter the elements for the first matrix:\n")
matrix1 <- matrix(nrow = rows, ncol = cols)
for (i in 1:rows) {
  for (j in 1:cols) {
    matrix1[i, j] <- as.numeric(readline(prompt = paste("Element [", i, ",", j, "]: ")))
  }
}

cat("Enter the elements for the second matrix:\n")
matrix2 <- matrix(nrow = rows, ncol = cols)
for (i in 1:rows) {
  for (j in 1:cols) {
    matrix2[i, j] <- as.numeric(readline(prompt = paste("Element [", i, ",", j, "]: ")))
  }
}

# Calculate the sum of the matrices
result_matrix <- sum_of_matrices(matrix1, matrix2)

# Print the results
cat("Matrix 1:\n")
```

```
print(matrix1)
cat("Matrix 2:\n")
print(matrix2)
cat("Sum of the two matrices:\n")
print(result_matrix)
```

Q2. Consider following dataset
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No'].
Use Naïve Bayes algorithm to predict [0: Overcast, 2: Mild]tuple belongs to which class whether to play the sports or not.

====>

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder

# Given data
weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy',
           'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast',
           'Overcast', 'Rainy']
temp = ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
        'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild',
        'Hot', 'Mild']
play = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No',
        'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes',
        'Yes', 'No']

# Create a DataFrame
data = pd.DataFrame({
    'Weather': weather,
    'Temperature': temp,
    'Play': play
})

# Encode the categorical data
label_encoder = LabelEncoder()
data['Weather'] = label_encoder.fit_transform(data['Weather'])
data['Temperature'] = label_encoder.fit_transform(data['Temperature'])
data['Play'] = label_encoder.fit_transform(data['Play'])

# Prepare features and target variable
X = data[['Weather', 'Temperature']]
y = data['Play']

# Create and train the Naive Bayes classifier
model = GaussianNB()
model.fit(X, y)

# New data point to predict: [0: Overcast, 2: Mild]
new_data = np.array([[label_encoder.transform(['Overcast'])[0], label_encoder.transform(['Mild'])[0]]])
```

```
# Make a prediction
prediction = model.predict(new_data)

# Decode the predicted value back to original label
predicted_play = label_encoder.inverse_transform(prediction)

# Output the prediction
print(f"The prediction for the input [Overcast, Mild] is: {predicted_play[0]}")
```

///SLIP NO:-7///

Q1. Write a R program to create a sequence of numbers from 20 to 50 and findthe mean of numbers from 20 to 60 and sum of numbers from 51 to 91.

====>

```
# Create a sequence of numbers from 20 to 50
sequence_20_to_50 <- 20:50

# Calculate the mean of numbers from 20 to 60
mean_20_to_60 <- mean(20:60)

# Calculate the sum of numbers from 51 to 91
sum_51_to_91 <- sum(51:91)

# Print the results
cat("Sequence from 20 to 50:", sequence_20_to_50, "\n")
cat("Mean of numbers from 20 to 60:", mean_20_to_60, "\n")
cat("Sum of numbers from 51 to 91:", sum_51_to_91, "\n")
```

O/P:-

Sequence from 20 to 50: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Mean of numbers from 20 to 60: 40

Sum of numbers from 51 to 91: 2911

Q2. Consider the following observations/data. And apply simple linear regression and find out estimated coefficients b1 and b1 Also analyse theperformance of the model

(Use sklearn package)

x = np.array([1,2,3,4,5,6,7,8])

y = np.array([7,14,15,18,19,21,26,23])

===>

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Given data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1) # Reshape for sklearn
y = np.array([7, 14, 15, 18, 19, 21, 26, 23])
# Create a Linear Regression model
model = LinearRegression()
# Fit the model
model.fit(x, y)
# Estimated coefficients
b0 = model.intercept_ # Intercept
b1 = model.coef_[0] # Slope
# Make predictions
y_pred = model.predict(x)
```

```
# Performance evaluation
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
# Print coefficients and performance metrics
print(f"Estimated coefficients:")
print(f"b0 (intercept): {b0:.2f}")
print(f"b1 (slope): {b1:.2f}")
print(f"\nPerformance metrics:")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R2 Score: {r2:.2f}")
# Plot the results
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label='Regression line')
plt.title('Simple Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

///SLIP NO:-8//

Q1. Write a R program to get the first 10 Fibonacci numbers.

====>

```
# Function to generate the first n Fibonacci numbers
fibonacci_numbers <- function(n) {
  fib_sequence <- numeric(n)  # Initialize a numeric vector to store Fibonacci numbers
  fib_sequence[1] <- 0        # First Fibonacci number
  fib_sequence[2] <- 1        # Second Fibonacci number

  # Generate Fibonacci numbers
  for (i in 3:n) {
    fib_sequence[i] <- fib_sequence[i - 1] + fib_sequence[i - 2]
  }

  return(fib_sequence)
}

# Get the first 10 Fibonacci numbers
n <- 10
fibonacci_sequence <- fibonacci_numbers(n)

# Print the result
cat("The first", n, "Fibonacci numbers are:", fibonacci_sequence, "\n")
```

O/P:-
The first 10 Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34

Q2. Write a python program to implement k-means algorithm to build prediction model (Use Credit Card Dataset CC GENERAL.csv Download from kaggle.com)

====>

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

```
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load the dataset
data = pd.read_csv('CC GENERAL.csv')

# Display the first few rows of the dataset
print(data.head())

# Check for missing values
print(data.isnull().sum())

# Drop any rows with missing values
data.dropna(inplace=True)

# Select relevant features for clustering (exclude 'CUST_ID')
features = data.drop(columns=['CUST_ID'])

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Determine the optimal number of clusters using the elbow method
inertia = []
K = range(1, 11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(10, 6))
plt.plot(K, inertia, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid()
plt.show()

# Based on the elbow plot, choose the optimal number of clusters
optimal_k = 4  # You can adjust this based on the elbow plot

# Apply K-means with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(scaled_features)

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

# Display the first few rows with cluster labels
print(data.head())

# Visualize the clusters
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(x=data['LIMIT_BAL'], y=data['AGE'], hue=data['Cluster'], palette='Set1', s=100)
plt.title('K-means Clustering of Credit Card Data')
plt.xlabel('Limit Balance')
plt.ylabel('Age')
plt.legend(title='Cluster')
plt.show()
```

//SLIP NO:-9///

Q1. Write an R program to create a Data frames which contain details of 5 employees and display summary of the data.

====>

```
# Create a data frame for employee details
employee_data <- data.frame(
  EmployeeID = 1:5,
  Name = c("Alice", "Bob", "Charlie", "David", "Eva"),
  Age = c(28, 34, 29, 42, 35),
  Department = c("HR", "IT", "Finance", "Marketing", "IT"),
  Salary = c(60000, 75000, 50000, 70000, 72000),
  stringsAsFactors = FALSE  # Prevents automatic conversion of strings to factors
)
# Display the data frame
print("Employee Data:")
print(employee_data)
# Display a summary of the data
summary(employee_data)
```

O/P:-

```
Employee Data:
  EmployeeID    Name   Age    Department   Salary
1         1       BAlice  28         HR        60000
2         2         Bob   34         IT         75000
3         3      Charlie  29      Finance      50000
4         4       David    42    Marketing    70000
5         5        Eva     35        IT         72000

     EmployeeID        Age           Salary
Min.          :1.0   Min.   :28.0   Min.   :50000
1st Qu.:2.0   1st Qu.:29.0   1st Qu.:60000
Median :3.0   Median :34.0   Median :72000
Mean   :3.0   Mean   :33.6   Mean   :66800
3rd Qu.:4.0   3rd Qu.:35.0   3rd Qu.:73500
Max.          :5.0   Max.   :42.0   Max.   :75000
```

Q2. Write a Python program to build an SVM model to Cancer dataset. The dataset is available in the scikit-learn library. Check the accuracyof model with precision and recall.

====>

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report
```

```
# Load the breast cancer dataset from sklearn
cancer_data = datasets.load_breast_cancer()

# Create a DataFrame
df = pd.DataFrame(data=cancer_data.data, columns=cancer_data.feature_names)
df['target'] = cancer_data.target

# Display the first few rows of the dataset
print(df.head())

# Split the data into features and target
X = df.drop('target', axis=1)
y = df['target']

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

# Create an SVM model
model = SVC(kernel='linear', random_state=42)

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")

# Print the classification report for detailed metrics
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

//SLIP NO:-11//

Q1. Write a R program to find all elements of a given list that are not inanother given list.

A=B list("x", "y", "z")
    = list("X", "Y", "Z", "x", "y", "z")

======>

```
# Define the first list
list1 <- list("x", "y", "z")

# Define the second list
list2 <- list("X", "Y", "Z", "x", "y", "z")

# Find elements in list1 that are not in list2
# Use sapply to check membership and filter elements
not_in_list2 <- list1[!sapply(list1, function(x) x %in% unlist(list2))]
```

# Display the results
cat("Elements in list1 that are not in list2:", unlist(not_in_list2), "\n")

O/P:-
Elements in list1 that are not in list2:

Q2. Write a python program to implement hierarchical clustering algorithm.(Download Wholesale customers data dataset from github.com).
======>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Load the dataset
url = 'https://raw.githubusercontent.com/plotly/datasets/master/wholesale_customers_data.csv'
data = pd.read_csv(url)

# Display the first few rows of the dataset
print(data.head())

# Preprocess the data
# Standardize the features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Perform hierarchical clustering
linked = linkage(scaled_data, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# Choose a threshold to cut the dendrogram for clusters
threshold = 10  # You can adjust this based on the dendrogram
clusters = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
labels = clusters.fit_predict(scaled_data)

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

# Display the data with cluster labels
print(data.head())

# Visualize clusters (for the first two features for simplicity)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data['Fresh'], y=data['Milk'], hue=data['Cluster'], palette='Set1', s=100)
```

```
plt.title('Hierarchical Clustering Results')
plt.xlabel('Fresh')
plt.ylabel('Milk')
plt.legend(title='Cluster')
plt.show()
```

//SLIP NO:-12//

Q1. Write a R program to create a Dataframes which contain details of 5employees and display the details.
Employee contain (empno,empname,gender,age,designation)
======>

```
# Create a DataFrame for employee details
employees <- data.frame(
  empno = c(1, 2, 3, 4, 5),
  empname = c("Alice", "Bob", "Charlie", "David", "Eva"),
  gender = c("Female", "Male", "Male", "Male", "Female"),
  age = c(25, 30, 28, 35, 22),
  designation = c("Manager", "Developer", "Designer", "Analyst", "Intern")
)
# Display the DataFrame
print(employees)
```

O/P:-

empno	empname	gender	age	designation
1	Alice	Female	25	Manager
2	Bob	Male	30	Developer
3	Charlie	Male	28	Designer
4	David	Male	35	Analyst
5	Eva	Female	22	Intern

Q2. Write a python program to implement multiple Linear Regression modelfor a car dataset.
Dataset can be downloaded from:
https://www.w3schools.com/python/python_ml_multiple_regression.asp
======>

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Load the dataset
url = "https://www.w3schools.com/python/pandas/data/car.csv"
data = pd.read_csv(url)

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

# Prepare the data for multiple linear regression
# Let's assume we're predicting 'Price' based on other features
# Convert categorical data to numerical if needed (e.g., using one-hot encoding)
data = pd.get_dummies(data, drop_first=True)

# Define the features (X) and the target variable (y)
X = data.drop('Price', axis=1)  # Features
```

```
y = data['Price']               # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Fit the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Absolute Error:", metrics.mean_absolute_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)))
```

//SLIP NO:-14//

Q1. Write a script in R to create a list of employees (name) and perform thefollowing:

a. Display names of employees in the list.

b. Add an employee at the end of the list

c. Remove the third element of the list.

======>

```
# Create a list of employee names
employees <- list("Alice", "Bob", "Charlie", "David", "Eva")

# a. Display names of employees in the list
cat("Employees in the list:\n")
print(employees)

# b. Add an employee at the end of the list
employees <- c(employees, "Frank")
cat("\nAfter adding an employee at the end:\n")
print(employees)

# c. Remove the third element of the list
employees <- employees[-3]
cat("\nAfter removing the third element:\n")
print(employees)
```

```
O/P:-Employees in the list:
[[1]]
[1] "Alice"

[[2]]
[1] "Bob"

[[3]]
[1] "Charlie"
```

[[4]]
[1] "David"

[[5]]
[1] "Eva"

After adding an employee at the end:
[[1]]
[1] "Alice"

[[2]]
[1] "Bob"

[[3]]
[1] "Charlie"

[[4]]
[1] "David"

[[5]]
[1] "Eva"

[[6]]
[1] "Frank"

After removing the third element:
[[1]]
[1] "Alice"

[[2]]
[1] "Bob"

[[3]]
[1] "David"

[[4]]
[1] "Eva"

[[5]]
[1] "Frank"

Q2. Write a Python Programme to apply Apriori algorithm on Groceries dataset. Dataset can be downloaded from (https://github.com/amankharwal/Websitedata/blob/master/Groceries_dataset.csv).
Also display support and confidence for each rule.

======>
pip install pandas mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

Load the dataset

```
url = "https://github.com/amankharwal/Websitedata/raw/master/Groceries_dataset.csv"
data = pd.read_csv(url)

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

# Convert the data into a basket format
# Create a basket for each customer
basket = (data
          .groupby(['Customer', 'Item'])['Item']
          .count().unstack().reset_index().fillna(0)
          .set_index('Customer'))

# Convert the values to 1s and 0s (1 for purchased, 0 for not purchased)
basket = basket.applymap(lambda x: 1 if x > 0 else 0)

# Display the basket format
print("\nBasket format:")
print(basket.head())

# Apply the Apriori algorithm
frequent_itemsets = apriori(basket, min_support=0.01, use_colnames=True)

# Display frequent itemsets
print("\nFrequent itemsets:")
print(frequent_itemsets)

# Generate the association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

# Display support and confidence for each rule
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence']])
```

//SLIP NO:-16//

Q1. Write a R program to create a simple bar plot of given data Year(2001, 2002, 2003) Export(25, 32,35) Import(35, 40, 50)

======>

```
years <- c(2001, 2002, 2003)
exports <- c(25, 32, 35)
imports <- c(35, 40, 50)

# Combine data into a data frame
data <- data.frame(years, exports, imports)

# Set up the bar plot
barplot(height = as.matrix(data[, -1]),
        beside = TRUE,
        names.arg = data$years,
        col = c("blue", "red"),
        legend = c("Exports", "Imports"),
        main = "Exports and Imports (2001-2003)",
        xlab = "Year",
```

```
        ylab = "Value")

# Add grid lines for better readability
grid()
```

Q2. Write a Python program build Decision Tree Classifier using Scikit-learnpackage for diabetes data set (download database from https://www.kaggle.com/uciml/pima-indians-diabetes-database)

======>

```
pip install pandas scikit-learn numpy
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv'
column_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
                'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
data = pd.read_csv(url, header=None, names=column_names)

# Display the first few rows of the dataset
print(data.head())

# Split the data into features and target variable
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']               # Target variable

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

# Create the Decision Tree Classifier
model = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print results
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(class_report)
```

//SLIP NO:-17///

Q1. Write a R program to get the first 20 Fibonacci numbers.# Function to generate Fibonacci numbers

======>
```
fibonacci <- function(n) {
  fib_seq <- numeric(n)  # Initialize a numeric vector of size n
  fib_seq[1] <- 0        # First Fibonacci number
  fib_seq[2] <- 1        # Second Fibonacci number

  for (i in 3:n) {
    fib_seq[i] <- fib_seq[i - 1] + fib_seq[i - 2]  # Calculate the next Fibonacci number
  }

  return(fib_seq)
}

# Get the first 20 Fibonacci numbers
first_20_fibonacci <- fibonacci(20)

# Print the result
print(first_20_fibonacci)
```

O/P:-
```
[1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377
[16]   610  987 1597 2584 4181
```

Q2. Write a python programme to implement multiple linear regression modelfor stock market data frame as follows:
Stock_Market = {'Year':
[2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2016,2
016,20,16,2016,2016,2016,2016,2016,2016,2016,2016,2016],
'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1],
'Interest_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1
.75,1.75,1.75,1.75,1.75,1.75],
'Unemployment_Rate':
[5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5
.9,6.2,6.2,6.1],
'Stock_Index_Price': [1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,
965,943,958,971,949,884,866,876,822,704,719] }
And draw a graph of stock market price verses interest rate.
======>
```
pip install pandas numpy scikit-learn matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Create the DataFrame
Stock_Market = {
    'Year': [2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017,
             2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016],
    'Month': [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
              12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1],
    'Interest_Rate': [2.75, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.25, 2.25, 2.25, 2, 2,
                      1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75],
    'Unemployment_Rate': [5.3, 5.3, 5.3, 5.3, 5.4, 5.6, 5.5, 5.5, 5.5, 5.6, 5.7, 5.9,
                          6, 5.9, 5.8, 6.1, 6.2, 6.1, 6.1, 6.1, 5.9, 6.2, 6.2, 6.1],
```

```
    'Stock_Index_Price': [1464, 1394, 1357, 1293, 1256, 1254, 1234, 1195, 1159, 1167,
                          1130, 1075, 1047, 965, 943, 958, 971, 949, 884, 866, 876, 822, 704, 719]
}

df = pd.DataFrame(Stock_Market)

# Define features and target variable
X = df[['Year', 'Month', 'Interest_Rate', 'Unemployment_Rate']]
y = df['Stock_Index_Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and fit the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print the coefficients
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Plotting Stock Index Price vs Interest Rate
plt.figure(figsize=(10, 6))
plt.scatter(df['Interest_Rate'], df['Stock_Index_Price'], color='blue', label='Data Points')
plt.title('Stock Index Price vs Interest Rate')
plt.xlabel('Interest Rate (%)')
plt.ylabel('Stock Index Price')
plt.grid()
plt.legend()
plt.show()
```

//SLIP NO:-18//

Q1. Write a R program to find the maximum and the minimum value of a givenvector

======>

```
# Create a vector of marks
Marks <- c(85, 90, 78, 92, 88, 76, 95, 89, 84)

# Find the maximum value
max_value <- max(Marks)

# Find the minimum value
min_value <- min(Marks)

# Print the results
cat("Maximum Marks:", max_value, "\n")
cat("Minimum Marks:", min_value, "\n")
```

O/P:-

Maximum Marks: 95

Minimum Marks: 76

Q2. Consider the following observations/data. And apply simple linear regression and find out

estimated coefficients b1 and b1 Also analyse theperformance of the model
(Use sklearn package)
x = np.array([1,2,3,4,5,6,7,8])
y = np.array([7,14,15,18,19,21,26,23])

======>

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1)  # Reshape for sklearn
y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

# Create a linear regression model
model = LinearRegression()

# Fit the model to the data
model.fit(x, y)

# Get the estimated coefficients
b1 = model.coef_[0]  # Slope
b0 = model.intercept_  # Intercept

# Make predictions
y_pred = model.predict(x)

# Calculate performance metrics
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# Print coefficients and performance metrics
print(f'Estimated coefficients: b0 (intercept) = {b0:.2f}, b1 (slope) = {b1:.2f}')
print(f'Mean Squared Error: {mse:.2f}')
print(f'R² Score: {r2:.2f}')

# Plotting the results
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, y_pred, color='red', label='Regression Line')
plt.title('Simple Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid()
plt.show()
```

//SLIP NO:-19//

Q1. Write a R program to create a Dataframes which contain details of 5 Studentsand display the details.
Students contain (Rollno,Studname,Address,Marks)
===========>

```
# Create a DataFrame for students
students <- data.frame(
  Rollno = c(101, 102, 103, 104, 105),
```

```
  Studname = c("Alice", "Bob", "Charlie", "David", "Eve"),
  Address = c("123 Main St", "456 Maple Ave", "789 Oak Dr", "321 Pine St", "654 Birch Ln"),
  Marks = c(85, 92, 78, 88, 95)
)

# Display the DataFrame
print(students)
```
O/P:-

Rollno	Studname	Address	Marks
101	Alice	123 Main St	85
102	Bob	456 Maple Ave	92
103	Charlie	789 Oak Dr	78
104	David	321 Pine St	88
105	Eve	654 Birch Ln	95

Q2. Write a python program to implement multiple Linear Regression modelfor a car dataset.
Dataset can be downloaded from:
https://www.w3schools.com/python/python_ml_multiple_regression.asp
===========>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset from the URL
url = 'https://www.w3schools.com/python/data/carprices.csv'
data = pd.read_csv(url)

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

# Define features (independent variables) and target variable (dependent variable)
X = data[['age', 'mileage', 'tax']]
y = data['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a multiple linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
# Print coefficients and performance metrics
print(f'Coefficients: {model.coef_}')
print(f'Intercept: {model.intercept_}')
print(f'Mean Squared Error: {mse:.2f}')
print(f'R² Score: {r2:.2f}')

# Plotting the results (for visual inspection, only showing actual vs predicted prices)
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lw=2)
plt.title('Actual vs Predicted Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.grid()
plt.show()
```

//SLIP NO:-20//

Q1. Write a R program to create a data frame from four given vectors.

===========>

```
# Create vectors
names <- c("Alice", "Bob", "Charlie", "David")
ages <- c(25, 30, 35, 40)
heights <- c(5.5, 6.0, 5.8, 5.7)  # Heights in feet
weights <- c(130, 180, 160, 170)   # Weights in pounds

# Create a data frame from the vectors
students_df <- data.frame(
  Name = names,
  Age = ages,
  Height = heights,
  Weight = weights
)

# Display the data frame
print(students_df)
```

Q2. Write a python program to implement hierarchical Agglomerativeclustering algorithm. (Download Customer.csv dataset from github.com).

===========>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage

# Load the dataset from the GitHub URL
url = 'https://raw.githubusercontent.com/your-repo/Customer.csv'  # Replace with actual URL if needed
data = pd.read_csv(url)

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

```
# Select features for clustering (modify based on your dataset)
# For this example, let's assume there are columns 'Age' and 'SpendingScore'
features = data[['Age', 'SpendingScore']]

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Perform hierarchical agglomerative clustering
model = AgglomerativeClustering(n_clusters=4)  # Choose the number of clusters
labels = model.fit_predict(scaled_features)

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

# Display the clustered data
print("Clustered Data:")
print(data.head())

# Create a dendrogram
plt.figure(figsize=(10, 7))
linked = linkage(scaled_features, method='ward')
dendrogram(linked, orientation='top', labels=data.index, distance_sort='descending',
show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()

# Plot the clusters
plt.figure(figsize=(10, 7))
plt.scatter(data['Age'], data['SpendingScore'], c=data['Cluster'], cmap='rainbow')
plt.title('Hierarchical Agglomerative Clustering')
plt.xlabel('Age')
plt.ylabel('Spending Score')
plt.show()
```

///------BEST OF
LUCK------///