** Anaconda Prompt Commands for package installation :-

```
pip install apyori
pip install graphviz
pip install mlxtend
conda install -c conda-forge mlxtend
pip install pydotplus
pip install --upgrade scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import train_test_split
```

SLIP 1)

Q.1) Write a R program to add, multiply and divide two vectors of integertype. (Vector length should be minimum 4)

->

```
vector1 = seq(10, 40 , length.out = 4)
vector2 = c(20, 10, 40, 40)
print("Original Vectors : ")
print(vector1)
print(vector2)
add = vector1 + vector2
print("Addition of vectors : ")
print(add)
sub = vector1 - vector2
print("Substraction of vectors : ")
print(sub)
mul = vector1 * vector2
print("Multiplication of vectors : ")
print(mul)
div = vector1 / vector2
print("Division of vectors : ")
print(div)
```

Q.2) Consider the student data set. It can be downloaded from: https://drive.google.com/open?id=1oakZCv7g3mlmCSdv9J8kdSaqO5_6dIOw . Write a programme in python to apply simple linear regression and find out mean absolute error, mean squared error and root mean squared error.

->

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error
# The dataset should have two columns: 'Hours' and 'Scores'
data = pd.read_csv('s1_student_data.csv')
# Split the dataset into features (X) and target variable (y)
X = data['Hours'].values.reshape(-1, 1)
y = data['Scores'].values
# Split 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)
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f'Mean Absolute Error : {mae: .2f}')
print(f'Mean Squared Error : {mse: .2f}')
print(f'Root Mean Squared Error : {rmse: .2f}')
```

SLIP 2)

Q.1) Write an R program to calculate the multiplication table using a function.

->

```
number <- as.integer(readline(prompt = "Please Enter a Number for
Table : "))
Disp_table = function(number)
{
for( t in 1:10)
{
```

```
print( paste( number, '*', t, '=', number * t))
}
}
Disp_table(number)
```

Q.2) Write a python program to implement k-means algorithms on a synthetic dataset.
->

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
data = make_blobs(n_samples = 300, n_features = 2, centers = 5,
cluster_std = 1.8, random_state = 101)
data[0].shape
data[1]
plt.scatter(data[0][:,0], data[0][:,1], c = data[1], cmap = 'brg')
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 5)
kmeans.fit(data[0])
kmeans.cluster_centers_
kmeans.labels_f, (ax1, ax2) = plt.subplots(1, 2, sharey = True,
figsize = (10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0], data[0][:,1], c = kmeans.labels_, cmap =
'brg')
ax2.set_title("Original")
ax2.scatter(data[0][:,0], data[0][:,1], c = data[1], cmap = 'brg')
```

SLIP 3)

Q.1) Write a R program to reverse a number and also calculate the sum of digits of that number.
->

```
Reverse_Sum = function(n)
{
sum = 0
rev = 0
while(n > 0)
{
r = n %% 10
sum = sum + r
```

```
rev = rev * 10 + r
n = n %/% 10
}
print(paste("Sum of digit : ", sum))
print(paste("Reverse of number : ", rev))
}
n = as.integer(readline(prompt = "Enter a number :"))
Reverse_Sum(n)
```

Q.2) 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
import matplotlib.pyplot as plt
def estimate_coef(x, y):
 # number of observations/points
 n = np.size(x)
 # mean of x and y vector
 m_x = np.mean(x)
 m_y = np.mean(y)
 # calculating cross-deviation and deviation about x
 SS_xy = np.sum(y * x) - n * m_y * m_x
 SS_xx = np.sum(x * x) - n* m_x * m_x
 # calculating regression coefficients
 b_1 = SS_xy / SS_xx
 b_0 = m_y - b_1 * m_x
 return (b_0, b_1)
def plot_regression_line(x, y, b):
 # plotting the actual points as scatter plot
 plt.scatter(x, y, color = "m", marker = "o", s = 30)
 # predicted response vector
 y_pred = b[0] + b[1] * x
 # plotting the regression line
 plt.plot(x, y_pred, color = "g")
 # putting labels
 plt.xlabel('x')
 plt.ylabel('y')
 # function to show plot
 plt.show()
def main():
 # observations / 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])
 # estimating coefficients
 b = estimate_coef(x, y)
```

```
 print("Estimated coefficients :\nb_0 = {} \nb_1 =
{}".format(b[0], b[1]))
 # plotting regression line
 plot_regression_line(x, y, b)
if __name__ == "__main__":
 main()
```

OR

```
import numpy as np# 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])
# Calculate the mean of x and y
mean_x = np.mean(x)
mean_y = np.mean(y)
# Calculate the differences between each data point and the mean
x_diff = x - mean_x
y_diff = y - mean_y
# Calculate b1 (slope) by taking the dot product of x_diff and
y_diff divided by the dot product of x_diff with itself
b1 = np.sum(x_diff * y_diff) / np.sum(x_diff ** 2)
# Calculate b0 (intercept) using the formula b0 = mean(y) - b1 *
mean(x)
b0 = mean_y - b1 * mean_x
# Print the estimated coefficients
print("Estimated Coefficient b0 (Intercept) : ", b0)
print("Estimated Coefficient b1 (Slope) : ", b1)
```

SLIP 4)

Q.1) Write a R program to calculate the sum of two matrices of given size.

->

```
A = matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
B = matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)
num_of_rows = nrow(A)
num_of_cols = ncol(A)
add = matrix( , nrow = num_of_rows, ncol = num_of_cols)
print(A)
print(B)
for(row in 1:num_of_rows)
{
for(col in 1:num_of_cols)
```

```
{
add[row, col] <- A[row, col] + B[row, col]
}
}
print(add)
```

Q.2) 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 Naive Bayes algorithm to predict [0: Overcast, 2: Mild] tuple belongs to which class whether to play the sports or not.

->

```
from sklearn.naive_bayes import MultinomialNB
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']
# Encode categorical variables
le_weather = LabelEncoder()
le_temp = LabelEncoder()
le_play = LabelEncoder()
weather_encoded = le_weather.fit_transform(weather)
temp_encoded = le_temp.fit_transform(temp)
play_encoded = le_play.fit_transform(play)
# Combine the encoded features into a feature matrix
X = list(zip(weather_encoded, temp_encoded))
# Create and train the Naïve Bayes model
model = MultinomialNB()
model.fit(X, play_encoded)
# New tuple to predict [0: Overcast, 2: Mild]
new_data = [(le_weather.transform(['Overcast'])[0],
le_temp.transform(['Mild'])[0])]
# Predict using the model
prediction = model.predict(new_data)
# Decode the prediction
predicted_class = le_play.inverse_transform(prediction)
# Print the prediction
print("Prediction : Whether to play sports or not ->  ",
predicted_class[0])
```

SLIP 5)

Q.1) Write a R program to concatenate two given factors.
->
```
fac1 <- factor(letters[1:3])
print("Factor1 : ")
print(fac1)
sapply(fac1, class)
fac2 <- factor(c(1:4))
print("Factor2 : ")
print(fac2)
sapply(fac2, class)
level1 <- levels(fac1)[fac1]
level2 <- levels(fac2)[fac2]
combined <- factor(c( level1, level2))
print("Combined Factor : ")
print(combined)
sapply(combined, class)
```

Q.2) Write a Python program build Decision Tree Classifier using Scikit- learn package for diabetes data set (download database from https://www.kaggle.com/uciml/pima-indians-diabetes-database)
->
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
pima = pd.read_csv('s5_diabetes.csv')
pima.head()
import seaborn as sns
corr = pima.corr()
ax = sns.heatmap(corr, vmin = -1, vmax = 1, center = 0, cmap =
sns.diverging_palette(20, 220, n = 200), square = True)
ax.set_xticklabels( ax.get_xticklabels(), rotation = 45,
horizontalalignment = 'right');
# feature selection
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']
```

```
x = pima[feature_cols]
y = pima.Outcome
# split data
X_train, X_test, Y_train, Y_test = train_test_split(x, y,
test_size = 0.3, random_state = 1)
# build model
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train, Y_train)
# predict
y_pred = classifier.predict(X_test)
print(y_pred)
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, y_pred)
print(confusion_matrix(Y_test, y_pred))
# accuracy
print("Accuracy : ", metrics.accuracy_score(Y_test, y_pred))
```

SLIP 6)

Q.1) Write a R programto create a data frame using two given vectors and display the duplicate elements.

->

```
companies <- data.frame(Shares = c( "TCS", "Reliance", "HDFC
Bank", "Infosys", "Reliance"), Price = c( 3200, 1900, 1500, 2200,
1900))
companies
print("After removing Duplicates ")
companies[ duplicated( companies),]
```

Q.2) Write a python program to implement hierarchical Agglomerative clustering algorithm. (Download Customer.csv dataset from github.com).

->

```
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('s6_mall_customers.csv')
X = dataset.iloc[:, [3, 4]].values
print(X)
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
```

```
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity =
'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

SLIP 7)

Q.1) Write a R program to create a sequence of numbers from 20 to 50 and find the mean of numbers from 20 to 60 and sum of numbers from 51 to 91.

->

```
print("Sequence of numbers from 20 to 50 : ")
print(seq(20,50))
print("Mean of numbers from 20 to 60 : ")
print(mean(20:60))
print("Sum of numbers from 51 to 91 : ")
print(sum(51:91))
```

Q.2) Consider the following observations/data and apply simple linear regression and find out estimated coefficients b1 and b1 Also analyse the performance 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 scipy import stats
x = np.array([ 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([ 7, 14, 15, 18, 19, 21, 26, 23])
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
 return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

SLIP 8)

Q.1) Write a R program to get the first 10 Fibonacci numbers.
->

```
Fibonacci <- numeric(10)
Fibonacci[1] <- Fibonacci[2] <- 1
for (i in 3:10) Fibonacci[i] <- Fibonacci[i - 2] + Fibonacci[i -
1]
print("First 10 Fibonacci numbers : ")
print(Fibonacci)
```

OR

```
print_fibonacci <- function(n)
{
a <- 1
b <- 2
print("Fibonacci Sequence : ")
print(a)
for (i in 1:n)
{
print(a)
next_num <- a + b
a <- b
b <- next_num
```

```
}
}
number_of_terms <- 10
print_fibonacci(number_of_terms)
```

Q.2) 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 matplotlib.pyplot as mtp
import pandas as pd
dataset = pd.read_csv('s8_credit_card.csv')
x = dataset.iloc[:, [3, 4]].values
print(x)
from sklearn.cluster import KMeans
wcss_list= []
for i in range(1, 11):
 kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state
= 42)
 kmeans.fit(x)
 wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state =
42)
y_predict = kmeans.fit_predict(x)
mtp.scatter(x [y_predict == 0, 0], x [y_predict == 0, 1], s = 100,
c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x [y_predict == 1, 0], x [y_predict == 1, 1], s = 100,
c = 'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x [y_predict== 2, 0], x [y_predict == 2, 1], s = 100,
c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label =
'Centroid')
mtp.title('Clusters of Credit Card')
mtp.xlabel('V3')
mtp.ylabel('V4')
mtp.legend()
mtp.show()
```

OR

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Load the dataset
data = pd.read_csv('s8_credit_card.csv')
# Fill missing values with mean of the respective columns
data.fillna(data.mean(), inplace = True)
# Select relevant features for clustering
X = data.iloc[:, 1:].values  # Exclude the 'CUST_ID' column for
clustering
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Apply K-means algorithm
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state =
42)
kmeans.fit(X_scaled)
# Add cluster labels to the original dataset
data['Cluster'] = kmeans.labels_
# Print the count of customers in each cluster
print(data['Cluster'].value_counts())
# Visualization (considering only 2 dimensions for plotting)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c = kmeans.labels_,
cmap = 'viridis', marker = '.')
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s = 300, c = 'red', label =
'Centroids')
plt.title('K-means Clustering')
plt.legend()
plt.show()
```

SLIP 9)

Q.1) Write an R program to create a Data frames which contain details of 5 employees and display summary of the data.

->

```
emp.data<- data.frame(
employee_id = c (101:105),
employee_name = c( "Ram","Sham","Neha","Siya","Sumit"),
salary = c( 40000, 35000, 20000, 25000, 30000),
starting_date = as.Date(c( "2020-01-01", "2019-09-01", "2021-01-
01", "2019-05-01", "2020-03-05")),
stringsAsFactors = FALSE
)
```

```
print(emp.data)
```

Q.2) Write a Python program to build an SVM model to Cancer dataset. The dataset is available in the scikit-learn library. Check the accuracy of model with precision and recall.
->

```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
print("Features : ", cancer.feature_names)
print("Labels : ", cancer.target_names)
cancer.data.shape
# print the cancer data features (top 5 records)
print(cancer.data[0:5])
# print the cancer labels (0 : malignant, 1 : benign)
print(cancer.target)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(cancer.data,
cancer.target, test_size = 0.3, random_state = 109) # 70% training
and 30% test
#Import svm model
from sklearn import svm
#Create a svm Classifier
clf = svm.SVC(kernel = 'linear') #Linear Kernel
#Train the model using the training sets
clf.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
from sklearn import metrics
print("Accuracy : ", metrics.accuracy_score(y_test, y_pred))
print("Precision : ", metrics.precision_score(y_test, y_pred))
print("Recall : ", metrics.recall_score(y_test, y_pred))
```

SLIP 10)

Q.1) Write a R program to find the maximum and the minimum value of a given vector.
->

```
my_vector <- c(3, 9, 1, 5, 7, 2, 8, 4, 6)
max_value <- max(my_vector)
print("Maximum value : ")
print(max_value)
```

```
min_value <- min(my_vector)
print("Minimum value : ")
print(min_value)
```

Q.2) Write a Python Programme to read the dataset ("Iris.csv"). Download dataset from https://archive.ics.uci.edu/ml/datasets/iris and apply Apriori algorithm.

->

```
import pandas as pd
from apyori import apriori
# Read the Iris dataset
data = pd.read_csv('s10_iris.csv')
# Discretize the numerical attributes for Apriori analysis (for
demonstration purposes)
data['sepal_length'] = pd.cut(data['sepal_length'], bins = 3,
labels = ['Short', 'Medium', 'Long'])
data['sepal_width'] = pd.cut(data['sepal_width'], bins = 3, labels
= ['Narrow', 'Medium', 'Wide'])
data['petal_length'] = pd.cut(data['petal_length'], bins = 3,
labels = ['Short', 'Medium', 'Long'])
data['petal_width'] = pd.cut(data['petal_width'], bins = 3, labels
= ['Narrow', 'Medium', 'Wide'])
# Select the columns for Apriori analysis
data_apriori = data[['sepal_length', 'sepal_width',
'petal_length', 'petal_width']]
# Convert dataframe to list of lists (transactions format for
Apriori)
transactions = data_apriori.values.tolist()
# Apply Apriori algorithm
association_rules = apriori(transactions, min_support = 0.1,
min_confidence = 0.7)
association_results = list(association_rules)
# Print association rules
for item in association_results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule : " + " , ".join(items))
    print("Support : " + str(item[1]))
    print("Confidence : " + str(item[2][0][2]))
    print("=====================================")
```

SLIP 11)

Q.1) Write a R program to find all elements of a given list that are not in another given list. list("x", "y", "z") ; list("X", "Y", "Z", "x", "y", "z")

->

```
l1 = list("x", "y", "z")
l2 = list("X", "Y", "Z", "x", "y", "z")
print("Original lists : ")
print(l1)
print(l2)
print("All elements of l2 that are not in l1 : ")
setdiff(l2, l1)
```

Q.2) Write a python program to implement hierarchical clustering algorithm. (Download Wholesale customers data dataset from github.com).

->

```
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('s11_wholesale_customers.csv')
X = dataset.iloc[:, [3, 4]].values
print(X)
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity =
'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
```

```
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

SLIP 12)

Q.1) Write a R program to create a Dataframes which contain details of 5 employees and display the details. Employee contain (empno, empname, gender, age, designation).

```
->
Employees = data.frame( Emp_No = c( 101, 102, 103, 104, 105),
Emp_Name = c( "Anastasia", "Dima", "Katherine", "James", "Laura"),
Gender = c( "M", "M", "F", "F", "M"),
Age = c( 23, 22, 25, 26, 32),
Designation = c( "Clerk", "Manager", "Exective", "CEO",
"ASSISTANT")
)
print("Details of the employees : ")
print(Employees)
```

Q.2) Write a python program to implement multiple Linear Regression model for a car dataset. Dataset can be downloaded from:
https://www.w3schools.com/python/python_ml_multiple_regression.asp

```
->
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
data = pd.read_csv('s12_cars.csv')
print(data.head())
# Selecting features and target variable
X = data[['Weight', 'Volume']]
y = data['CO2']
# Splitting 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)
# Creating a Linear Regression model
model = LinearRegression()
```

```
# Training the model
model.fit(X_train, y_train)
# Making predictions
y_pred = model.predict(X_test)
# Model evaluation
print('Coefficients : ', model.coef_)
print('Intercept : ', model.intercept_)
print('Mean Squared Error (MSE) : %.2f ' %
mean_squared_error(y_test, y_pred))
print('Coefficient of Determination (R^2) : %.2f ' %
r2_score(y_test, y_pred))
# Plotting predicted vs actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual CO2 Emissions')
plt.ylabel('Predicted CO2 Emissions')
plt.title('Actual vs Predicted CO2 Emissions')
plt.show()
```

SLIP 13)

Q.1) Draw a pie chart using R programming for the following data distribution:

Digits on Dice	1	2	3	4	5	6
Frequency of getting each number	7	2	6	3	4	8

->

```
digits <- c(7,2,6,3,4,8)
Frequency <- c(1,2,3,4,5,6)
pie(digits, Frequency)
```

OR

```
Digits <- c( 1, 2, 3, 4, 5, 6)
Frequency <- c( 7, 2, 6, 3, 4, 8)
pi <- data.frame( Digits, Frequency)
pi
pct <- round( 100 * pi $ Frequency / sum( pi $ Frequency))
pie( pi $ Frequency, labels = paste( pi $ Digits, " ->  ", pct,
"%"), col = rainbow( length( pi $ Frequency) ), main = "Frequency
of getting each number")
```

Q.2) Write a Python program to read “StudentsPerformance.csv” file. Solve ollowing:
- To display the shape of dataset.
- To display the top rows of the dataset with their columns.
Note: Download dataset from following link : https://www.kaggle.com/spscientist/students-performance-inexams?select=StudentsPerformance.csv)
->

```
import pandas as pd
data = pd.read_csv('s13_students_performance.csv')
# Display the shape of the dataset
print("Shape of the dataset : ", data.shape)
# Display the top rows of the dataset
print("Top rows of the dataset : ")
print(data.head())
```

SLIP 14)

Q.1) 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.
->

```
list_data <- list( "Ram Sharma", "Sham Varma", "Raj Jadhav", "Ved Sharma")
print(list_data)
new_Emp <- "Kavya Anjali"
list_data <- append( list_data, new_Emp)
print(list_data)
list_data[3] <- NULL
print(list_data)
```

Q.2) Write a Python Programme to apply Apriori algorithm on Groceries dataset. Dataset can be downloaded from (htps://github.com/amankharwal/Websitedata/blob/master/Groceries_dataset.csv). Also display support and confidence for each rule.
->

```
import pandas as pd
from apyori import apriori
# Read the Groceries dataset from the provided link
store_data = pd.read_csv('s14_groceries_dataset.csv', header =
None)
# Convert the dataset into a list of lists (transactions format
for Apriori)
transactions = []
for index, row in store_data.iterrows():
 transactions.append([str(item) for item in row if
pd.notnull(item)])
# Apply Apriori algorithm
association_rules = apriori(transactions, min_support=0.0045,
min_confidence=0.2, min_lift=3, max_length=None)
association_results = list(association_rules)
# Display support and confidence for each rule
for item in association_results:
 pair = item[0]
 items = [x for x in pair]
 print("Rule: " + ", ".join(items))
 print("Support: " + str(item[1]))
 print("Confidence: " + str(item[2][0][2]))
 print("====================================")
```

SLIP 15)

Q.1) Write a R program to add, multiply and divide two vectors of integer type. (vector length should be minimum 4)
->

```
vector1 = seq( 10, 40, length.out = 4)
vector2 = c( 20, 10, 40, 40)
print("Original Vectors : ")
print(vector1)
print(vector2)
add = vector1 + vector2
print("Addition of vectors : ")
print(add)
sub = vector1 - vector2
print("Substraction of vectors  : ")
print(sub)
```

```
mul = vector1 * vector2
print("Multiplication of vectors : ")
print(mul)
div = vector1 / vector2
print("Division of Vectors : ")
print(div)
```

Q.2) Write a Python program build Decision Tree Classifier for shows.csv from pandas and predict class label for show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7. Create a csv file as shown in https://www.w3schools.com/python/python_ml_decision_tree.asp

->

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
# Load the dataset
data = pd.DataFrame({'Age': [36, 42, 23, 52, 43, 44, 66, 35, 52,
35, 24, 18, 45],
    'Experience': [10, 12, 4, 4, 21, 14, 3, 14, 13, 5, 3, 3, 9],
    'Rank': [9, 4, 6, 4, 8, 5, 7, 9, 7, 9, 5, 7, 9],
    'Nationality': ['UK', 'USA', 'N', 'USA', 'USA', 'UK', 'N',
'UK', 'N', 'N', 'USA', 'UK', 'UK'],
    'Go': ['NO', 'NO', 'NO', 'NO', 'YES', 'NO', 'YES', 'YES',
'YES', 'YES', 'NO', 'YES', 'YES'] })
# Encode 'Nationality' feature
label_encoder = LabelEncoder()
data['Nationality'] =
label_encoder.fit_transform(data['Nationality'])
# Define features (X) and target (y)
X = data.drop(columns = ['Go'])
y = data['Go']
# Create a Decision Tree Classifier
clf = DecisionTreeClassifier(random_state = 42)
# Train the classifier on the entire dataset
clf.fit(X, y)
# Create a new data point for prediction
new_data = pd.DataFrame({'Age' : [40], 'Experience' : [10], 'Rank'
: [7], 'Nationality' : ['USA']})
# Encode 'Nationality' feature
new_data['Nationality'] =
label_encoder.transform(new_data['Nationality'])
# Make a prediction for the new data point
prediction = clf.predict(new_data)
print("Can the comedian go to the show : ", prediction[0])
```

OR

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
df = pandas.read_csv('s15_shows.csv')
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
features = ['Age', 'Experience', 'Rank', 'Nationality']
X = df[features]
y = df['Go']
print(X)
print(y)
```

SLIP 16)

Q.1) Write a R program to create a simple bar plot of given data

Year	Export	Import
2001	26	35
2002	32	40
2003	35	50

->

```
# Create a data frame with the given data
data <- data.frame(
  Year = c(2001, 2002, 2003),
  Export = c(26, 32, 35),
  Import = c(35, 40, 50)
)
# Plotting a bar plot
barplot(
  height = t(data[, -1]),   # Exclude 'Year' column and transpose
for barplot
  beside = TRUE,            # Place bars beside each other
  names.arg = data$Year,    # Year on x-axis
  col = c("blue", "red"),   # Colors for Export and Import bars
  xlab = "Year",            # X-axis label
  ylab = "Value",           # Y-axis label
  main = "Export and Import Data"  # Title of the plot
)
```

```
legend("topright", legend = c("Export", "Import"), fill =
c("blue", "red"))
```

OR

```
# Import lattice
library(lattice)
# Create data
gfg <- data.frame(
 x = c(26,35,32,40,35,50),
 grp = rep(c("group 1", "group 2", "group 3"),
 each = 2),
 subgroup = LETTERS[1:2]
)
# Create grouped barplot using lattice
barchart(x ~ grp, data = gfg, groups = subgroup)
```

Q.2) 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)

->

```
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
# Load the dataset
data = pd.read_csv('s16_diabetes.csv')
# Display the first few rows of the dataset
print("First few rows of the dataset : ")
print(data.head())
# Features and target variable
X = data.drop('Outcome', axis = 1)
y = data['Outcome']
# Splitting 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)
# Build Decision Tree Classifier
clf = DecisionTreeClassifier(random_state = 42)
clf.fit(X_train, y_train)
# Make predictions on the test set
y_pred = clf.predict(X_test)
# Evaluate model
print("Accuracy Score : ", accuracy_score(y_test, y_pred))
print("Classification Report : ")
print(classification_report(y_test, y_pred))
```

#OR

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
pima = pd.read_csv('s16_diabetes.csv')
pima.head()
import seaborn as sns
corr = pima.corr()
ax = sns.heatmap(corr, vmin = -1, vmax = 1, center = 0, cmap =
sns.diverging_palette(20, 220, n = 200), square = True)
ax.set_xticklabels(ax.get_xticklabels(), rotation = 45,
horizontalalignment = 'right');
# feature selection
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']
x = pima[feature_cols]
y = pima.Outcome
# split data
X_train, X_test, Y_train, Y_test = train_test_split(x, y,
test_size = 0.3, random_state = 1)
# build model
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train, Y_train)
# predict
y_pred = classifier.predict(X_test)
print(y_pred)
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, y_pred)
print(confusion_matrix(Y_test, y_pred))
# accuracy
print("Accuracy : ", metrics.accuracy_score(Y_test, y_pred))
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(classifier, out_file = dot_data, filled = True,
rounded = True, special_characters = True, feature_names =
feature_cols, class_names = ['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

SLIP 17)

Q.1) Write a R program to get the first 20 Fibonacci numbers.
->

```
Fibonacci <- numeric(20)
Fibonacci[1] <- Fibonacci[2] <- 1
for (i in 3:20) Fibonacci[i] <- Fibonacci[i - 2] + Fibonacci[i -
1]
print("First 20 Fibonacci numbers : ")
print(Fibonacci)
```

OR

```
print_fibonacci <- function(n)
{
a <- 1
b <- 2
print("Fibonacci Sequence : ")
print(a)
for (i in 1:n)
{
print(a)
next_num <- a + b
a <- b
b <- next_num
}
}
number_of_terms <- 20
print_fibonacci(number_of_terms)
```

Q.2) Write a python programme to implement multiple linear regression model for stock market data frame as follows:

```
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, 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.
->
```
import pandas as pd
import matplotlib.pyplot as plt
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, 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,columns =  ['Year', 'Month',
'Interest_Rate', 'Unemployment_Rate', 'Stock_Index_Price'])
plt.scatter(df['Interest_Rate'], df['Stock_Index_Price'], color =
'red')
plt.title('Stock Index Price Vs Interest Rate', fontsize = 14)
plt.xlabel('Interest Rate', fontsize = 14)
plt.ylabel('Stock Index Price', fontsize = 14)
plt.grid(True)
plt.show()
```

SLIP 18)

Q.1) Write a R program to find the maximum and the minimum value of a given vector.
->
```
my_vector <- c(3, 9, 1, 5, 7, 2, 8, 4, 6)
max_value <- max(my_vector)
print("Maximum value : ")
print(max_value)
min_value <- min(my_vector)
print("Minimum value : ")
print(min_value)
```

Q.2) Consider the following observations/data. And apply simple linear regression and find out estimated coefficients b1 and b1. Also analyse the performance 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 scipy import stats
x = np.array([ 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([ 7, 14, 15, 18, 19, 21, 26, 23])
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
 return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

SLIP 19)

Q.1) Write a R program to create a Dataframes which contain details of 5 Students and display the details. Students contain (Rollno , Studname, Address, Marks)
->

```
Students = data.frame( Roll_No = c( 20, 21, 22, 23, 24),
Name = c( "Kiran", "Shikha", "Kamal", "Jay", "Lokesh"),
Address = c( "Kokan", "Beed", "Kolhapur", "Mumbai", "Pune"),
Marks = c( 23, 22, 25, 26, 32)
)
print("Details of the students : ")
print(Students)
```

Q.2) Write a python program to implement multiple Linear Regression model for a car dataset. Dataset can be downloaded from:
https://www.w3schools.com/python/python_ml_multiple_regression.asp

->

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
data = pd.read_csv('s19_cars.csv')
print(data.head())
# Selecting features and target variable
X = data[['Weight', 'Volume']]
y = data['CO2']
# Splitting 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)
# Creating a Linear Regression model
model = LinearRegression()
# Training the model
model.fit(X_train, y_train)
# Making predictions
y_pred = model.predict(X_test)
# Model evaluation
print('Coefficients : ', model.coef_)
print('Intercept : ', model.intercept_)
print('Mean Squared Error (MSE) : %.2f ' %
mean_squared_error(y_test, y_pred))
print('Coefficient of Determination (R^2) : %.2f ' %
r2_score(y_test, y_pred))
# Plotting predicted vs actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual CO2 Emissions')
plt.ylabel('Predicted CO2 Emissions')
plt.title('Actual vs Predicted CO2 Emissions')
plt.show()
```

SLIP 20)

Q.1) Write a R program to create a data frame from four given vectors.

->

```
name = c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily',
'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas')
score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)
attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)
qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no',
'no', 'yes')
```

```
print("Original data frame :")
print(name)
print(score)
print(attempts)
print(qualify)
df = data.frame(name, score, attempts, qualify)
print(df)
```

Q.2) Write a python program to implement hierarchical Agglomerative clustering algorithm. (Download Customer.csv dataset from github.com).

->

```
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('s20_mall_customers.csv')
X = dataset.iloc[:, [3, 4]].values
print(X)
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity =
'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```