ML- Lab programs:

1. K-Means:

```
    def getSquaredDistance(point1, point2):
        return round(((point2[0] - point1[0])**2 + (point2[1] -
    point1[1])**2),4)

def getDistanceFromPoints(centroids, datapoints):
    distance_from_cluster = []     # [cluster1_distances,
cluster2_distances,...]
    intermediate_result = []
    for i in centroids:
        for j in datapoints:
            intermediate_result.append(getSquaredDistance(i,j))
        distance_from_cluster.append(intermediate_result)
        intermediate_result = []   # reset intermediate_result as empty
list
    return distance_from_cluster


def printResult(centroids, point_to_cluster_mapping):
    for i in range(len(centroids)):
        print("Centroid",i,centroids[i])
    for i in point_to_cluster_mapping:
        print("Point: ",i,"Cluster:",point_to_cluster_mapping[i])

def kmeansclustering(centroids, datapoints):
    '''
    Driver code for K-Means clustering
    '''
    k = len(centroids)
    distance_from_cluster = getDistanceFromPoints(centroids, datapoints) #
[cluster1_distances, cluster2_distances,...]


    # assign each datapoint to the nearest cluster

    point_to_cluster_mapping = {}   # point -> cluster
    max_valued_cluster = 0
    for i in range(len(datapoints)):
        point_to_cluster_mapping[i] = None  # initial mapping as None
        for cluster in range(len(centroids)):
            if distance_from_cluster[cluster][i] <
distance_from_cluster[max_valued_cluster][i]:
                max_valued_cluster = cluster
        point_to_cluster_mapping[i] = max_valued_cluster

    # compute new centroids by averaging with new points
```

```
    cluster_counter = 0       # [cluster1_new_elements_added,
cluster2_new_elements_added...]
    for i in range(len(centroids)):
        for j in point_to_cluster_mapping:
            if point_to_cluster_mapping[j] == i:
                centroids[i][0] += datapoints[j][0]   # x-coordinate adding
                centroids[i][1] += datapoints[j][1]   # y-coordinate adding
                cluster_counter += 1
        if cluster_counter != 0:
            centroids[i][0] = round(centroids[i][0]/cluster_counter,4)
            centroids[i][1] = round(centroids[i][1]/cluster_counter,4)
            cluster_counter = 0

    printResult(centroids, point_to_cluster_mapping)
    return centroids


def kmeans_iterator(centroids, datapoints):
    old_centroids = centroids
    new_centroids = centroids
    iteration = 0

    while iteration != 15:
        iteration += 1
        print("\nIteration ", iteration)
        old_centroids = new_centroids
        new_centroids = kmeansclustering(new_centroids, datapoints)


centroids = [[2,10], [5,8], [1,2]]
datapoints = [[2,10], [2,5], [8,4], [5,8], [7,5], [6,4], [1,2], [4,9]]


kmeans_iterator(centroids, datapoints)
```

2. Decision tree:

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

df= pd.read_csv("/home/admn/Downloads/zoo1.csv")
df.head()

class_type_output = df["class_type"]
df = df.drop("class_type", axis=1).drop("animal_name",axis=1)
print(df)
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(df, class_type_output,
test_size=0.20)


from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(x_train, y_train)


y_prediction = classifier.predict(x_test)
y_prediction


from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
confusion_matrix(y_test,y_prediction)


print(classification_report(y_test, y_prediction))


print(accuracy_score(y_test, y_prediction))


predicted_class = list(y_prediction)
actual_class = list(y_test)
for i in range(len(predicted_class)):
    print("Predicted class =", predicted_class[i],"\tActual class
=",actual_class[i])
```

3. Linear regression:

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


df = pd.read_csv("/home/admn/Downloads/student_scores.csv")
df


df.plot(x="Hours", y="Scores", style="o")
plt.show()
```

```
x_mean = df["Hours"].mean()
y_mean = df["Scores"].mean()
print(x_mean, y_mean)
```

```
df["x"] = df["Hours"] - x_mean
df["y"] = df["Scores"] - y_mean
df["x*y"] = df["x"] * df["y"]
df["x^2"] = df["x"]**2
df["y^2"] = df["y"]**2
df
```

```
summation_x_y = df["x*y"].sum()
summation_x_squared = df["x^2"].sum()
summation_y_squared = df["y^2"].sum()
print(summation_x_y, summation_x_squared, summation_y_squared)
```

```
correlation = summation_x_y / (summation_x_squared *
summation_y_squared)**0.5
correlation
```

```
def getMean(numbers):
    if len(numbers) == 0:
        return None
    else:
        current_sum = 0
        for i in numbers:
            current_sum += i
            current_avg = current_sum/len(numbers)
        return current_avg

def getStandardDeviation(numbers):
    if len(numbers) == 0:
        return 0
    else:
        mean = getMean(numbers)
        std_deviation = 0
        for i in numbers:
            std_deviation += (i - mean)**2
        return (std_deviation/len(numbers))**0.5
```

```
std_deviation_x = getStandardDeviation(df["x"].tolist())
std_deviation_y = getStandardDeviation(df["y"].tolist())
print(std_deviation_x, std_deviation_y)
```

```
m = correlation * (std_deviation_y / std_deviation_x)
m
```

```
c = df["Scores"].mean() - m * df["Hours"].mean()
c
```

```
df["y_prediction"] = m * df["Hours"] + c
df
```

```
plot1 = plt.scatter(df["Hours"], df["Scores"])
plot2 = plt.scatter(df["Hours"], df["y_prediction"])
plt.show()
```

4.mean, median, mode, standard variation and normalization:

```
def getMode(numbers):
    max_occur = -1
    if len(numbers) == 0:
        return None
    else:
        occurences = {}
        for i in numbers:
            if occurences.get(i) == None:
                occurences[i] = 1
            else:
                occurences[i] += 1
            if occurences[i] > max_occur:
                max_occur = occurences[i]

    # get max occurence number
    for i in occurences:
        if occurences[i] == max_occur:
            return i
    return None
```

```
def getMean(numbers):
    if len(numbers) == 0:
        return None
    else:
```

```
        current_sum = 0
        for i in numbers:
            current_sum += i
            current_avg = current_sum/len(numbers)
        return current_avg


def getMedian():
    numbers = []
    inp = 0
    while True:
        inp = int(input("Enter a number OR type 'exit'"))
        if inp == 'exit':
            break
        else:
            numbers.append(inp)
    if len(numbers) == 0:
        return None
    else:
        middle_index = len(numbers)//2
        return numbers[middle_index]


def getStandardDeviation(numbers):
    if len(numbers) == 0:
        return 0
    else:
        mean = getMean(numbers)
        std_deviation = 0
        for i in numbers:
            std_deviation += (i - mean)**2
        return (std_deviation/len(numbers))**0.5


def getVariance(numbers):
    return getStandardDeviation(numbers)**2


def getNormalization(features):
    x_min = min(features)
    x_max = max(features)
    normalized_vals = []
    for i in features:
        normalized_vals.append((i - x_min)/(x_max - x_min))
    return normalized_vals
getNormalization([10,20,30,40])
```

```
def getStandardization(features):
    mean = getMean(features)
    std_deviation = getStandardDeviation(features)
    standardized_vals = []
    for i in features:
        standardized_vals.append((i - mean)/std_deviation)
    return standardized_vals
getStandardization([10,20,30,40])



# MinMax Normalization
def doMinMaxNormalization(numbers):
    result = []
    if len(numbers) == 0:
        return result
    else:
        min_value = min(numbers)
        max_value = max(numbers)
        for i in numbers:
            result.append((i - min_value)/(max_value - min_value))
        return result

features = [100000,-2,50,12,700,9000]
print(doMinMaxNormalization(features))
```