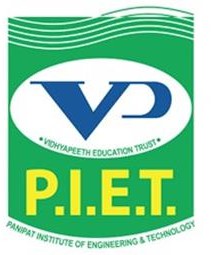
**PANIPAT INSTITUTE OF ENGINEERING & TECHNOLOGY SAMALKHA**

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

****

# Practical File Of

**“Reinforcement Learning Lab” PE-CS-AIDS-404LA**

**Submitted By: Submitted To:**

Name: Kunal Tyagi Ms. Roshni Jha

Roll No.: 2821412 Assistant Professor

Branch: B.Tech. CSE (AI & DS) 8th Sem (CSE- AI & DS)

**Affiliated To**

****

# KURUKSHETRA UNIVERSITY KURUKSHETRA, INDIA

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Sr. No.*** | ***AIM*** | ***Page No.*** | ***Date*** | ***Signature*** |
| **1.** | The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20%.What is the probability that a student is absent given that today is Friday? Apply Bayes rule in python to get the result. |  |  |  |
| **2.** | Extract the data from database using python. |  |  |  |
| **3.** | Implement k-nearest neighbours classification using python. |  |  |  |
| **4.** | Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606,using the result of k-means clustering with 3 means(i.e.,3centroids) |  |  |  |
| **5.** | The following training examples data map description so find visuals onto high, medium and low Credit-worthiness.  Input attributes are (from left to right) income, recreation, job, status, age- group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `med Risk' in the dataset? |  |  |  |
| **6.** | Implement linear regression using python. |  |  |  |
| **7.** | Implement Naïve Bayes theorem to classify the English text |  |  |  |
| **8.** | Implement an algorithm to demonstrate the significance of genetic algorithm |  |  |  |

# PROGRAM- 1

**AIM:** The probability that it is Friday and that a student is absent is 3%. Since there are 5 school days in a week, the probability that it is Friday is 20%. What is the probability that a student is absent given that today is Friday? Apply Bayes rule in python to get the result.

## DESCRIPTION:

1. Given values:
   * P\_Friday\_and\_Absent = 0.03: This is the joint probability that it is Friday and a student is absent. In other words, there's a 3% chance that on any given day, it is Friday and the student is absent simultaneously.
   * P\_Friday = 0.20: This represents the marginal probability of it being Friday, which is 20%. This is based on the assumption that all weekdays are equally likely, thus each day (Monday through Friday) has a 20% chance of occurring.
2. Bayes' Rule Application:
   * P\_Absent\_given\_Friday: This variable will store the result of our Bayes' rule calculation. Bayes' rule in this context is applied as follows:

P(Absent | Friday) =

𝑃(𝐹𝑟𝑖𝑑𝑎𝑦 𝑎𝑛𝑑 𝐴𝑏𝑠𝑒𝑛𝑡)

𝑃(𝐹𝑟𝑖𝑑𝑎𝑦)

## CODE:

# Given values

This formula calculates the conditional probability of a student being absent given that it is Friday by dividing the probability of both being Friday and the student being absent by the probability of it being Friday.

P\_Friday\_and\_Absent = 0.03 # Probability of it being Friday and a student being absent P\_Friday = 0.20 # Probability that today is Friday

# Applying Bayes' rule to find P(Absent | Friday)

# P(Absent | Friday) = P(Friday and Absent) / P(Friday)

P\_Absent\_given\_Friday = P\_Friday\_and\_Absent / P\_Friday

print("Probability that a student is absent given that today is Friday:",P\_Absent\_given\_Friday,"or", P\_Absent\_given\_Friday\*100,"%")

## OUTPUT:



The calculation shows that the probability of a student being absent given that it is Friday is 15%. This insight can be useful in educational planning, attendance monitoring, or even in setting policies for student attendance.

# PROGRAM- 2

**AIM:** Extract the data from database using python.

**DESCRIPTION:** Setting up a Python script to connect to and interact with a MySQL database involves a few key steps, including installing the necessary library, configuring your MySQL server, and writing the Python code to establish the connection.

## STEPS:

1. **Install MySQL Server:** First, ensure that MySQL Server is installed on the machine. Download it from the [official MySQL website](https://dev.mysql.com/downloads/mysql/). Follow the installation instructions for the operating system.

## Set Up MySQL Database and User

* 1. Access MySQL: Open the terminal or command prompt and access the MySQL shell using the MySQL root user:

### mysql -u root -p

Enter the root password when prompted.

* 1. Create a Database and adding data: Once inside the MySQL shell, create a new database:

### CREATE DATABASE example\_db; USE example\_db;

**CREATE TABLE tableName(columnName datatype(size)); INSERT into tableName VALUES(<data>);**

* 1. Create a User and Grant Permissions: It's a good practice to use a dedicated user for the applications. Create a user and grant it the necessary permissions on the database:

**CREATE USER ‘user’@'localhost' IDENTIFIED BY ‘passsword’; GRANT ALL PRIVILEGES ON userDatabase.\* TO ‘user’@'localhost';**

**FLUSH PRIVILEGES;**

## Install Python and pymysql Library

* 1. Install Python: If not already installed, download and install Python from python.org. Make sure to add Python to your system's PATH.
  2. Install pymysql: Open your command line or terminal and install the pymysql library using pip:

### pip install pymysql

1. **Python Code to Connect to MySQL:** Write Python code to connect to the database and perform operations.
2. **Running the Python Scripts:** Run your Python script from the command line or your preferred IDE. It should connect to the MySQL database and print the MySQL server version.

## CODE:

import pymysql

def connect\_to\_mysql(): try:

# Connect to the database

conn = pymysql.connect(host='localhost', user='akshat', password='2820324', database='student', charset='utf8mb4')

# Create a cursor object cursor = conn.cursor() # Execute a query

cursor.execute("SELECT VERSION()") # Fetch one result

version = cursor.fetchone() print("Database version:", version)

# Query the database cursor.execute('SELECT \* FROM data')

print("\nExtracting Data from Database Table: 'DATA'\n",) # Fetch all rows

records = cursor.fetchall() # Print the records

for record in records: print(record)

except pymysql.MySQLError as e:

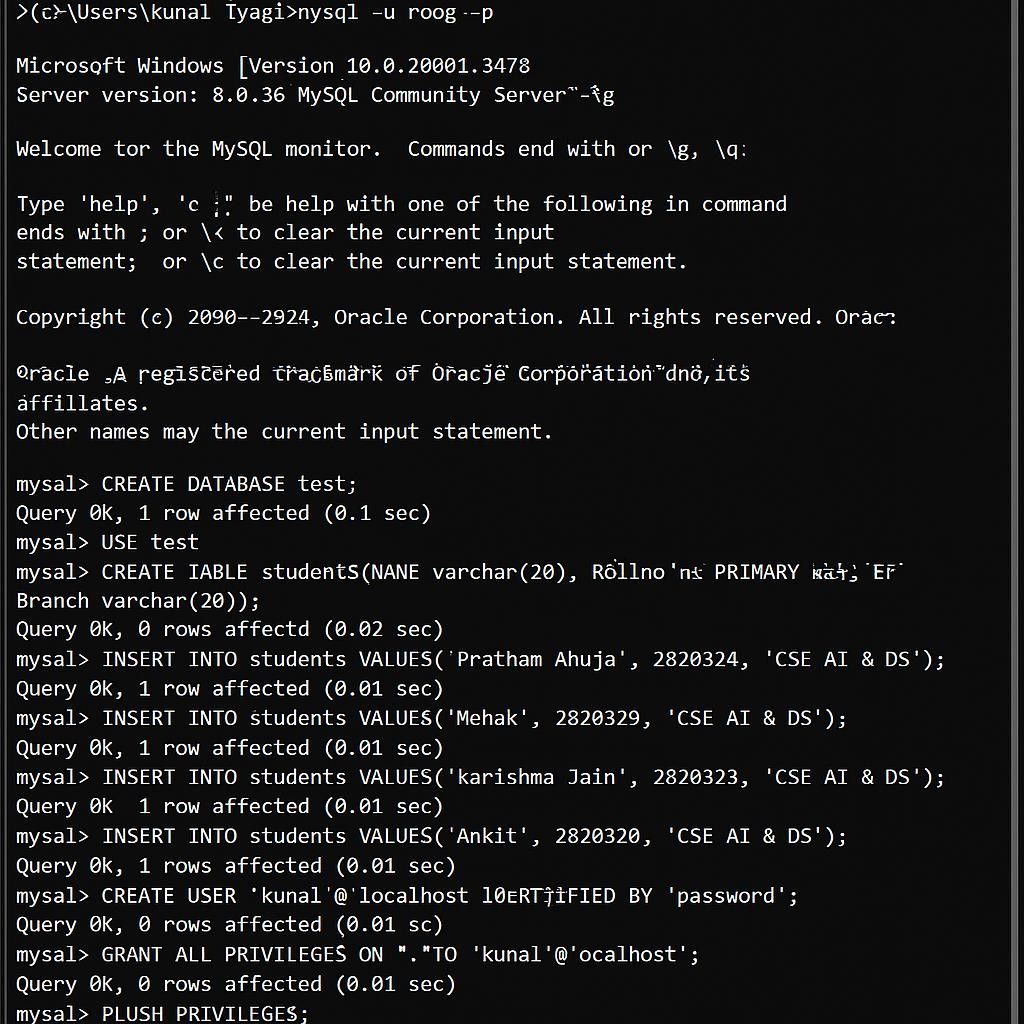
print("Error connecting to MySQL Platform:", e) finally:

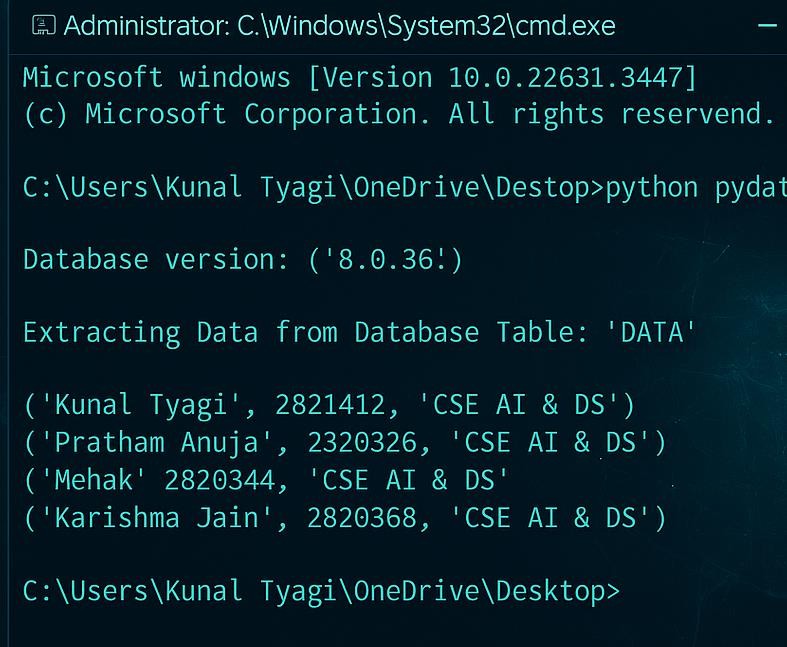
# Close the connection, if connection was successfully made if conn:

conn.close()

if name == " main ": connect\_to\_mysql()

**OUTPUT:**





# PROGRAM- 3

**AIM:** Implement k-nearest neighbors classification using python.

**DESCRIPTION:** The k-nearest neighbour algorithm is a simple yet powerful classification algorithm used for both regression and classification tasks. It is a non-parametric method used for classification and regression. In the context of classification, the algorithm works by assigning the class membership of a data point based on the majority class among its k-nearest neighbours in the feature space.

The k-NN working can be explained on the basis of the below algorithm:

**Step-1:** Select the number K of the neighbours

**Step-2:** Calculate the Euclidean distance of K number of neighbours

**Step-3:** Take the K nearest neighbours as per the calculated Euclidean distance.

**Step-4:** Among these k neighbours, count the number of the data points in each category.

**Step-5:** Assign the new data points to that category for which the number of the neighbour is maximum.

**Step-6:** Our model is ready.

## CODE:

# importing libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd #importing datasets

data\_set= pd.read\_csv('knn\_data.csv') data\_set.head()

#Extracting Independent and dependent Variable x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

# Splitting the dataset into training and test set.

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

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

from sklearn.preprocessing import StandardScaler st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train) x\_test= st\_x.transform(x\_test)

#feature Scaling

from sklearn.preprocessing import StandardScaler st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train) x\_test= st\_x.transform(x\_test)

#Fitting K-NN classifier to the training set

from sklearn.neighbors import KNeighborsClassifier

classifier= KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2 ) classifier.fit(x\_train, y\_train)

#Predicting the test set result

y\_pred= classifier.predict(x\_test)

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix cm= confusion\_matrix(y\_test, y\_pred)

import matplotlib.pyplot as plt import seaborn as sns

# Create heatmap plt.figure(figsize=(4, 3))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

# Add labels and title plt.title('Confusion Matrix') plt.xlabel('Predicted') plt.ylabel('True')

# Show plot plt.show()

# Visualizing the training set result

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

x1, x2 = np.meshgrid(np.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step

=0.01),

np.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(x1.min(), x1.max())

plt.ylim(x2.min(), x2.max())

for i, j in enumerate(np.unique(y\_set)): plt.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN Algorithm (Training set)') plt.xlabel('Age')

plylabel('Estimated Salary')

plt.legend() plt.show()

# Visualizing the training set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = np.meshgrid(np.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step

=0.01),

np.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(x1.min(), x1.max())

plt.ylim(x2.min(), x2.max())

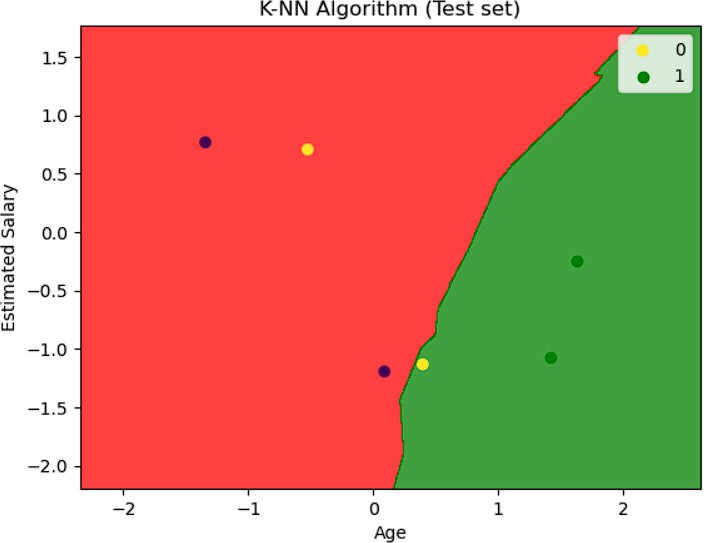
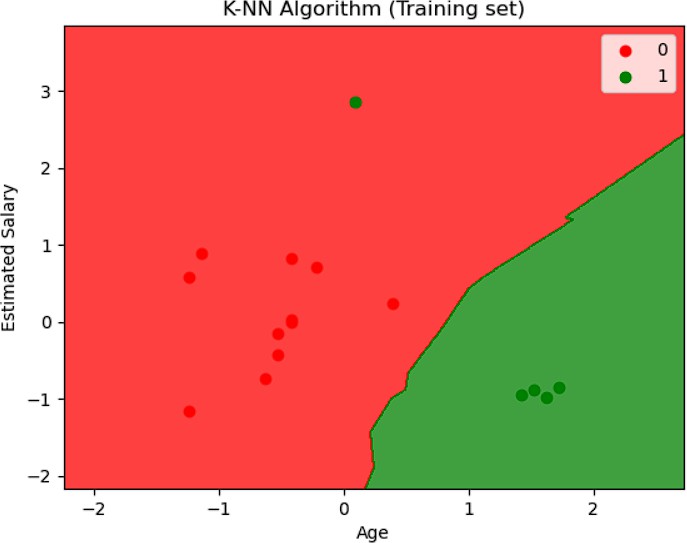
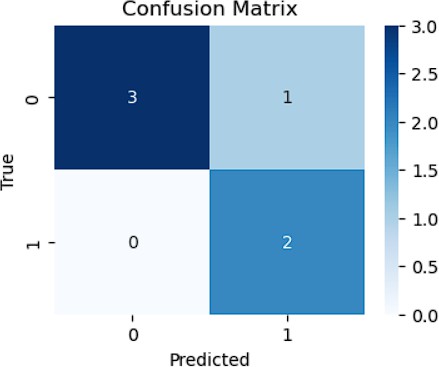
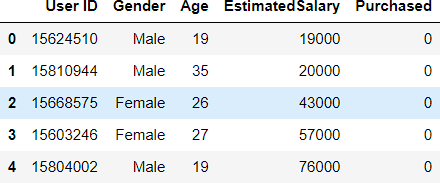
for i, j in enumerate(np.unique(y\_set)): plt.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN Algorithm (Test set)') plt.xlabel('Age') plt.ylabel('Estimated Salary')

plt.legend() plt.show()

**OUTPUT:**



# PROGRAM- 4

**AIM:** Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means(i.e.,3centroids).

|  |  |  |
| --- | --- | --- |
| VAR1 | VAR2 | CLASS |
| 1.713 | 1.586 | 0 |
| 0.180 | 1.786 | 1 |
| 0.353 | 1.240 | 1 |
| 0.940 | 1.566 | 0 |
| 1.486 | 0.759 | 1 |
| 1.266 | 1.106 | 0 |
| 1.540 | 0.419 | 1 |
| 0.459 | 1.799 | 1 |
| 0.773 | 0.186 | 1 |

**DESCRIPTION:** K-means clustering is a popular unsupervised machine learning algorithm used for clustering data points into groups or clusters. It aims to partition the data into k clusters where each data point belongs to the cluster with the nearest mean (centroid).

Here's a brief description of how the k-means clustering algorithm works:

1. Initialization: The algorithm starts by randomly initializing k centroids, which are the centers of the clusters.
2. Assignment Step: Each data point is assigned to the nearest centroid, based on the Euclidean distance between the data point and each centroid. This step creates clusters.
3. Update Step: After all data points are assigned to clusters, the centroids are recalculated as the mean of all data points assigned to each cluster.
4. Repeat: Steps 2 and 3 are repeated iteratively until the centroids no longer change significantly or a maximum number of iterations is reached.
5. Convergence: Eventually, the centroids converge to stable positions, and the algorithm stops.

K-means clustering aims to minimize the within-cluster sum of squares (WCSS), which measures the distance between each data point and its assigned centroid. The algorithm's objective is to find centroids that minimize the total WCSS across all clusters. K-means clustering is widely used for

various tasks, including customer segmentation, document clustering, image segmentation, and anomaly detection. However, it has some limitations, such as sensitivity to the initial centroids and the need to specify the number of clusters (k) beforehand.

When using k-means clustering, it's essential to evaluate the results carefully and consider factors such as the distribution of data points, cluster cohesion, and cluster separation. Additionally, preprocessing steps such as scaling or normalization may be necessary to improve clustering performance.

## CODE:

import pandas as pd import numpy as np

from sklearn.cluster import KMeans # Given data

data = {

'VAR1': [1.713, 0.180, 0.353, 0.940, 1.486, 1.266, 1.540, 0.459, 0.773],

'VAR2': [1.586, 1.786, 1.240, 1.566, 0.759, 1.106, 0.419, 1.799, 0.186],

'CLASS': [0, 1, 1, 0, 1, 0, 1, 1, 1]

}

# Convert data to DataFrame df = pd.DataFrame(data)

print(df)

# Fit k-means clustering with 3 clusters

kmeans = KMeans(n\_clusters=3, random\_state=42) kmeans.fit(df[['VAR1', 'VAR2']])

# Get test data from the user print("Provide the Test Data to Predict ")

VAR1 = float(input("Enter Value for VAR1 :")) VAR2 = float(input("Enter Value for VAR2 :"))

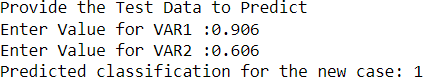
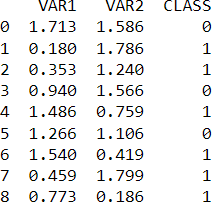
new\_case = pd.DataFrame({'VAR1': [VAR1], 'VAR2': [VAR2]}) predicted\_cluster = kmeans.predict(new\_case)

# Get the indices of data points in the predicted cluster cluster\_indices = np.where(kmeans.labels\_ == predicted\_cluster)[0]

# Determine the majority class for the predicted cluster majority\_class = df.loc[cluster\_indices, 'CLASS'].mode()[0]

print("Predicted classification for the new case:", majority\_class)

**OUTPUT:**



# PROGRAM- 5

**AIM:** The following training examples map description so find visuals onto high, medium and low Credit-worthiness.

medium skiing design single twenties no ->high Risk high golf trading married forties yes ->low Risk

low speedway transport married thirties yes ->med Risk medium football banking single thirties yes ->low Risk high flying mediamarried fifties yes ->high Risk

low football security single twenties no ->med Risk medium golf media single thirties yes ->med Risk medium golftransport married forties yes ->low Risk high skiing bankingsingle thirties yes ->high Risk

low golf unemployed married forties yes ->high Risk

Input attributes are (from left to right) income, recreation, job, status, age- group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `med Risk' in the dataset?

## DESCRIPTION:

The problem involves analyzing a dataset containing attributes such as income, recreation, job, marital status, age group, and home ownership, along with corresponding credit-worthiness categories (high, medium, low risk). Each entry in the dataset represents an individual's profile and their credit-worthiness category.

The goal is to compute the unconditional probability of individuals playing golf and the conditional probability of individuals being single given that they are in the medium risk category. To solve this problem, we need to count occurrences of relevant attributes and conditions in the dataset and then apply probability formulas to calculate the desired probabilities. Finally, a Python program is developed to perform these computations and obtain the results.

To find the unconditional probability of 'golf' and the conditional probability of 'single' given 'med Risk' in the dataset, we first need to count occurrences of each relevant combination.

Let's denote:

N as the total number of examples in the dataset.

N\_golf as the number of examples where 'golf' is mentioned.

N\_single as the number of examples where 'single' status is mentioned. N\_med\_risk as the number of examples classified as 'med Risk'.

N\_golf\_and\_med\_risk) as the number of examples where both 'golf' and 'med Risk' are mentioned.

N\_single\_and\_med\_risk as the number of examples where both 'single' and 'med Risk' are mentioned.

Then, the unconditional probability of 'golf' is given by:

### P(golf) = N\_golf / N

And the conditional probability of 'single' given 'med Risk' is given by:

### P(single | med\_risk) = N\_single\_and\_med\_risk / N\_med\_risk

Without the actual counts, we cannot compute these probabilities. We need to count how many times 'golf', 'single', 'med Risk', 'golf and med Risk', and 'single and med Risk' appear in the dataset. Once we have those counts, we can plug them into the formulas above to calculate the probabilities.

## CODE:

# Define the dataset dataset = [

("medium", "skiing", "design", "single", "twenties", "no"), # high Risk

("high", "golf", "trading", "married", "forties", "yes"), # low Risk

("low", "speedway", "transport", "married", "thirties", "yes"), # med Risk

("medium", "football", "banking", "single", "thirties", "yes"), # low Risk

("high", "flying", "media", "married", "fifties", "yes"), # high Risk

("low", "football", "security", "single", "twenties", "no"), # med Risk

("medium", "golf", "media", "single", "thirties", "yes"), # med Risk

("medium", "golf", "transport", "married", "forties", "yes"), # low Risk

("high", "skiing", "banking", "single", "thirties", "yes"), # high Risk

("low", "golf", "unemployed", "married", "forties", "yes") # high Risk

]

# Count occurrences total\_instances = len(dataset)

golf\_instances = sum(1 for item in dataset if item[1] == 'golf') single\_instances = sum(1 for item in dataset if item[3] == 'single') med\_risk\_instances = sum(1 for item in dataset if item[-1] == 'med Risk')

golf\_med\_risk\_instances = sum(1 for item in dataset if item[1] == 'golf' and item[-1] == 'med Risk')

single\_med\_risk\_instances = sum(1 for item in dataset if item[3] == 'single' and item[-1] == 'med Risk')

# Calculate probabilities

unconditional\_prob\_golf = golf\_instances / total\_instances

# Calculate conditional probability (handling division by zero) if med\_risk\_instances != 0:

conditional\_prob\_single\_given\_med\_risk = single\_med\_risk\_instances / med\_risk\_instances else:

conditional\_prob\_single\_given\_med\_risk = 0

# Print results

print("Unconditional probability of 'golf':", unconditional\_prob\_golf)

print("Conditional probability of 'single' given 'medium Risk':", conditional\_prob\_single\_given\_med\_risk)

**OUTPUT:**



# PROGRAM- 6

**AIM:** Implement linear regression using python.

**DESCRIPTION:** [Linear regression](https://www.geeksforgeeks.org/ml-linear-regression/) is a statistical method that is used to predict a continuous dependent variable(target variable) based on one or more independent variables(predictor variables). This technique assumes a linear relationship between the dependent and independent variables, which implies that the dependent variable changes proportionally with changes in the independent variables. In other words, linear regression is used to determine the extent to which one or more variables can predict the value of the dependent variable.

## CODE:

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')

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {}\nb\_1 = {}".format(b[0], b[1]))

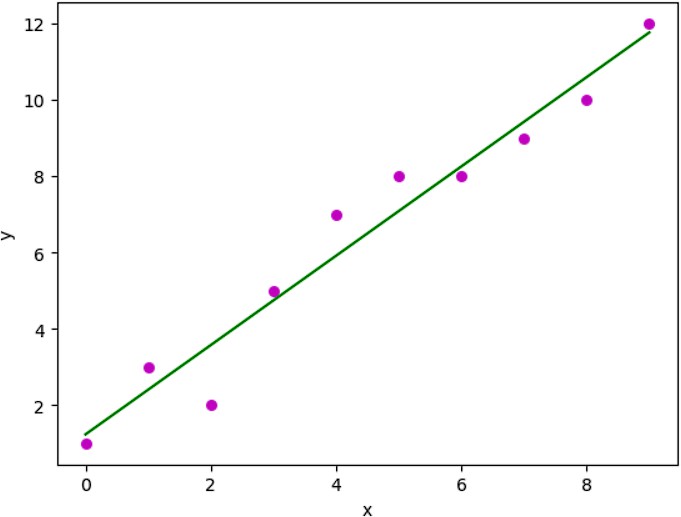
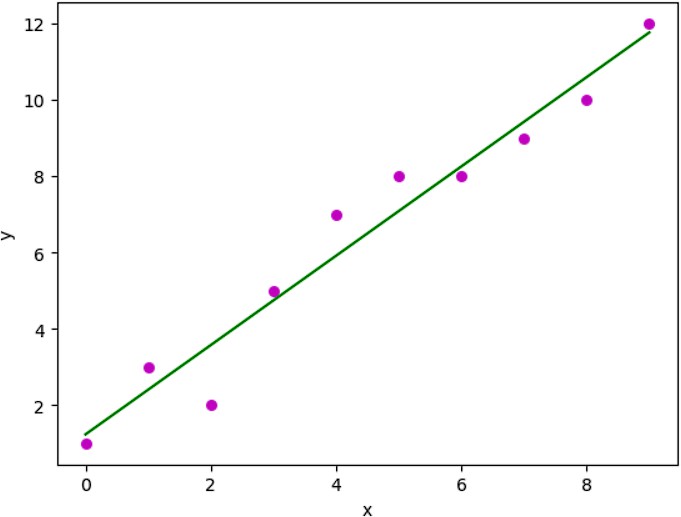
# plotting regression line plot\_regression\_line(x, y, b)

plt.show()

# Call the main function

if name == " main ": main()

**OUTPUT:**



# PROGRAM- 7

**AIM:** Implement Naïve Bayes theorem to classify the English text.

**DESCRIPTION:** The [Naive Bayes algorithm](https://www.geeksforgeeks.org/ml-naive-bayes-scratch-implementation-using-python/) is a probabilistic classification method that bases its predictions on the Bayes theorem. Based on observable data, the Bayes theorem determines a hypothesis’s probability. When using Naive Bayes, an instance’s features serve as the evidence, while the class to which the instance belongs serves as the hypothesis.

The algorithm employing the Bayes theory is broken down as follows:

### Bayes Theorem

* P(C|F): Probability of the instance belonging to a specific class given its features.
* P(F|C): Probability of observing the features given the class.
* P(C): Prior probability of the class.
* P(F): Probability of observing the features.

The assumption of feature independence is what gives Naive Bayes its “naive” quality. It is computationally efficient since this makes calculations simpler.



## CODE:

import prettytable

print('\n \*-----\* Classification using Naïve bayes \* \* \n')

total\_documents = int(input("Enter the Total Number of documents: ")) doc\_class = []

i = 0 keywords = []

while not i == total\_documents: doc\_class.append([])

text = input(f"\nEnter the text of Doc-{i+1} : ").lower() clas = input(f"Enter the class of Doc-{i+1} : ") doc\_class[i].append(text.split()) doc\_class[i].append(clas)

keywords.extend(text.split()) i = i+1

keywords = set(keywords) keywords = list(keywords) keywords.sort()

to\_find = input(

"\nEnter the Text to classify using Naive Bayes: ").lower().split()

probability\_table = []

for i in range(total\_documents): probability\_table.append([]) for j in keywords:

probability\_table[i].append(0) doc\_id = 1

for i in range(total\_documents): for k in range(len(keywords)):

if keywords[k] in doc\_class[i][0]:

probability\_table[i][k] += doc\_class[i][0].count(keywords[k]) print('\n')

keywords.insert(0, 'Document ID') keywords.append("Class") Prob\_Table = prettytable.PrettyTable() Prob\_Table.field\_names = keywords

Prob\_Table.title = 'Probability of Documents'

x = 0

for i in probability\_table: i.insert(0, x+1) i.append(doc\_class[x][1]) Prob\_Table.add\_row(i)

x = x+1 print(Prob\_Table) print('\n')

for i in probability\_table: i.pop(0)

totalpluswords = 0

totalnegwords = 0

totalplus = 0

totalneg = 0

vocabulary = len(keywords)-2 for i in probability\_table:

if i[len(i)-1] == "+": totalplus += 1

totalpluswords += sum(i[0:len(i)-1]) else:

totalneg += 1

totalnegwords += sum(i[0:len(i)-1]) keywords.pop(0) keywords.pop(len(keywords)-1)

# For Positive class temp = []

for i in to\_find: count = 0

x = keywords.index(i) for j in probability\_table:

if j[len(j)-1] == "+": count = count+j[x]

temp.append(count) count = 0

for i in range(len(temp)):

temp[i] = format((temp[i]+1)/(vocabulary+totalpluswords), ".4f") print()

temp = [float(f) for f in temp]

print("Probabilities of Each word to be in '+' class are: ") h = 0

for i in to\_find:

print(f"P({i}/+) = {temp[h]}") h = h+1

print()

pplus = float(format((totalplus)/(totalplus+totalneg), ".8f")) for i in temp:

pplus = pplus\*i

pplus = format(pplus, ".8f")

print("Probability of Given text to be in '+' class is :", pplus) print()

# For Negative class temp = []

for i in to\_find: count = 0

x = keywords.index(i) for j in probability\_table:

if j[len(j)-1] == "-": count = count+j[x]

temp.append(count) count = 0

for i in range(len(temp)):

temp[i] = format((temp[i]+1)/(vocabulary+totalnegwords), ".4f") print()

temp = [float(f) for f in temp]

print("Probabilities of Each word to be in '-' class are: ") h = 0

for i in to\_find:

print(f"P({i}/-) = {temp[h]}") h = h+1

print()

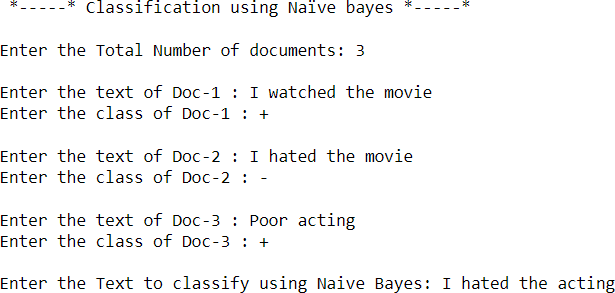
pneg = float(format((totalneg)/(totalplus+totalneg), ".8f")) for i in temp:

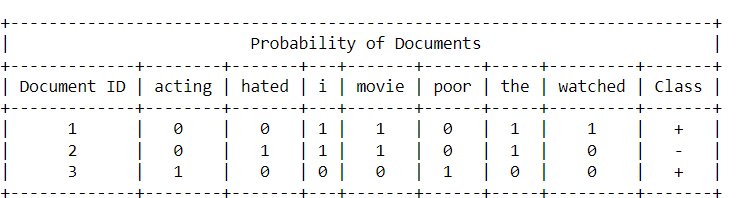
pneg = pneg\*i

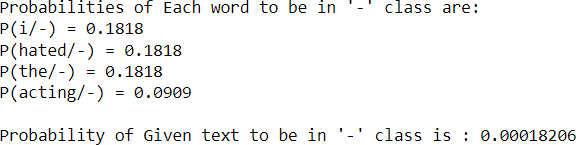
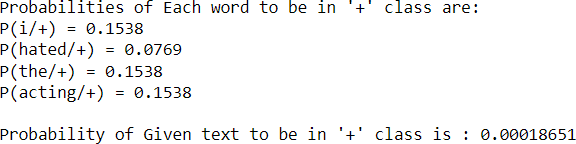
pneg = format(pneg, ".8f")

print("Probability of Given text to be in '-' class is :", pneg) print('\n')

**OUTPUT:**







# PROGRAM- 8

**AIM:** Implement an algorithm to demonstrate the significance of genetic algorithm.

**DESCRIPTION:** Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space**. They are commonly used to generate high-quality solutions for optimization problems and search problems.**

**Genetic algorithms simulate the process of natural selection** which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate “survival of the fittest” among individuals of consecutive generations to solve a problem. **Each generation consists of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

**Operators of Genetic Algorithms:** Once the initial generation is created, the algorithm evolves the generation using following operators:-

1. Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.
2. Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).
3. Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

**CODE:**

import random

# Number of individuals in each generation POPULATION\_SIZE = 100

# Valid genes

GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP QRSTUVWXYZ 1234567890, .-;:\_!"#%&/()=?@${[]}'''

# Target string to be generated TARGET = "I love GeeksforGeeks"

PIET

class Individual(object): '''

Class representing individual in population '''

def init (self, chromosome): self.chromosome = chromosome self.fitness = self.cal\_fitness()

@classmethod

def mutated\_genes(self): '''

create random genes for mutation '''

global GENES

gene = random.choice(GENES) return gene

@classmethod

def create\_gnome(self):

'''

create chromosome or string of genes '''

global TARGET gnome\_len = len(TARGET)

return [self.mutated\_genes() for \_ in range(gnome\_len)]

def mate(self, par2): '''

Perform mating and produce new offspring '''

# chromosome for offspring child\_chromosome = []

for gp1, gp2 in zip(self.chromosome, par2.chromosome):

# random probability prob = random.random()

# if prob is less than 0.45, insert gene # from parent 1

if prob < 0.45: child\_chromosome.append(gp1)

# if prob is between 0.45 and 0.90, insert # gene from parent 2

elif prob < 0.90: child\_chromosome.append(gp2)

# otherwise insert random gene(mutate), # for maintaining diversity

else:

child\_chromosome.append(self.mutated\_genes())

# create new Individual(offspring) using # generated chromosome for offspring return Individual(child\_chromosome)

def cal\_fitness(self): '''

Calculate fitness score, it is the number of characters in string which differ from target string.

'''

global TARGET fitness = 0

for gs, gt in zip(self.chromosome, TARGET): if gs != gt: fitness+= 1

return fitness

# Driver code def main():

global POPULATION\_SIZE #current generation generation = 1

found = False population = []

# create initial population

for \_ in range(POPULATION\_SIZE):

gnome = Individual.create\_gnome() population.append(Individual(gnome))

while not found:

# sort the population in increasing order of fitness score population = sorted(population, key = lambda x:x.fitness)

# if the individual having lowest fitness score ie.

# 0 then we know that we have reached to the target # and break the loop

if population[0].fitness <= 0: found = True

break

# Otherwise generate new offsprings for new generation

new\_generation = []

# Perform Elitism, that mean 10% of fittest population # goes to the next generation

s = int((10\*POPULATION\_SIZE)/100)

new\_generation.extend(population[:s])

# From 50% of fittest population, Individuals # will mate to produce offspring

s = int((90\*POPULATION\_SIZE)/100)

for \_ in range(s):

parent1 = random.choice(population[:50]) parent2 = random.choice(population[:50]) child = parent1.mate(parent2) new\_generation.append(child)

population = new\_generation

print("Generation: {}\tString: {}\tFitness: {}".format(generation, "".join(population[0].chromosome),

population[0].fitness))

generation += 1

print("Generation: {}\tString: {}\tFitness: {}".format(generation, "".join(population[0].chromosome),

population[0].fitness)) if name == ' main ':

main()

**OUTPUT:**

