A close up of a document

Description automatically generatedA logo of a institute of technology

Description automatically generated

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

# 22ADL38 PRINCIPLES OF DATA SCIENCE LABORATORY



**RECORD NOTE BOOK**

REGNO.

Certified that this is a bonafide observation of Practical work done by Mr/Ms/Mrs……………………………………of the……………………………………

Semester………………………………………………. Branch during the Academic year……………in the… laboratory.

# Staff–in–Charge Head of the Department

**Internal Examiner External Examine**

**LIST OF EXPERIMENTS**

| **S.No** | **NAME OF THE EXPERIMENT** |
| --- | --- |
| 1. | Working with Numpy arrays |
| 2. | Working with Pandas data frames |
| 3. | Basic plots using Matplotlib |
| 4. | Working with Vectors |
| 5. | Develop a python program to find covariance and correlation |
| 6. | Develop a Python program with adequate comments to examine, the probability both are girls given that the older child is a girl. The program also needs to examine the probability that both are girls given that at least one of the children is a girl. |
| 7. | Develop a Python program with adequate comments to find Gradient Descent for square function |
| 8. | Linear Regression |
| 9. |  |
| 10. |  |

**CONTENTS**

| **Sl. No.** | **Name of the Experiment** | **Page No.** | **Marks (10)** | **Staff Signature** |
| --- | --- | --- | --- | --- |
| 1. | Working with Numpy arrays |  |  |  |
| 2. | Working with Pandas data frames |  |  |  |
| 3. | Basic plots using Matplotlib |  |  |  |
| 4. | Working with Vectors |  |  |  |
| 5. | Develop a python program to find covariance and correlation |  |  |  |
| 6. | Develop a Python program with adequate comments to examine, the probability both are girls given that the older child is a girl. The program also needs to examine the probability that both are girls given that at least one of the children is a girl. |  |  |  |
| 7. | Develop a Python program with adequate comments to find Gradient Descent for square function |  |  |  |
| 8. | Linear Regression |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Ex no: 1 Working with Numpy arrays**

### AIM

Working with Numpy arrays

### ALGORITHM

Step1: Start

Step2: Import numpy module

Step3: Print the basic characteristics and operactions of array Step4: Stop

### PROGRAM

import numpy as np

# Creating array object arr = np.array( [[ 1, 2, 3],

[ 4, 2, 5]] )

# Printing type of arr object print("Array is of type: ", type(arr)) # Printing array dimensions (axes)

print("No. of dimensions: ", arr.ndim) # Printing shape of array print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array print("Size of array: ", arr.size)

# Printing type of elements in array

print("Array stores elements of type: ", arr.dtype)

### OUTPUT

Array is of type: <class 'numpy.ndarray'> No. of dimensions: 2

Shape of array: (2, 3) Size of array: 6

Array stores elements of type: int32

### Program to Perform Array Slicing

a = np.array([[1,2,3],[3,4,5],[4,5,6]])

print(a)

print("After slicing") print(a[1:])

### Output

[[1 2 3]

[3 4 5]

[4 5 6]]

After slicing [[3 4 5]

[4 5 6]]

### Program to Perform Array Slicing

# array to begin with import numpy as np

a = np.array([[1,2,3],[3,4,5],[4,5,6]])

print('Our array is:' ) print(a)

# this returns array of items in the second column print('The items in the second column are:' ) print(a[...,1])

print('\n' )

# Now we will slice all items from the second row print ('The items in the second row are:' ) print(a[1,...])

print('\n' )

# Now we will slice all items from column 1 onwards print('The items column 1 onwards are:' ) print(a[...,1:])

### Output:

Our array is:

[[1 2 3]

[3 4 5]

[4 5 6]]

The items in the second column are:

[2 4 5]

The items in the second row are:

[3 4 5]

The items column 1 onwards are:

[[2 3]

[4 5]

[5 6]]

### Result:

Thus the working with Numpy arrays was successfully completed.

### Ex no: 2 Create a dataframe using a list of elements. Aim:

To work with Pandas data frames

### ALGORITHM

Step1: Start

Step2: import numpy and pandas module Step3: Create a dataframe using the dictionary Step4: Print the output

Step5: Stop

### PROGRAM

import numpy as np import pandas as pd

data = np.array([['','Col1','Col2'], ['Row1',1,2],

['Row2',3,4]])

print(pd.DataFrame(data=data[1:,1:],

index = data[1:,0], columns=data[0,1:]))

# Take a 2D array as input to your DataFrame my\_2darray = np.array([[1, 2, 3], [4, 5, 6]]) print(pd.DataFrame(my\_2darray))

# Take a dictionary as input to your DataFrame my\_dict = {1: ['1', '3'], 2: ['1', '2'], 3: ['2', '4']}

print(pd.DataFrame(my\_dict))

# Take a DataFrame as input to your DataFrame

my\_df = pd.DataFrame(data=[4,5,6,7], index=range(0,4), columns=['A']) print(pd.DataFrame(my\_df))

# Take a Series as input to your DataFrame

my\_series = pd.Series({"United Kingdom":"London", "India":"New Delhi", "United States":"Washington", "Belgium":"Brussels"})

print(pd.DataFrame(my\_series))

df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]))

# Use the `shape` property print(df.shape)

# Or use the `len()` function with the `index` property print(len(df.index))

### Output:

Col1 Col2

| Row1 | | | 1 | 2 |
| --- | --- | --- | --- | --- |
| Row2 | | | 3 | 4 |
| 0 | 1 | 2 |  |  |
| 0 | 1 | 2 | 3 |  |
| 1 | 4 | 5 | 61 | 2 3 |
| 0 | 1 | 1 | 2 |  |
| 1 | 3 | 2 | 4A |  |
| 0 | 4 |  |  |  |
| 1 | 5 |  |  |  |
| 2 | 6 |  |  |  |
| 3 | 7 |  |  |  |
| 0 |  |  |  |  |

United Kingdom London India New Delhi United States Washington Belgium Brussels

(2, 3)

2

### Result:

Thus the working with Pandas data frames was successfully completed.

### Ex. No.:3 Basic plots using Matplotlib Aim:

To draw basic plots in Python program using Matplotlib

### ALGORITHM

Step1: Start

Step2: import Matplotlib module

Step3: Create a Basic plots using Matplotlib Step4: Print the output

Step5: Stop

### Program:3a

# importing the required module import matplotlib.pyplot as plt

# x axis values x = [1,2,3]

# corresponding y axis values y = [2,4,1]

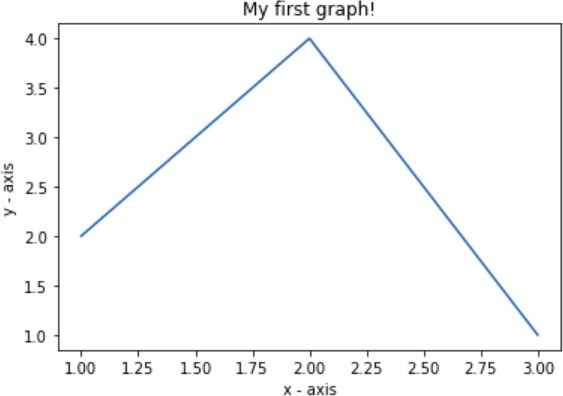
# plotting the points plt.plot(x, y)

# naming the x axis plt.xlabel('x - axis') # naming the y axis plt.ylabel('y - axis')

# giving a title to my graph plt.title('My first graph!')

# function to show the plot plt.show()

**Output:**



**Program:3b**

import matplotlib.pyplot as plt a = [1, 2, 3, 4, 5]

b = [0, 0.6, 0.2, 15, 10, 8, 16, 21]

plt.plot(a)

# o is for circles and r is # for red

plt.plot(b, "or") plt.plot(list(range(0, 22, 3)))

# naming the x-axis plt.xlabel('Day ->')

# naming the y-axis plt.ylabel('Temp ->')

c = [4, 2, 6, 8, 3, 20, 13, 15]

plt.plot(c, label = '4th Rep')

# get current axes command ax = plt.gca()

# get command over the individual # boundary line of the graph body ax.spines['right'].set\_visible(False) ax.spines['top'].set\_visible(False)

# set the range or the bounds of

# the left boundary line to fixed range ax.spines['left'].set\_bounds(-3, 40)

# set the interval by which # the x-axis set the marks

plt.xticks(list(range(-3, 10)))

# set the intervals by which y-axis # set the marks plt.yticks(list(range(-3, 20, 3)))

# legend denotes that what color # signifies what

ax.legend(['1st Rep', '2nd Rep', '3rd Rep', '4th Rep'])

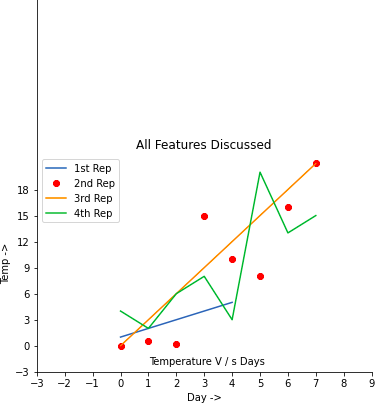
# annotate command helps to write

# ON THE GRAPH any text xy denotes # the position on the graph

plt.annotate('Temperature V / s Days', xy = (1.01, -2.15))

# gives a title to the Graph plt.title('All Features Discussed') plt.show()

### Output:



**Result:**

Thus the basic plots using Matplotlib in Python program was successfully completed.

### Ex. No.:4 Programs for working on Vectors

### Aim:

To work on vectors

### ALGORITHM

Step1: Start

Step2: import Math

Step3: Create all Basic functions to work on vectors

Step4: Print the output

Step5: Stop

import math

from functools import reduce

vector1 = [1, 2, 3]

vector2 = [4, 5, 6]

scalar = 3

vectors\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

vector = [1, 2, 3]

def vector\_add(v, w):

"""Adds two vectors componentwise."""

return [v\_i + w\_i for v\_i, w\_i in zip(v, w)]

result = vector\_add(vector1, vector2)

print("Result of vector addition:", result)

def vector\_subtract(v, w):

"""Subtracts two vectors componentwise."""

return [v\_i - w\_i for v\_i, w\_i in zip(v, w)]

result = vector\_subtract(vector1, vector2)

print("Result of vector subtraction:", result)

def scalar\_multiply(c, v):

"""Multiplies each element of a vector by a scalar."""

return [c \* v\_i for v\_i in v]

result = scalar\_multiply(scalar, vector)

print("Result of scalar multiplication:", result)

def vector\_add(v, w):

"""Adds two vectors componentwise."""

return [v\_i + w\_i for v\_i, w\_i in zip(v, w)]

def scalar\_multiply(c, v):

"""Multiplies each element of a vector by a scalar."""

return [c \* v\_i for v\_i in v]

def vector\_sum(vectors):

"""Returns the sum of a list of vectors."""

return reduce(vector\_add, vectors)

result = vector\_sum(vectors\_list)

print("Result of vector sum:", result)

def vector\_mean(vectors):

"""Compute the vector whose i-th element is the mean of the

i-th elements of the input vectors."""

n = len(vectors)

return scalar\_multiply(1/n, vector\_sum(vectors))

result = vector\_mean(vectors\_list)

print("Result of vector mean:", result)

def dot(v, w):

"""Calculates the dot product of two vectors."""

return sum(v\_i \* w\_i for v\_i, w\_i in zip(v, w))

result = dot(vector1, vector2)

print("Dot product result:", result)

def sum\_of\_squares(v):

"""Calculates the sum of squares of the elements in a vector."""

return dot(v, v)

result = sum\_of\_squares(vector)

print("Sum of squares result:", result)

def magnitude(v):

"""Calculates the magnitude of a vector."""

return math.sqrt(sum\_of\_squares(v))

result = magnitude(vector)

print("Magnitude result:", result)

def sum\_of\_squares(v):

"""Calculates the sum of squares of the elements in a vector."""

return sum(v\_i \*\* 2 for v\_i in v)

def squared\_distance(v, w):

"""Calculates the squared Euclidean distance between two vectors."""

return sum\_of\_squares(vector\_subtract(v, w))

def distance(v, w):

"""Calculates the Euclidean distance between two vectors."""

return math.sqrt(squared\_distance(v, w))

squared\_dist\_result = squared\_distance(vector1, vector2)

dist\_result = distance(vector1, vector2)

print("Squared distance result:", squared\_dist\_result)

print("Distance result:", dist\_result)

Result of vector addition: [5, 7, 9]

Result of vector subtraction: [-3, -3, -3]

Result of scalar multiplication: [3, 6, 9]

Result of vector sum: [12, 15, 18]

Result of vector mean: [4.0, 5.0, 6.0]

Dot product result: 32

Sum of squares result: 14

Magnitude result: 3.7416573867739413

Squared distance result: 27

Distance result: 5.196152422706632

**Ex No : 5**

Develop a python program to find covariance and correlation

from typing import List

def standard\_deviation(data: List[float]) -> float:

"""Calculate the standard deviation of a list of numbers"""

mean = sum(data) / len(data)

variance = sum((x - mean) \*\* 2 for x in data) / len(data)

return variance\*\*0.5

def covariance(xs: List[float], ys: List[float]) -> float:

"""Calculate the covariance between two lists of numbers"""

mean\_x = sum(xs) / len(xs)

mean\_y = sum(ys) / len(ys)

cov = sum((x - mean\_x) \* (y - mean\_y) for x, y in zip(xs, ys)) / len(xs)

return cov

def correlation(xs: List[float], ys: List[float]) -> float:

"""Measures how much xs and ys vary in tandem about their means"""

stdev\_x = standard\_deviation(xs)

stdev\_y = standard\_deviation(ys)

if stdev\_x > 0 and stdev\_y > 0:

return covariance(xs, ys) / stdev\_x / stdev\_y

else:

return 0 # if no variation, correlation is zero

# Example usage:

data1 = [1, 2, 3, 4, 5]

data2 = [2, 3, 4, 5, 6]

result = correlation(data1, data2)

print(f'Correlation: {result}')

result1 = covariance(data1, data2)

print(f'Covariance: {result1}')

**Ex No : 6**

Develop a Python program with adequate comments to examine, the probability both are girls given that the older child is a girl. The program also needs to examine the probability that both are girls given that at least one of the children is a girl.

import random

class Kid:

BOY = 0

GIRL = 1

def random\_kid() -> Kid:

"""Simulate a random child being a boy or a girl"""

return random.choice([Kid.BOY, Kid.GIRL])

def simulate\_family(num\_simulations: int = 100):

# Initialize counters for the desired probabilities

both\_girls\_given\_older\_girl = 0

both\_girls\_given\_either\_girl = 0

# Run simulations

for \_ in range(num\_simulations):

older\_child = random\_kid()

younger\_child = random\_kid()

# Check the conditions and update counters

if older\_child == Kid.GIRL:

both\_girls\_given\_older\_girl += (younger\_child == Kid.GIRL)

both\_girls\_given\_either\_girl += (older\_child == Kid.GIRL and younger\_child == Kid.GIRL)

elif younger\_child == Kid.GIRL:

both\_girls\_given\_either\_girl += 1

# Calculate probabilities

probability\_both\_girls\_given\_older\_girl = both\_girls\_given\_older\_girl / num\_simulations

probability\_both\_girls\_given\_either\_girl = both\_girls\_given\_either\_girl / num\_simulations

# Print results

print("Probability both are girls given that the older child is a girl:",

probability\_both\_girls\_given\_older\_girl)

print("Probability both are girls given that at least one is a girl:",

probability\_both\_girls\_given\_either\_girl)

# Run the simulation

simulate\_family(

**Ex No : 7**

Develop a Python program with adequate comments to find Gradient Descent for square function

def gradient\_descent(initial\_value, learning\_rate, num\_iterations):

# Define the function (example: f(x) = x^2)

def square\_function(x):

return x\*\*2

# Initialize the variable to store the current value

current\_value = initial\_value

# Perform gradient descent iterations

for iteration in range(num\_iterations):

# Calculate the gradient (derivative of the objective function)

gradient = 2 \* current\_value

# Update the current value using the gradient and learning rate

current\_value = current\_value - learning\_rate \* gradient

# Print the progress

print(f'Iteration {iteration + 1}: x = {current\_value}, f(x) = {square\_function(current\_value)}')

return current\_value

# Set the initial value, learning rate, and number of iterations

initial\_value = 3.0

learning\_rate = 0.1

num\_iterations = 10

# Perform gradient descent

final\_value = gradient\_descent(initial\_value, learning\_rate, num\_iterations)

# Print the final result

print(f'\nAfter {num\_iterations} iterations, the optimized value is x = {final\_value}')

**Ex. No.: 8 Simple Linear Regression**

**Aim:**

To write a python program for Simple Linear Regression

### ALGORITHM

Step 1: Start the Program

Step 2: Import numpy and matplotlib package Step 3: Define coefficient function

Step 4: Calculate cross-deviation and deviation about x Step 5: Calculate regression coefficients

Step 6: Plot the Linear regression and define main function

Step 7: Print the result

Step 8: Stop the process

### Program:

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])

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)

if name == " main ": main()

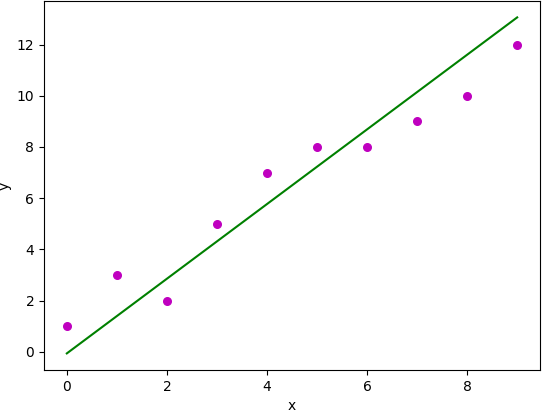
**Output :**

### Estimated coefficients:

b\_0 = -0.0586206896552

b\_1 = 1.45747126437

**Graph:**



# Result:

*Thus the computation for Simple Linear Regression was successfully completed.*