_____

(name of your school)

PRACTICAL FILE

Artificial Intelligence (Subject Code 417)

Class X · Part C: Practical Work

Session _____

Name: _____

Class & Section: _____

Roll No.: _____

Submitted to: _____

CERTIFICATE

This is to certify that _____ of Class X, Roll No. _____, has completed this practical file of Artificial Intelligence (Subject Code 417) during the session _____, as prescribed in Part C of the curriculum of the Central Board of Secondary Education.

_____ _____

Teacher-in-charge External Examiner

Date: _____

INDEX

S. NO.	PROGRAM	PAGE NO.	DATE	TEACHER'S SIGN
1	Add the Elements of Two Lists	4		
2	Mean, Median and Mode with NumPy	5		
3	Variance, Standard Deviation and Outliers	6		
4	A Line Chart from (2, 5) to (9, 10)	8		
5	A Scatter Chart of Five Points	9		
6	Read a CSV File and Display 10 Rows	11		
7	Read a CSV File and Display Its Information	12		
8	Read an Image and Display It	14		
9	Read an Image and Identify Its Shape	16		
10	Grayscale and RGB Images	17		
11	The Convolution Operator	19		
12	Text Normalisation	22		
13	Bag of Words	24		
14	TFIDF	26		
15	Confusion Matrix and Evaluation Metrics	28		

Page numbers count from the cover page. Programs 6 and 7 use penguins.csv (Palmer Penguins, CC0 licence); programs 8 to 11 use photo.jpg (public-domain photograph of Grace Hopper from the Matplotlib sample data). All fifteen programs and both files are in ai-practical-file.zip at lambdalab.in.

PROGRAM 1

Add the Elements of Two Lists

AIM

Write a program to add the elements of the two lists.

PROGRAM

add_lists.py

```
# Program 1: Add the elements of two lists
# Each element of the first list is added to the element in the same
# position of the second list.

list1 = [10, 20, 30, 40, 50]
list2 = [5, 15, 25, 35, 45]

result = []
for i in range(len(list1)):
    result.append(list1[i] + list2[i])

print("List 1 :", list1)
print("List 2 :", list2)
print("Sum    :", result)
print()

# Careful: the + sign does NOT add two lists element by element.
# It joins them into one longer list.
print("list1 + list2 gives :", list1 + list2)
print()

# With NumPy arrays, + does add element by element.
import numpy as np
a = np.array(list1)
b = np.array(list2)
print("Using NumPy arrays  :", a + b)
```

OUTPUT

```
List 1 : [10, 20, 30, 40, 50]
List 2 : [5, 15, 25, 35, 45]
Sum    : [15, 35, 55, 75, 95]

list1 + list2 gives : [10, 20, 30, 40, 50, 5, 15, 25, 35, 45]

Using NumPy arrays  : [15 35 55 75 95]
```

PROGRAM 2

Mean, Median and Mode with NumPy

AIM

Write a program to calculate mean, median and mode using NumPy.

PROGRAM

mean_median_mode.py

```
# Program 2: Calculate mean, median and mode using NumPy
# Marks scored by 15 students in an AI unit test (out of 20).

import numpy as np

marks = np.array([12, 15, 18, 15, 9, 20, 15, 11, 18, 14, 16, 15, 7, 18, 13])
print("Marks :", marks)
print("Number of students :", len(marks))
print()

mean = np.mean(marks)
median = np.median(marks)

# NumPy has no mode() function. np.unique() lists each different value
# once, and return_counts=True says how many times each one occurs.
values, counts = np.unique(marks, return_counts=True)
mode = values[np.argmax(counts)]

print("Mean   :", round(mean, 2))
print("Median :", median)
print("Mode   :", mode)
print()
print("How often each mark occurs:")
for i in range(len(values)):
    print("  ", values[i], ":", counts[i], "time(s)")
```

OUTPUT

```
Marks : [12 15 18 15  9 20 15 11 18 14 16 15  7 18 13]
Number of students : 15

Mean   : 14.4
Median : 15.0
Mode   : 15

How often each mark occurs:
   7 : 1 time(s)
   9 : 1 time(s)
   11 : 1 time(s)
   12 : 1 time(s)
   13 : 1 time(s)
   14 : 1 time(s)
   15 : 4 time(s)
   16 : 1 time(s)
   18 : 3 time(s)
   20 : 1 time(s)
```

PROGRAM 3

Variance, Standard Deviation and Outliers

AIM

Write a program to find the variance and standard deviation of a set of values, detect an outlier, and show the distribution as a histogram.

PROGRAM

variance_outliers.py

```
# Program 3: Variance, standard deviation and outliers
# Minutes taken by 20 students to finish a quiz. One student left the
# quiz open and came back much later - an outlier.

import numpy as np
import matplotlib.pyplot as plt

minutes = np.array([12, 14, 11, 15, 13, 12, 16, 14, 13, 12,
                    15, 11, 14, 13, 58, 12, 15, 13, 14, 12])

mean = np.mean(minutes)
variance = np.var(minutes)
sd = np.std(minutes)
print("Mean               :", round(mean, 2), "minutes")
print("Variance           :", round(variance, 2))
print("Standard deviation :", round(sd, 2), "minutes")
print()

# An outlier lies an abnormal distance from the other values. A common
# rule: more than 2 standard deviations away from the mean.
low = mean - 2 * sd
high = mean + 2 * sd
print("Normal range       :", round(low, 2), "to", round(high, 2), "minutes")
outliers = minutes[(minutes < low) | (minutes > high)]
print("Outliers           :", outliers)
print()

# Without the outlier, the mean and standard deviation tell the truth.
clean = minutes[(minutes >= low) & (minutes <= high)]
print("Without the outlier:")
print("Mean               :", round(np.mean(clean), 2), "minutes")
print("Standard deviation :", round(np.std(clean), 2), "minutes")

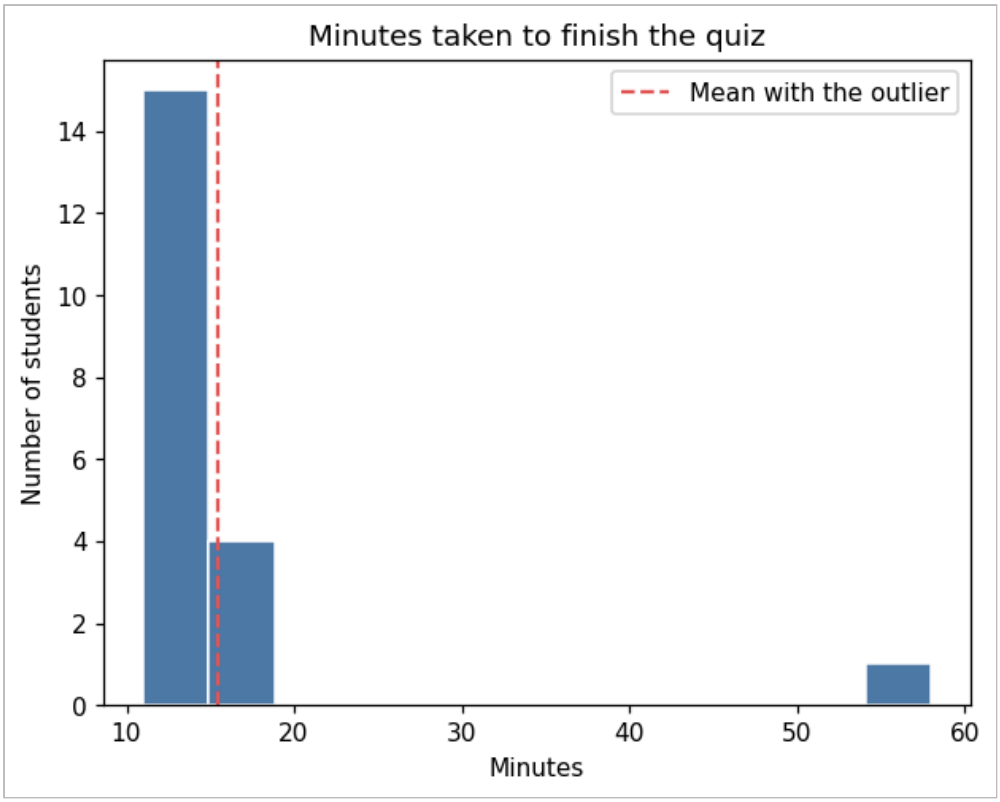
# The distribution of the times, as a histogram.
plt.hist(minutes, bins=12, color="#4e79a7", edgecolor="white")
plt.axvline(mean, color="#e15759", linestyle="--", label="Mean with the outlier")
plt.title("Minutes taken to finish the quiz")
plt.xlabel("Minutes")
plt.ylabel("Number of students")
plt.legend()
plt.show()
```

OUTPUT

```
Mean               : 15.45 minutes
Variance           : 97.15
Standard deviation : 9.86 minutes

Normal range       : -4.26 to 35.16 minutes
Outliers           : [58]
```

```
Without the outlier:
Mean               : 13.21 minutes
Standard deviation : 1.4 minutes
```



Output figure — shown by plt.show()

PROGRAM 4

A Line Chart from (2, 5) to (9, 10)

AIM

Write a program to display a line chart from (2, 5) to (9, 10).

PROGRAM

`line_chart.py`

```
# Program 4: Display a line chart from (2, 5) to (9, 10)

import matplotlib.pyplot as plt

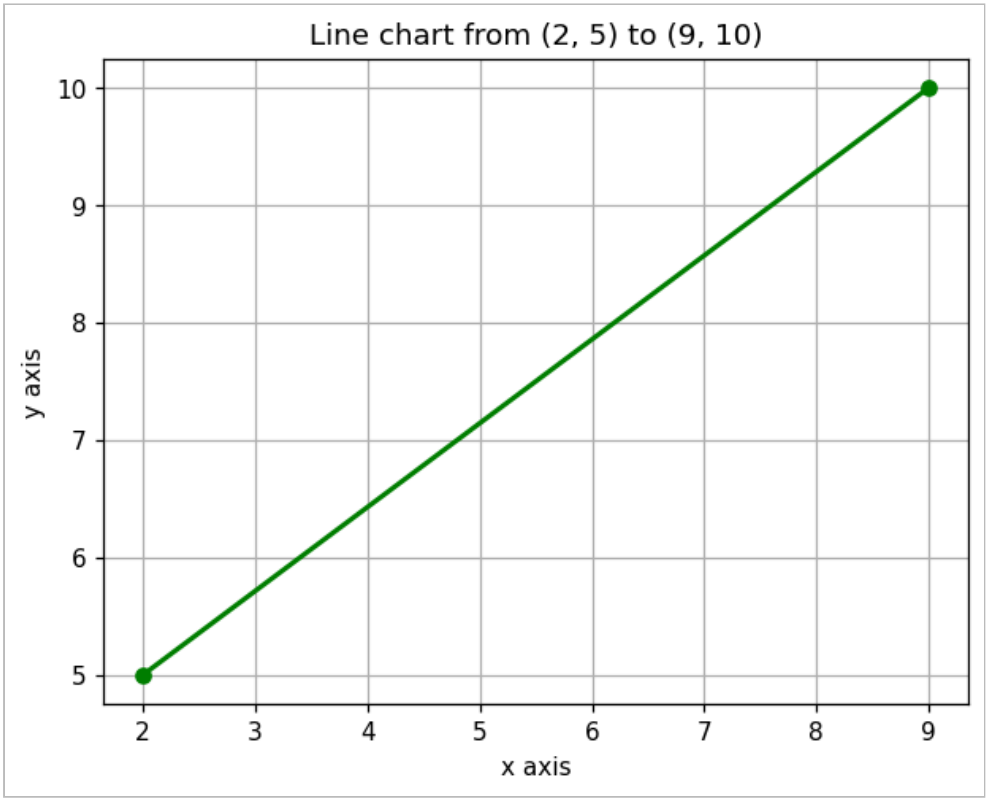
x = [2, 9]       # x values of the two points
y = [5, 10]      # y values of the two points

plt.plot(x, y, color="green", marker="o", linewidth=2)
plt.title("Line chart from (2, 5) to (9, 10)")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.grid(True)
plt.show()

# The slope of the line: how much y rises for each step in x.
slope = (y[1] - y[0]) / (x[1] - x[0])
print("The line rises", round(slope, 3), "in y for every 1 in x")
```

OUTPUT

```
The line rises 0.714 in y for every 1 in x
```



Output figure — shown by plt.show()

PROGRAM 5

A Scatter Chart of Five Points

AIM

Write a program to display a scatter chart for the following points: (2, 5), (9, 10), (8, 3), (5, 7), (6, 18).

PROGRAM

scatter_chart.py

```
# Program 5: Display a scatter chart for the points
# (2, 5), (9, 10), (8, 3), (5, 7), (6, 18)

import matplotlib.pyplot as plt

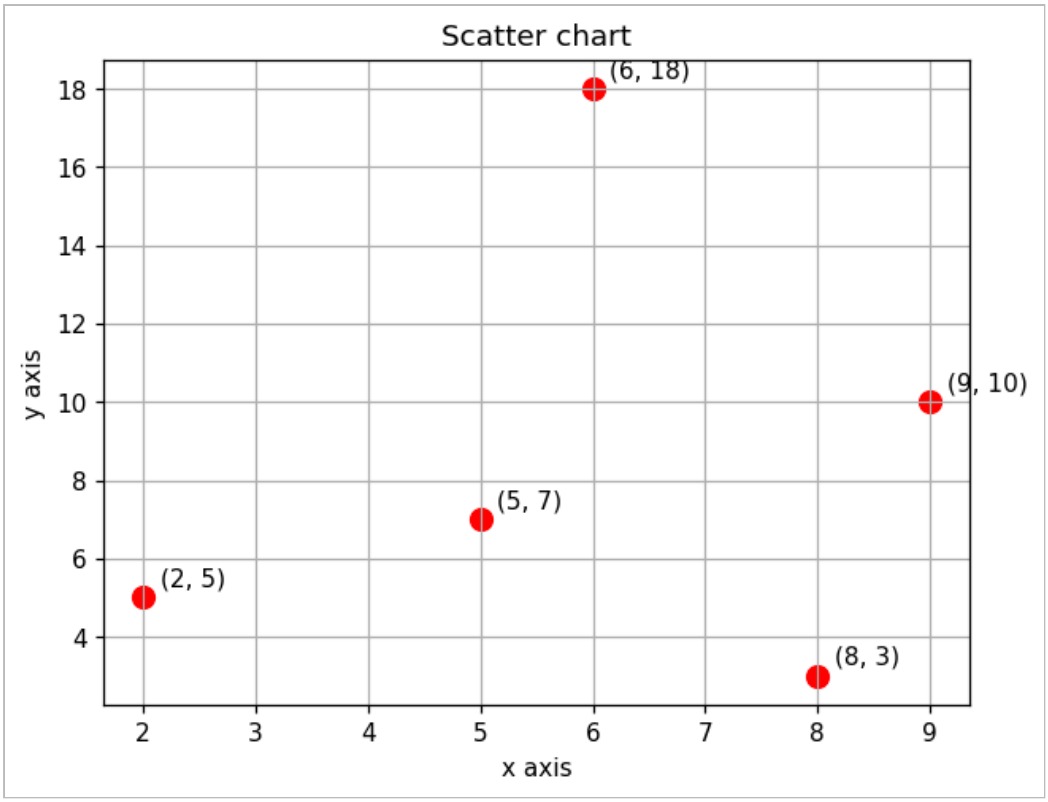
x = [2, 9, 8, 5, 6]
y = [5, 10, 3, 7, 18]

plt.scatter(x, y, color="red", s=80)
for i in range(len(x)):
    # Write each point's coordinates beside it.
    plt.text(x[i] + 0.15, y[i] + 0.3, "(" + str(x[i]) + ", " + str(y[i]) + ")")
plt.title("Scatter chart")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.grid(True)
plt.show()

print("Points plotted:", len(x))
print("Highest point :", "(" + str(x[y.index(max(y))]) + ", " + str(max(y)) + ")")
```

OUTPUT

```
Points plotted: 5
Highest point : (6, 18)
```



Output figure — shown by plt.show()

PROGRAM 6

Read a CSV File and Display 10 Rows

AIM

Read the CSV file saved in your system and display 10 rows.

Files used: penguins.csv (kept in the same folder as the program)

PROGRAM

csv_rows.py

```
# Program 6: Read a CSV file saved in your system and display 10 rows
# penguins.csv is the Palmer Penguins dataset used in the Statistical
# Data unit. Keep it in the same folder as this program.

import pandas as pd

# Show every column instead of "..." in the middle of wide tables.
pd.set_option("display.max_columns", None)
pd.set_option("display.width", 120)

data = pd.read_csv("penguins.csv")

print("The first 10 rows:")
print(data.head(10))
print()
print("The last 3 rows:")
print(data.tail(3))
```

OUTPUT

```
The first 10 rows:
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g     sex  year
0  Adelie  Torgersen            39.1           18.7              181.0       3750.0    male  2007
1  Adelie  Torgersen            39.5           17.4              186.0       3800.0  female  2007
2  Adelie  Torgersen            40.3           18.0              195.0       3250.0  female  2007
3  Adelie  Torgersen             NaN            NaN                NaN          NaN     NaN  2007
4  Adelie  Torgersen            36.7           19.3              193.0       3450.0  female  2007
5  Adelie  Torgersen            39.3           20.6              190.0       3650.0    male  2007
6  Adelie  Torgersen            38.9           17.8              181.0       3625.0  female  2007
7  Adelie  Torgersen            39.2           19.6              195.0       4675.0    male  2007
8  Adelie  Torgersen            34.1           18.1              193.0       3475.0     NaN  2007
9  Adelie  Torgersen            42.0           20.2              190.0       4250.0     NaN  2007

The last 3 rows:
       species island  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g     sex  year
341  Chinstrap  Dream            49.6           18.2              193.0       3775.0    male  2009
342  Chinstrap  Dream            50.8           19.0              210.0       4100.0    male  2009
343  Chinstrap  Dream            50.2           18.7              198.0       3775.0  female  2009
```

PROGRAM 7

Read a CSV File and Display Its Information

AIM

Read the CSV file saved in your system and display its information.

Files used: penguins.csv (kept in the same folder as the program)

PROGRAM

csv_info.py

```
# Program 7: Read a CSV file saved in your system and display its information

import pandas as pd

pd.set_option("display.max_columns", None)
pd.set_option("display.width", 120)

data = pd.read_csv("penguins.csv")

print("Rows and columns :", data.shape)
print()
print("Information about the data:")
data.info()
print()
print("Missing values in each column:")
print(data.isnull().sum())
print()
print("Summary of the number columns:")
print(data.describe().round(1))
```

OUTPUT

```
Rows and columns : (344, 8)

Information about the data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
 7   year               344 non-null    int64
dtypes: float64(4), int64(1), object(3)
memory usage: 21.6+ KB

Missing values in each column:
species              0
island               0
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
sex                 11
year                 0
dtype: int64
```

```
Summary of the number columns:
       bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g    year
count           342.0          342.0              342.0        342.0   344.0
mean             43.9           17.2              200.9       4201.8  2008.0
std               5.5            2.0               14.1        802.0     0.8
min              32.1           13.1              172.0       2700.0  2007.0
25%              39.2           15.6              190.0       3550.0  2007.0
50%              44.4           17.3              197.0       4050.0  2008.0
75%              48.5           18.7              213.0       4750.0  2009.0
max              59.6           21.5              231.0       6300.0  2009.0
```

PROGRAM 8

Read an Image and Display It

AIM

Write a program to read an image and display it using Python.

Files used: photo.jpg (kept in the same folder as the program)

PROGRAM

read_image.py

```
# Program 8: Read an image and display it using Python
# photo.jpg must be in the same folder as this program.

import cv2
import matplotlib.pyplot as plt

image = cv2.imread("photo.jpg")

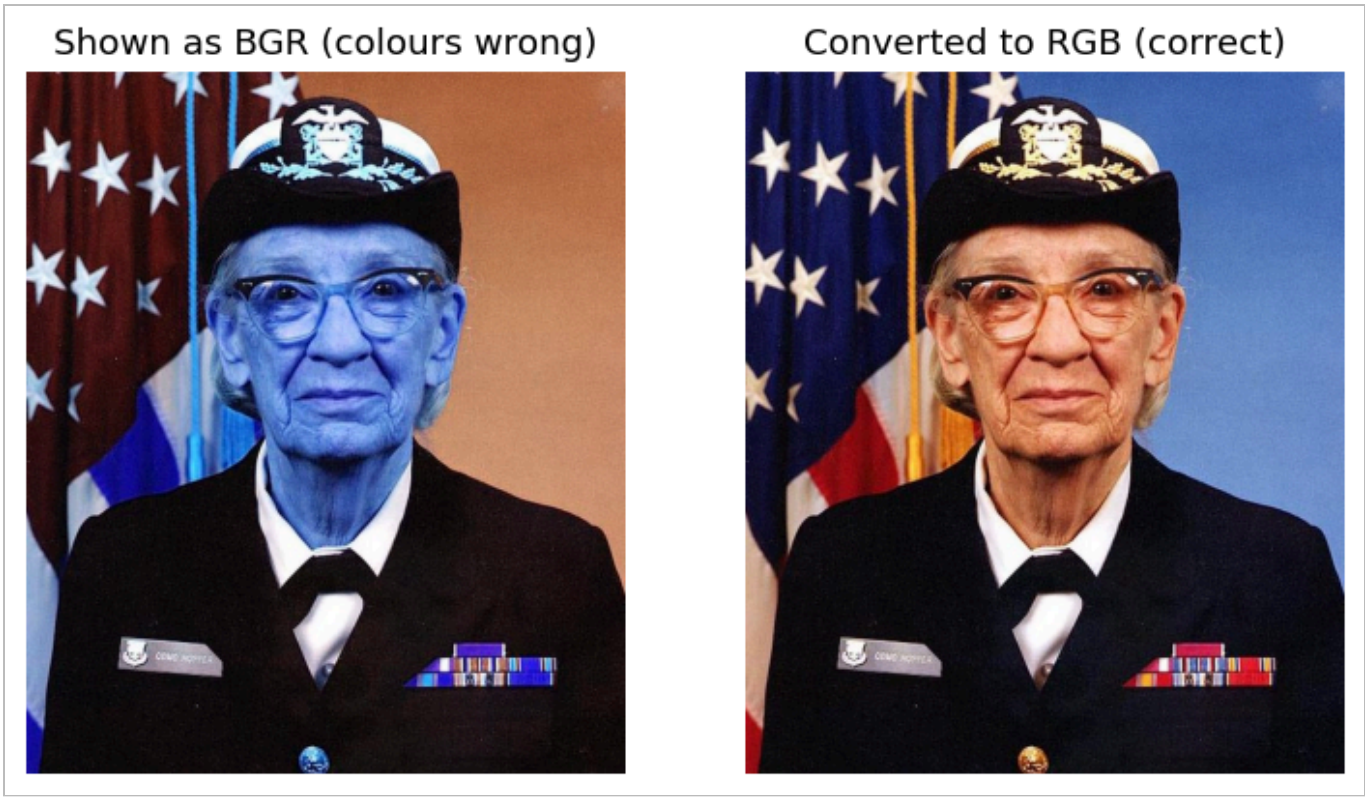
# OpenCV stores colours in the order Blue, Green, Red (BGR).
# Matplotlib expects Red, Green, Blue (RGB), so the order is swapped.
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(8, 4.5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Shown as BGR (colours wrong)")
plt.axis("off")
plt.subplot(1, 2, 2)
plt.imshow(image_rgb)
plt.title("Converted to RGB (correct)")
plt.axis("off")
plt.show()

print("Image read successfully")
```

OUTPUT

```
Image read successfully
```



Output figure — shown by plt.show()

PROGRAM 9

Read an Image and Identify Its Shape

AIM

Write a program to read an image and identify its shape using Python.

Files used: photo.jpg (kept in the same folder as the program)

PROGRAM

image_shape.py

```
# Program 9: Read an image and identify its shape using Python
# The "shape" of an image is its size as an array of numbers:
# height (rows of pixels) x width (columns of pixels) x colour channels.

import cv2

image = cv2.imread("photo.jpg")
print("Shape of the colour image :", image.shape)

height, width, channels = image.shape
print("Height   :", height, "pixels")
print("Width    :", width, "pixels")
print("Channels :", channels, "(Blue, Green, Red)")
print("Total pixels         :", height * width)
print("Total numbers stored :", image.size)
print()

grey = cv2.imread("photo.jpg", cv2.IMREAD_GRAYSCALE)
print("Shape of the grayscale image :", grey.shape)
print("A grayscale image has no third number: one value per pixel.")
print()

if height > width:
    print("This image is in portrait shape (taller than wide).")
elif width > height:
    print("This image is in landscape shape (wider than tall).")
else:
    print("This image is square.")
```

OUTPUT

```
Shape of the colour image : (600, 512, 3)
Height   : 600 pixels
Width    : 512 pixels
Channels : 3 (Blue, Green, Red)
Total pixels         : 307200
Total numbers stored : 921600

Shape of the grayscale image : (600, 512)
A grayscale image has no third number: one value per pixel.

This image is in portrait shape (taller than wide).
```

PROGRAM 10

Grayscale and RGB Images

AIM

Write a program to convert a colour image to grayscale, display the pixel value at a point, and split the image into its red, green and blue channels.

Files used: photo.jpg (kept in the same folder as the program)

PROGRAM

grayscale_rgb.py

```
# Program 10: Grayscale and RGB images - pixel values and colour channels

import cv2
import matplotlib.pyplot as plt

image = cv2.imread("photo.jpg")
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# One pixel: a colour pixel has three values, a grey pixel has one.
row, col = 150, 250
print("Pixel at row", row, "column", col)
print("  in the colour image (R, G, B) :", rgb[row, col])
print("  in the grayscale image        :", grey[row, col])
print()
print("Pixel values run from", grey.min(), "(black) to", grey.max(), "(white)")
print("Data type of each value :", grey.dtype, "(whole numbers 0 to 255)")
print()

# Split the colour image into its three channels.
red = rgb[:, :, 0]
green = rgb[:, :, 1]
blue = rgb[:, :, 2]
print("Average red value   :", round(red.mean(), 1))
print("Average green value :", round(green.mean(), 1))
print("Average blue value  :", round(blue.mean(), 1))

# To see one channel, keep it and set the other two channels to 0.
only_red = rgb.copy()
only_red[:, :, 1] = 0
only_red[:, :, 2] = 0
only_green = rgb.copy()
only_green[:, :, 0] = 0
only_green[:, :, 2] = 0
only_blue = rgb.copy()
only_blue[:, :, 0] = 0
only_blue[:, :, 1] = 0

plt.figure(figsize=(10, 3.6))
pictures = [rgb, grey, only_red, only_green, only_blue]
titles = ["Colour (RGB)", "Grayscale", "Red channel", "Green channel", "Blue channel"]
for i in range(5):
    plt.subplot(1, 5, i + 1)
    if i == 1:
        plt.imshow(pictures[i], cmap="gray")
    else:
        plt.imshow(pictures[i])
    plt.title(titles[i], fontsize=9)
```

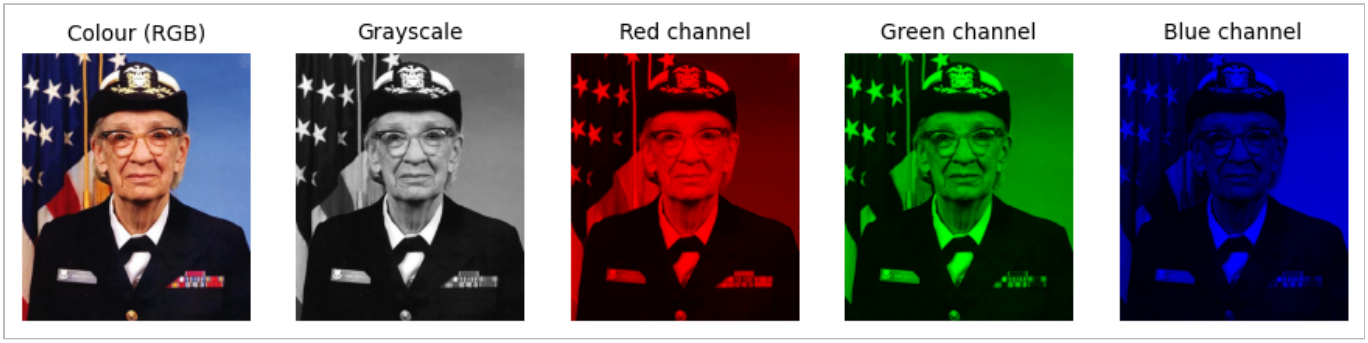
```
    plt.axis("off")
plt.show()
```

OUTPUT

```
Pixel at row 150 column 250
  in the colour image (R, G, B) : [252 192 164]
  in the grayscale image        : 207

Pixel values run from 0 (black) to 255 (white)
Data type of each value : uint8 (whole numbers 0 to 255)

Average red value   : 82.5
Average green value : 72.4
Average blue value  : 86.4
```



Output figure — shown by plt.show()

PROGRAM 11

The Convolution Operator

AIM

Write a program to perform convolution of an image array with a kernel — the worked example from the CBSE AI handbook — and apply an edge-detecting kernel to a photo.

Files used: photo.jpg (kept in the same folder as the program)

PROGRAM

convolution.py

```
# Program 11: The convolution operator
# Part 1 works the example from the CBSE AI handbook: a 5 x 5 image array
# and a 3 x 3 kernel. The edges of the image are extended with zeros, so
# the result is the same size as the image.
# Part 2 applies an edge-finding kernel to a real photo.

import numpy as np
import cv2
import matplotlib.pyplot as plt

image = np.array([[17, 24,  1,  8, 15],
                  [23,  5,  7, 14, 16],
                  [ 4,  6, 13, 20, 22],
                  [10, 12, 19, 21,  3],
                  [11, 18, 25,  2,  9]])

# The kernel, already rotated as in the handbook.
kernel = np.array([[2, 9, 4],
                   [7, 5, 3],
                   [6, 1, 8]])

# Extend every edge by one pixel of value 0.
padded = np.pad(image, 1)

result = np.zeros((5, 5), dtype=int)
for r in range(5):
    for c in range(5):
        # The 3 x 3 patch of the image under the kernel, centred on (r, c).
        patch = padded[r:r + 3, c:c + 3]
        # Multiply matching numbers and add all nine products.
        result[r, c] = (patch * kernel).sum()

print("Image array I:")
print(image)
print()
print("Kernel K:")
print(kernel)
print()
print("First value, worked out by hand:")
print("  centre 5 x 17 + right 3 x 24 + below 1 x 23 + below-right 8 x 5")
print("  =", 5 * 17, "+", 3 * 24, "+", 1 * 23, "+", 8 * 5, "=", 5 * 17 + 3 * 24 + 1 * 23 + 8 * 5)
print()
print("Result I * K:")
print(result)
print()

# Check: OpenCV's filter2D slides a kernel the same way.
check = cv2.filter2D(image.astype(np.float32), -1, kernel.astype(np.float32),
                     borderType=cv2.BORDER_CONSTANT)
```

```
print("Same answer from OpenCV filter2D :", np.array_equal(result, check.astype(int)))

# Part 2: an edge kernel on the photo.
grey = cv2.imread("photo.jpg", cv2.IMREAD_GRAYSCALE)
edge_kernel = np.array([[-1, -1, -1],
                        [-1,  8, -1],
                        [-1, -1, -1]], dtype=np.float32)
edges = cv2.filter2D(grey.astype(np.float32), -1, edge_kernel)

plt.figure(figsize=(8, 4.5))
plt.subplot(1, 2, 1)
plt.imshow(grey, cmap="gray")
plt.title("Grayscale photo")
plt.axis("off")
plt.subplot(1, 2, 2)
plt.imshow(np.abs(edges), cmap="gray", vmin=0, vmax=120)
plt.title("After the edge kernel")
plt.axis("off")
plt.show()
```

OUTPUT

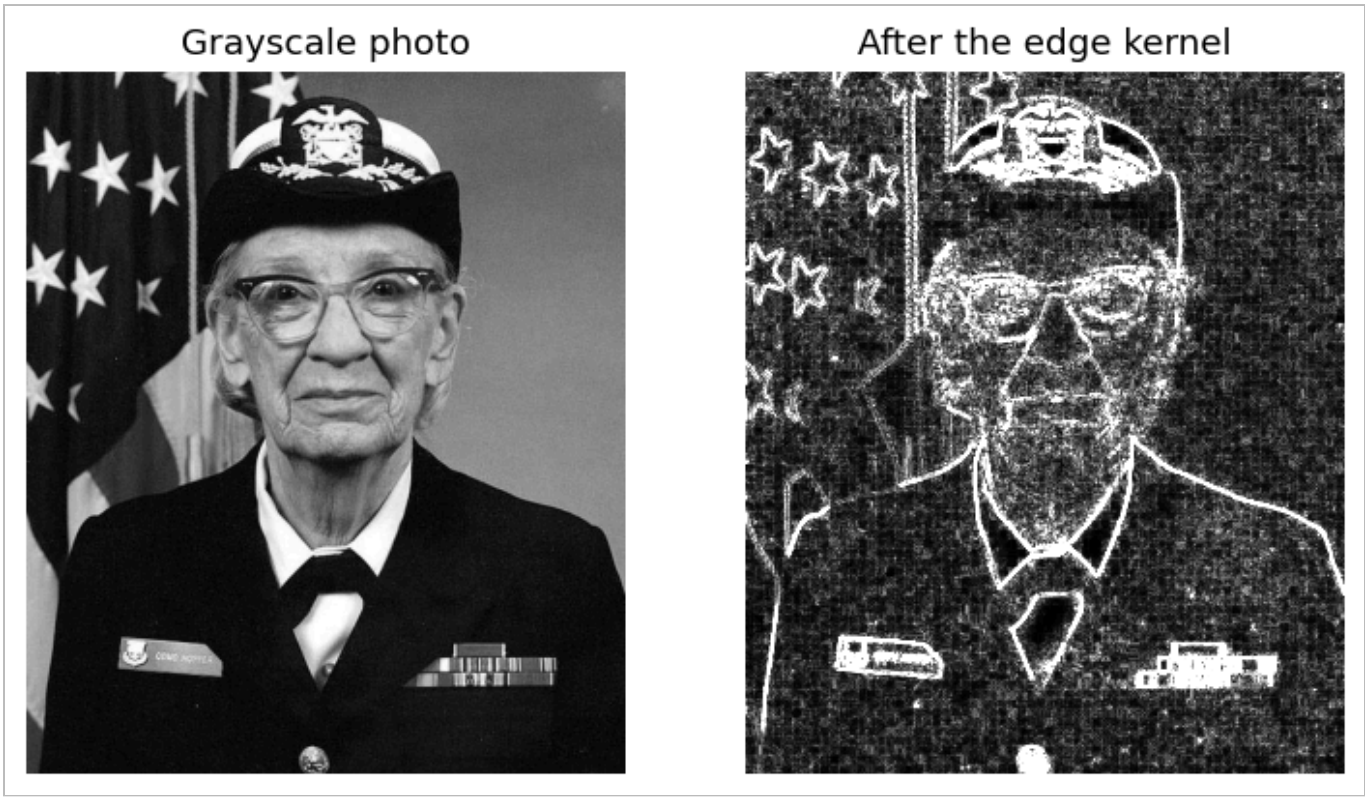
```
Image array I:
[[17 24  1  8 15]
 [23  5  7 14 16]
 [ 4  6 13 20 22]
 [10 12 19 21  3]
 [11 18 25  2  9]]

Kernel K:
[[2 9 4]
 [7 5 3]
 [6 1 8]]

First value, worked out by hand:
  centre 5 x 17 + right 3 x 24 + below 1 x 23 + below-right 8 x 5
  = 85 + 72 + 23 + 40 = 220

Result I * K:
[[220 441 346 276 231]
 [431 595 410 575 471]
 [371 440 555 620 551]
 [301 585 600 765 421]
 [247 446 536 451 128]]

Same answer from OpenCV filter2D : True
```



Output figure — shown by plt.show()

PROGRAM 12

Text Normalisation

AIM

Write a program to perform text normalisation on a corpus: sentence segmentation, tokenisation, removing stop words, special characters and numbers, converting to lower case, and stemming.

PROGRAM

text_normalisation.py

```
# Program 12: Text normalisation - the steps from the CBSE AI handbook
#   1. Sentence segmentation   2. Tokenisation
#   3. Removing stop words, special characters and numbers
#   4. Converting to a common case   5. Stemming

corpus = "Raj and Vijay are best friends. They play together with other friends! Raj likes to play
football but Vijay prefers to play online games. Vijay wants to be an online gamer in 2030."

print("Corpus:")
print(corpus)
print()

# 1. Sentence segmentation: split wherever a sentence ends.
text = corpus.replace("!", ".").replace("?", ".")
sentences = []
for s in text.split("."):
    if s.strip() != "":
        sentences.append(s.strip())
print("1. Sentences:")
for s in sentences:
    print("  ", s)
print()

# 2. Tokenisation: split each sentence into words (tokens).
tokens = []
for s in sentences:
    for word in s.split():
        tokens.append(word)
print("2. Tokens (" + str(len(tokens)) + "):")
print(tokens)
print()

# 3. Remove stop words, special characters and numbers.
stop_words = ["and", "are", "the", "a", "an", "to", "is", "in", "they", "with",
              "but", "be", "of", "other"]
kept = []
for word in tokens:
    if word.lower() in stop_words or not word.isalpha():
        continue
    kept.append(word)
print("3. After removing stop words and numbers (" + str(len(kept)) + "):")
print(kept)
print()

# 4. Convert to a common case (lower case).
lower = []
for word in kept:
    lower.append(word.lower())
print("4. Lower case:")
print(lower)
```

```
print()

# 5. Stemming: remove the affixes from the end of each word.
affixes = ["ing", "ed", "er", "es", "s"]
stems = []
for word in lower:
    for a in affixes:
        if word.endswith(a) and len(word) > len(a) + 2:
            word = word[:-len(a)]
            break
    stems.append(word)
print("5. After stemming:")
print(stems)
print()

# The handbook's own stemming table, worked by the same rule.
print("Word        Stem")
for word in ["healed", "healing", "healer", "studies", "studying"]:
    stem = word
    for a in affixes:
        if word.endswith(a) and len(word) > len(a) + 2:
            stem = word[:-len(a)]
            break
    print(word.ljust(11), stem)
```

OUTPUT

```
Corpus:
Raj and Vijay are best friends. They play together with other friends! Raj likes to play football but
Vijay prefers to play online games. Vijay wants to be an online gamer in 2030.

1. Sentences:
   Raj and Vijay are best friends
   They play together with other friends
   Raj likes to play football but Vijay prefers to play online games
   Vijay wants to be an online gamer in 2030

2. Tokens (33):
['Raj', 'and', 'Vijay', 'are', 'best', 'friends', 'They', 'play', 'together', 'with', 'other',
'friends', 'Raj', 'likes', 'to', 'play', 'football', 'but', 'Vijay', 'prefers', 'to', 'play',
'online', 'games', 'Vijay', 'wants', 'to', 'be', 'an', 'online', 'gamer', 'in', '2030']

3. After removing stop words and numbers (20):
['Raj', 'Vijay', 'best', 'friends', 'play', 'together', 'friends', 'Raj', 'likes', 'play', 'football',
'Vijay', 'prefers', 'play', 'online', 'games', 'Vijay', 'wants', 'online', 'gamer']

4. Lower case:
['raj', 'vijay', 'best', 'friends', 'play', 'together', 'friends', 'raj', 'likes', 'play', 'football',
'vijay', 'prefers', 'play', 'online', 'games', 'vijay', 'wants', 'online', 'gamer']

5. After stemming:
['raj', 'vijay', 'best', 'friend', 'play', 'togeth', 'friend', 'raj', 'lik', 'play', 'football',
'vijay', 'prefer', 'play', 'online', 'gam', 'vijay', 'want', 'online', 'gam']

Word        Stem
healed      heal
healing     heal
healer      heal
studies     studi
studying    study
```

PROGRAM 13

Bag of Words

AIM

Write a program to create the dictionary and the document vector table for a corpus using the Bag of Words algorithm.

PROGRAM

bag_of_words.py

```
# Program 13: Bag of Words
# The three documents used in the CBSE AI handbook.

import pandas as pd

documents = ["Aman and Avni are stressed",
             "Aman went to a therapist",
             "Avni went to download a health chatbot"]

# Step 1: text normalisation - lower case, split into tokens.
# (As in the handbook, stop words are kept so the table matches it.)
tokens = []
for d in documents:
    tokens.append(d.lower().split())
for i in range(len(tokens)):
    print("Document", i + 1, ":", tokens[i])
print()

# Step 2: create the dictionary - every different word, in the order met.
dictionary = []
for t in tokens:
    for word in t:
        if word not in dictionary:
            dictionary.append(word)
print("Dictionary (" + str(len(dictionary)) + " words):")
print(dictionary)
print()

# Step 3: document vectors - how many times each dictionary word occurs.
vectors = []
for t in tokens:
    row = []
    for word in dictionary:
        row.append(t.count(word))
    vectors.append(row)

table = pd.DataFrame(vectors, columns=dictionary, index=["Doc 1", "Doc 2", "Doc 3"])
pd.set_option("display.width", 160)
pd.set_option("display.max_columns", None)
print("Document vector table:")
print(table)
print()

# Step 4: how often each word occurs in the whole corpus.
print("Occurrences in the whole corpus:")
print(table.sum().sort_values(ascending=False).to_string())
```

OUTPUT

```
Document 1 : ['aman', 'and', 'avni', 'are', 'stressed']
Document 2 : ['aman', 'went', 'to', 'a', 'therapist']
Document 3 : ['avni', 'went', 'to', 'download', 'a', 'health', 'chatbot']

Dictionary (12 words):
['aman', 'and', 'avni', 'are', 'stressed', 'went', 'to', 'a', 'therapist', 'download', 'health',
'chatbot']

Document vector table:
       aman  and  avni  are  stressed  went  to  a  therapist  download  health  chatbot
Doc 1     1    1     1    1         1     0   0  0          0         0       0        0
Doc 2     1    0     0    0         0     1   1  1          1         0       0        0
Doc 3     0    0     1    0         0     1   1  1          0         1       1        1

Occurrences in the whole corpus:
aman         2
avni         2
went         2
to           2
a            2
and          1
are          1
stressed     1
therapist    1
download     1
health       1
chatbot      1
```

PROGRAM 14

TFIDF

AIM

Write a program to calculate term frequency, document frequency, inverse document frequency and TFIDF for the words of a corpus.

PROGRAM

tfidf.py

```
# Program 14: TFIDF - Term Frequency and Inverse Document Frequency
#   TF(W)    = number of times word W occurs in a document
#   DF(W)    = number of documents in which W occurs
#   IDF(W)   = N / DF(W), where N is the number of documents
#   TFIDF(W) = TF(W) x log(IDF(W)), with log to the base 10

import numpy as np
import pandas as pd

documents = ["Aman and Avni are stressed",
             "Aman went to a therapist",
             "Avni went to download a health chatbot"]

tokens = []
for d in documents:
    tokens.append(d.lower().split())
dictionary = []
for t in tokens:
    for word in t:
        if word not in dictionary:
            dictionary.append(word)

pd.set_option("display.width", 160)
pd.set_option("display.max_columns", None)

# Term frequency: the document vector table from Bag of Words.
tf = []
for t in tokens:
    row = []
    for word in dictionary:
        row.append(t.count(word))
    tf.append(row)
tf = pd.DataFrame(tf, columns=dictionary, index=["Doc 1", "Doc 2", "Doc 3"])
print("Term frequency (TF):")
print(tf)
print()

N = len(documents)
df = (tf > 0).sum()
print("Document frequency (DF):")
print(df.to_frame().T.to_string(index=False))
print()

idf = N / df
print("Inverse document frequency (IDF = N / DF), N =", N)
print(idf.round(2).to_frame().T.to_string(index=False))
print()

tfidf = tf * np.log10(idf)
print("TFIDF = TF x log10(IDF):")
```

```
print(tfidf.round(3))
print()

print("Highest TFIDF in each document:")
for i in range(N):
    row = tfidf.iloc[i]
    best = row.max()
    words = list(row[row == best].index)
    print("  Doc", i + 1, ":", ", ".join(words), "(" + str(round(best, 3)) + ")")
print()

# The handbook's two worked examples, for a corpus of 10 documents.
print("'and' in 10 of 10 documents      : log10(10/10) =", round(np.log10(10 / 10), 3))
print("'pollution' in 3 of 10 documents : log10(10/3)  =", round(np.log10(10 / 3), 3))
```

OUTPUT

```
Term frequency (TF):
       aman  and  avni  are  stressed  went  to  a  therapist  download  health  chatbot
Doc 1     1    1     1    1         1     0   0  0          0         0       0        0
Doc 2     1    0     0    0         0     1   1  1          1         0       0        0
Doc 3     0    0     1    0         0     1   1  1          0         1       1        1

Document frequency (DF):
 aman  and  avni  are  stressed  went  to  a  therapist  download  health  chatbot
    2    1     2    1         1     2   2  2          1         1       1        1

Inverse document frequency (IDF = N / DF), N = 3
 aman  and  avni  are  stressed  went  to   a  therapist  download  health  chatbot
  1.5  3.0   1.5  3.0       3.0   1.5 1.5 1.5        3.0       3.0     3.0      3.0

TFIDF = TF x log10(IDF):
        aman    and   avni    are  stressed   went     to      a  therapist  download  health  chatbot
Doc 1  0.176  0.477  0.176  0.477     0.477  0.000  0.000  0.000      0.000     0.000   0.000    0.000
Doc 2  0.176  0.000  0.000  0.000     0.000  0.176  0.176  0.176      0.477     0.000   0.000    0.000
Doc 3  0.000  0.000  0.176  0.000     0.000  0.176  0.176  0.176      0.000     0.477   0.477    0.477

Highest TFIDF in each document:
  Doc 1 : and, are, stressed (0.477)
  Doc 2 : therapist (0.477)
  Doc 3 : download, health, chatbot (0.477)

'and' in 10 of 10 documents      : log10(10/10) = 0.0
'pollution' in 3 of 10 documents : log10(10/3)  = 0.523
```

PROGRAM 15

Confusion Matrix and Evaluation Metrics

AIM

Write a program to construct the confusion matrix for a classification model and calculate its accuracy, precision, recall and F1 score.

PROGRAM

confusion_matrix.py

```
# Program 15: Confusion matrix, accuracy, precision, recall and F1 score
# The activity from the CBSE AI handbook: a model is tested on 1000
# students - 900 actually passed (Yes) and 100 did not (No). A faulty
# model predicts "Yes" for everybody. Then a better model is tested.

import matplotlib.pyplot as plt

actual = ["Yes"] * 900 + ["No"] * 100
faulty = ["Yes"] * 1000
# A better model: right for most students, wrong for a few.
better = ["Yes"] * 870 + ["No"] * 30 + ["No"] * 85 + ["Yes"] * 15

names = ["Faulty model (always Yes)", "Better model"]
for k in range(2):
    predicted = [faulty, better][k]
    tp = tn = fp = fn = 0
    for i in range(len(actual)):
        if actual[i] == "Yes" and predicted[i] == "Yes":
            tp = tp + 1
        elif actual[i] == "No" and predicted[i] == "No":
            tn = tn + 1
        elif actual[i] == "No" and predicted[i] == "Yes":
            fp = fp + 1
        else:
            fn = fn + 1
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1 = 2 * precision * recall / (precision + recall)
    # For the "No" class: how many of the students who failed were found?
    no_found = tn / (tn + fp)

    print(names[k])
    print("  TP =", tp, " FN =", fn, " FP =", fp, " TN =", tn)
    print("  Accuracy  =", round(accuracy, 3), "=", round(accuracy * 100, 1), "%")
    print("  Precision =", round(precision, 3))
    print("  Recall    =", round(recall, 3))
    print("  F1 score  =", round(f1, 3))
    print("  Students who did not pass, found by the model:", tn, "of", tn + fp)
    print()

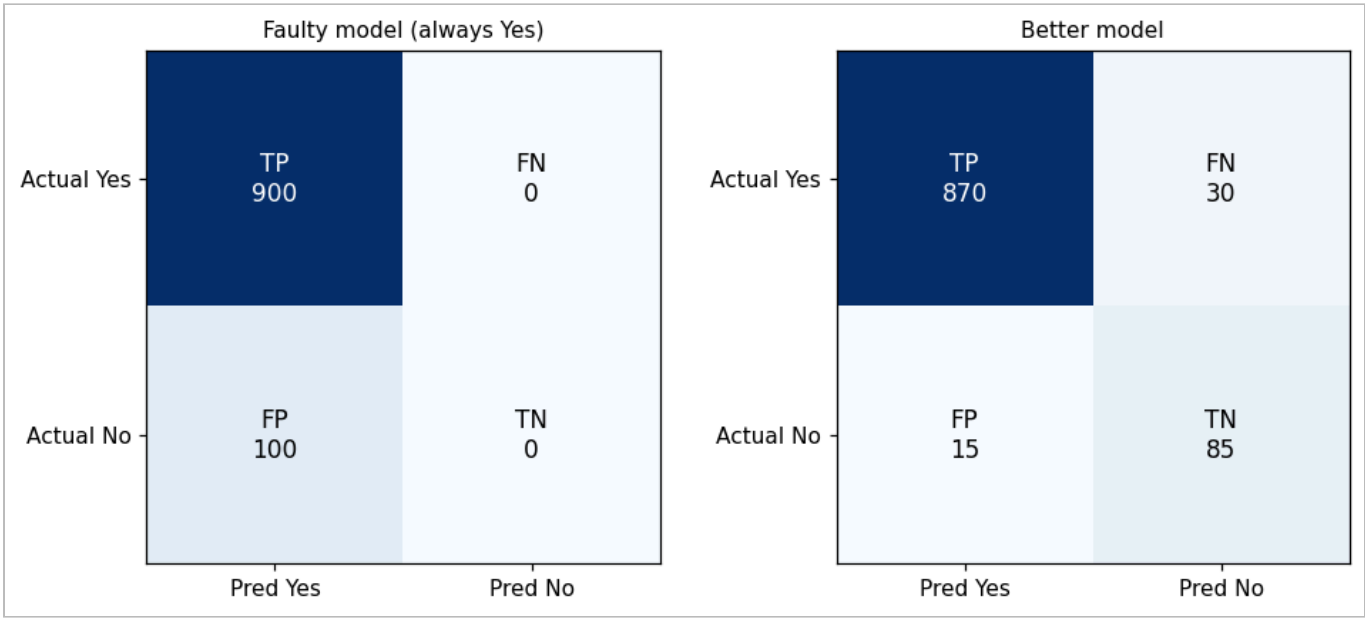
    plt.subplot(1, 2, k + 1)
    grid = [[tp, fn], [fp, tn]]
    plt.imshow(grid, cmap="Blues")
    labels = [["TP", "FN"], ["FP", "TN"]]
    for r in range(2):
        for c in range(2):
            if grid[r][c] > 450:
                ink = "white"
            else:
```

```
                ink = "black"
            plt.text(c, r, labels[r][c] + "\n" + str(grid[r][c]), ha="center", va="center",
                     color=ink, fontsize=11)
    plt.xticks([0, 1], ["Pred Yes", "Pred No"])
    plt.yticks([0, 1], ["Actual Yes", "Actual No"])
    plt.title(names[k], fontsize=10)
plt.gcf().set_size_inches(9, 4)
plt.tight_layout()
plt.show()
```

OUTPUT

```
Faulty model (always Yes)
  TP = 900  FN = 0  FP = 100  TN = 0
  Accuracy  = 0.9 = 90.0 %
  Precision = 0.9
  Recall    = 1.0
  F1 score  = 0.947
  Students who did not pass, found by the model: 0 of 100

Better model
  TP = 870  FN = 30  FP = 15  TN = 85
  Accuracy  = 0.955 = 95.5 %
  Precision = 0.983
  Recall    = 0.967
  F1 score  = 0.975
  Students who did not pass, found by the model: 85 of 100
```



Output figure — shown by plt.show()