SOFTWARE ENGINEERING

LAB FILE

CO-301

SUBMITTED IN COMPLETE FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE

DEGREE OF BACHELOR OF TECHNOLOGY IN

COMPUTER ENGINEERING

Submitted by:
Vatsalya Kumar Mishra
(2K21/CO/509)

Submitted to:
Prof.Rahul Kataria

DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY (FORMERLY Delhi College Of Engineering) Bawana Road, Delhi-110042

Title INDEX Date

Page No.

Experiment	Description	Page	Date	Signature
1	Hands-on experience on working with CASE tools like Meta Edit + and Rational Rose for Human Body Language Detector machine learning model.	1-4		
2	Apply various elicitation techniques for Human Body Language Detector model	5-8		
3	Draw Use Case Diagram for Human Body Language Detector model	9		
4	Draw use case diagram along with use case description	10-14		
5	Draw ER diagram for Human Body Language Detector model	15		
6	Draw different levels of DFD for Human Body Language Detector model	16-17		
7	Convert DFD to structure chart	18		
8	Develop a code to calculate effort and time period using different forms of COCOMO model.	19-23		
9	Develop test cases using boundary value analysis for the area of a triangle and factorial of a number n	25-27		
10	Develop a program to find cyclomatic complexity using DD path for problems like roots of quadratic equation	28-29		
11	Develop equivalence class test cases for classification of a triangle and calculating weekdays for a given date and next date	31-36		
12	Develop a program to predict reliability metric using different models	37-38		
DELTA®				

Title INDEX Date

Page No.

Title Experiment - 1 Date

Page No. 1

Aim : Hands-On Experience with CASE tools like Meta Edit+ and Rational Rose for Human Body Language Detector machine learning model.

Abstract

Computer-Aided Software Engineering (CASE) tools play a crucial role in the development of complex software systems. In this theoritical discussion, we will explore the hands-on experience of utilizing CASE tools, specifically Meta Edit+ and Rational Rose, to develop a Human Body Language Detector machine learning model. This comprehensive theory discusses the rationale, processes & benefits of using CASE tools in this context.

1 Introduction

The development of a Human Body Language Detector involves various stages, from requirements analysis to design, implementation and testing. CASE tools like Meta Edit+ and Rational Rose can significantly streamline these processes. This theoritical discussion delves into the application of these tools and their advantages.

2 Rationale for using CASE tools

- Complexity – developing a machine learning model for body language detection is a complex task that requires precise design and documentation.
- Collaboration – CASE tools enable better collaboration among team members and stakeholders, facilitating effective communication
- Productivity – automation of certain processes and tasks speeds up development and reduces human error.

DELTA®

- Traceability - CASE tools provide traceability of requirements to design, code and testing, ensuring that the final model aligns with the initial objectives.

3 Meta Edit+ and Rational Rose

3.1 Meta Edit+

- Meta Edit+ is a CASE tool designed for modelling complex systems using the Entity-Relationship modelling approach.
- It supports the creation of various diagrams such as class diagrams, activity diagrams and state diagrams.
- In our content, Meta Edit+ can be used for defining the structure of the machine learning model and its components.
- The tool allows for modelling the relationships between various elements involved in body language detection, such as features, algorithms and data sources.

3.2 Rational Rose

- Rational Rose is a CASE tool developed by IBM for visual modelling, code generation and documentation of software systems.
- It supports various UML diagrams like use case diagrams, sequence diagrams and class diagrams.
- In this scenario, Rational Rose can be used for designing the overall architecture of the body language detector model, specifying use cases and designing the system's classes and their relationships.

4 The Process of using CASE tools

4.1 Requirement Analysis

- Define the requirements for the Human Body Language Detector model, including its input sources, output formats and functionalities.
- Use Meta Edit+ to create Entity-Relationship diagrams that model the key components and relationships between them.

4.2 System Design

- Utilize Rational Rose to design the system's architecture using class diagrams, specifying the structure of the model, data flows and algorithms.
- Develop use case diagrams to depict the system's interactions with users and other components.

4.3 Implementation

- Depending on the programming language chosen for the model, translate the design from Rational Rose into code.
- Utilize Meta Edit+ for database schema design if required for data storage.

4.4 Testing

- Create test cases based on the requirements and design models
- Use CASE tools to trace the test cases to the design and requirements to ensure comprehensive test coverage.

5 Benefits of using CASE tools

- Improvised visualization — CASE tools provide clear visual representation of the system, making it easier to understand and communicate.
- Consistency — By using a consistent modelling approach it reduces inconsistencies and ambiguities in the design and implementation.
- Documentation — Automatically generated documentation from CASE tools is crucial for maintaining and evolving the system.
- Change management — CASE tools facilitate changes and updates in the model while maintaining its integrity.

6 Conclusion

Utilizing CASE tools like Meta Edit+ and Rational Rose in the development of a Human Body Language Detector machine learning model streamlines the entire process from requirements analysis to implementation and testing. These tools enhance collaboration, improve productivity and ensure the traceability of requirements throughout the development lifecycle. In conclusion, CASE tools are invaluable when developing complex software systems, including machine learning models for intricate tasks like body language detection.

Title Experiment-2 Date

Page No. 5

Aim : Apply various elicitation techniques for Human Body Language machine learning model.

Introduction

Eliciting requirements for a Human Body Language machine learning model is a critical step in its development process. Accurately capturing and understanding the stakeholders' needs is essential to building a successful model. Elicitation techniques are methods used to gather, clarify and document these requirements. This detailed note explores various elicitation techniques that can be applied in the content of developing a Human Body Language Detector model.

1 **Interviews**

- Conducting one-on-one or group interviews with potential users, domain experts and stakeholders can provide valuable insights.
- Key questions could include inquiries about the specific body language cues they want the model to detect, the content in which the model will be used and any special considerations.

2 **Surveys and Questionaires**

- Surveys and questionaires can be distributed to a broader audience to collect a large volume of requirements.
- Closed-ended questions can help gather quantitative data, while open-ended questions can provide qualitative insights.
- Stakeholders can be asked about their expectations, desired features and any challenges they anticipate with the system.

DELTA®

3 Workshops and Brainstorming Sessions

- Organize workshops with diverse participants, including developers, domain experts and potential users.
- Facilitate brainstorming sessions to encourage creative thinking and collaborative idea generation.
- These sessions can help uncover hidden requirements and innovate ideas for the body language detector.

4 Prototyping

- Create a basic prototype of the body language detector to demonstrate its functionalities to stakeholders.
- Users can interact with the prototype, providing feedback and helping to refine requirements based on their experience

5 Use Case Analysis

- Analyze use cases to understand how the model will be used in different scenarios.
- This technique can help identify system functionalities and interactions between the model and users, revealing specific requirements related to the model's behavior.

6 Contextual Inquiry

- Observe potential users in their natural environment to understand how they interpret and respond to body language cues.
- This technique can reveal unspoken or implicit requirements by observing real world behavior.

7 Document Analysis

- Review existing documents, reports or research papers related to Body language and human communication.
- This can help in understanding established principles and incorporating them into the models requirements.

8 Storytelling

- Encourage stakeholders to share stories and examples of situations where accurate body language detection would be beneficial.
- These narratives can provide content and inspire specific requirements.

9 Role Play

- Conduct role-playing sessions where stakeholders act out scenarios that involve using the body language detector.
- This technique can help uncover hidden user needs and validate requirements.

10 Cognitive Walkthrough

- Walk through potential user interactions with the model, step by step & evaluate the user experience.
- This can reveal usability issues and inform requirements related to the user interface and interaction design.

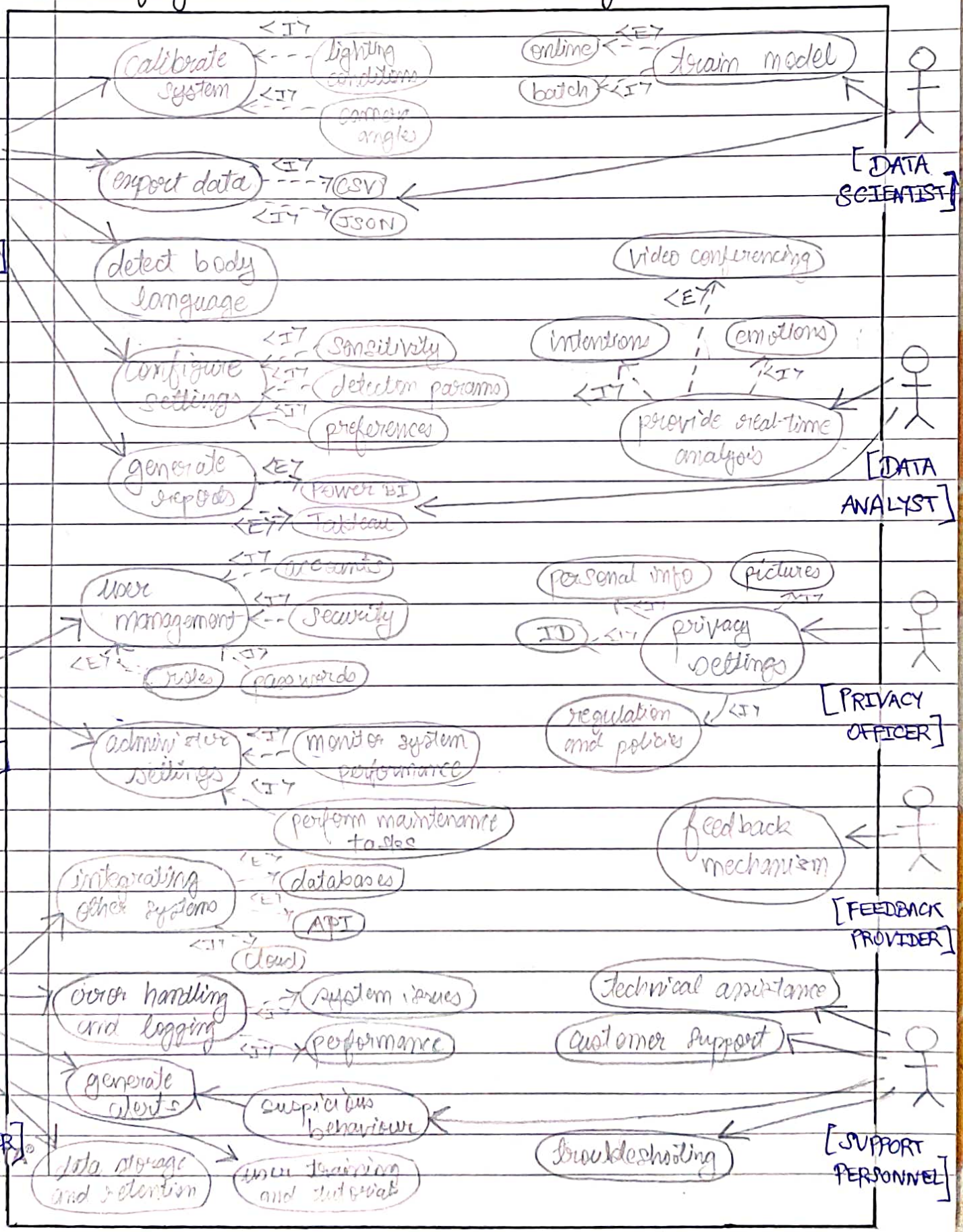
11 FAST (Facilitated Application Specification Technique)

- Introduction – start the workshop with key stakeholders, domain experts, potential users and developers by providing an introduction to the projects goals and objectives.
- Identify High Level goals – have participants brainstorm and identify high-level goals for the model. These goals should align with the project's mission and purpose.
- Define and Categorize objectives – for each high-level goal, break it down into specific objectives which must be concrete and actionable. Group similar objectives into categories or themes.
- Prioritize objectives – ask participants to use a prioritization technique (voting or ranking) to identify the most critical objectives. This helps in focusing on the most important features and functionalities.
- Document the results – create a matrix or document that lists the high-level goals, their corresponding objectives, categories and priorities.
- Review and Validate – review the results with the workshop participants to ensure that everyone is in agreement with the defined objectives and priorities.
- Iterate and Refine – if necessary, iterate through the process to further refine the objectives and priorities based on feedback from stakeholders.

Title Experiment - 3 Date

Page No. 9

Aim : Draw Use Case Diagram for Human Body Language Detector machine learning model

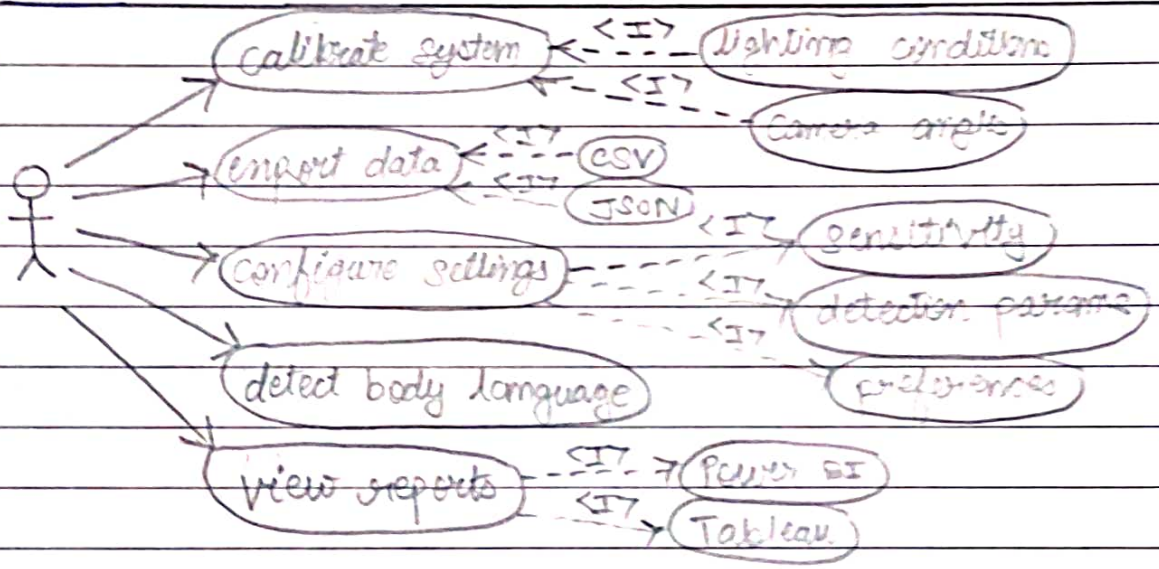


Title Experiment-4 Date

Page No. 10

Aim: Draw use case diagram along with use case description.

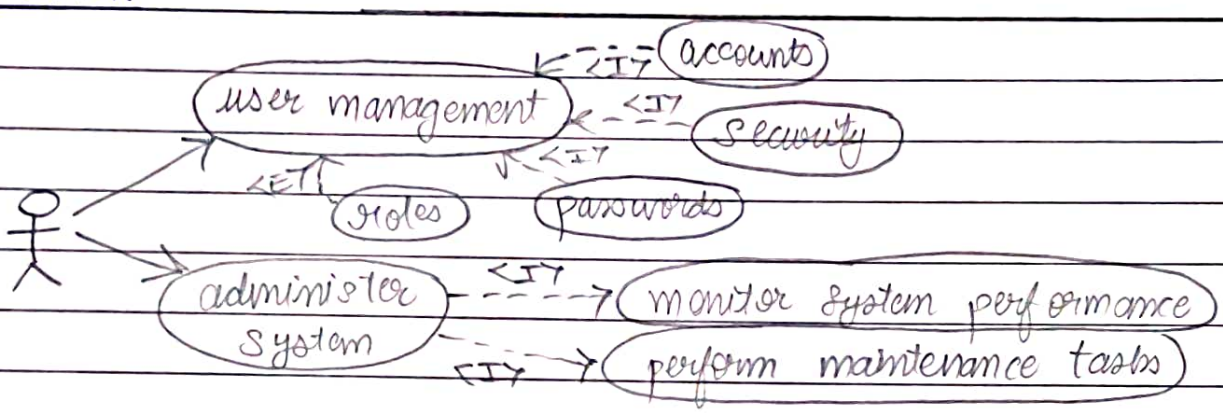
1 USER



- Usex – the primary actor who interacts with the system to use the model. This could be an individual, group or organization interested in analyzing body language axes for various purposes.
- Calibrate system – users may need to calibrate the system to account for different environmental factors such as lighting conditions or camera angles. This ensures accuracy.
- Export data – users may want to export the analyzed body language data for further analysis or reporting. This allows them to save the data in various formats (e.g. CSV, JSON)
- Configure settings – users may want to configure the systems settings such as adjusting sensitivity, selecting detection parameters, or setting preferences.
- Detect body language – this is the primary use case and represents the core functionality of the system.
- View reports – users can view detailed reports or visualizations of the analyzed bondy language data.

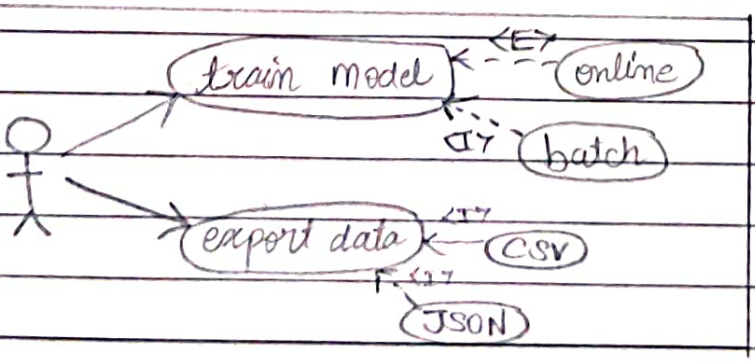
DELTA®

2 ADMIN



- Admin – an individual or team responsible for system administration, user management, system configuration and maintenance
- User management – in cases where administrators have access to the systems, they can manage user accounts, permissions, roles, passwords, security ensuring efficient system operation.
- Administer systems – this is for system administrators i.e use case who can manage user accounts, monitor system performance and perform maintenance tasks.

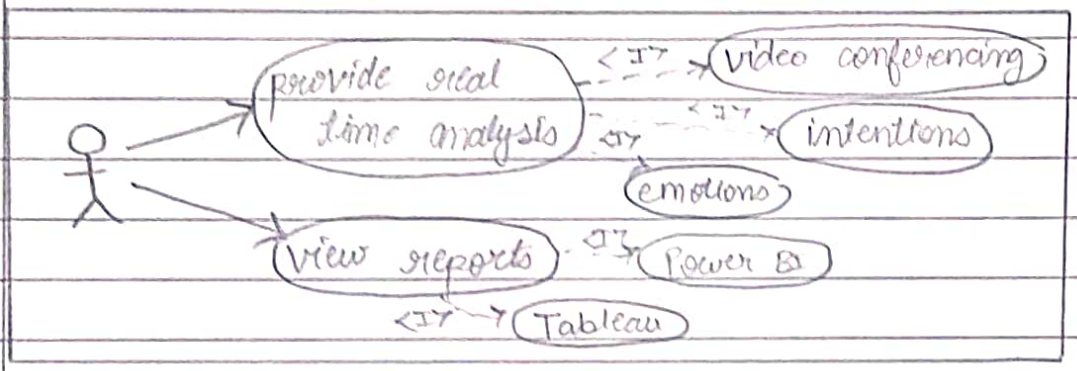
3 DATA SCIENTIST



- Data Scientist – individuals who are responsible for training and building the model doing necessary data cleaning and exploratory data analysis.

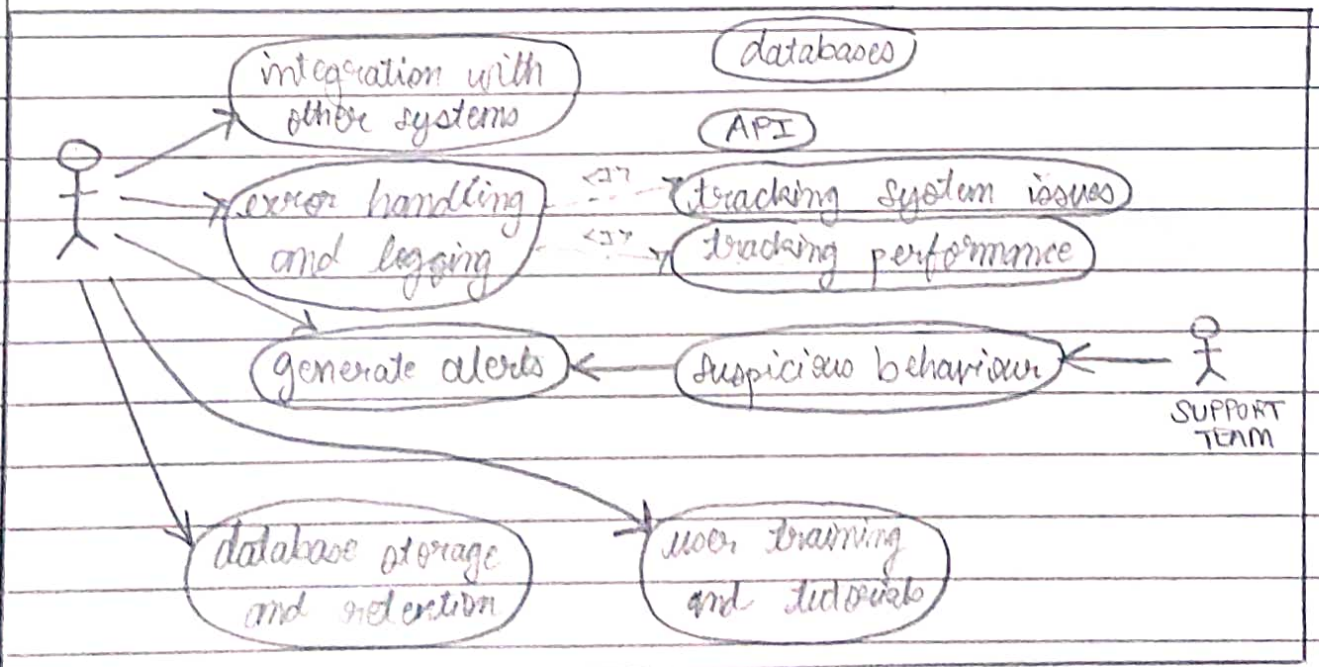
- Train model – this use case may be included if the system allows for training the machine learning model with new data to improve its accuracy over time.

4 DATA ANALYST



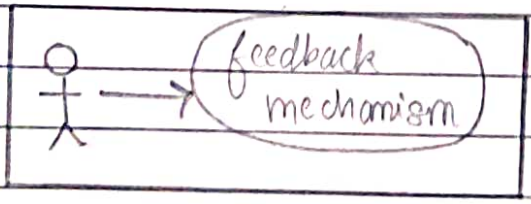
- Data Analyst – individuals who may analyze the data collected by the system, extract insights and provide feeback for improving the model's accuracy.
- Real time analysis – this use case involves continuously analyzing body language cues in real-time, which can be crucial in applications like video conferencing, security or healthcare.

5 SYSTEM DEVELOPER



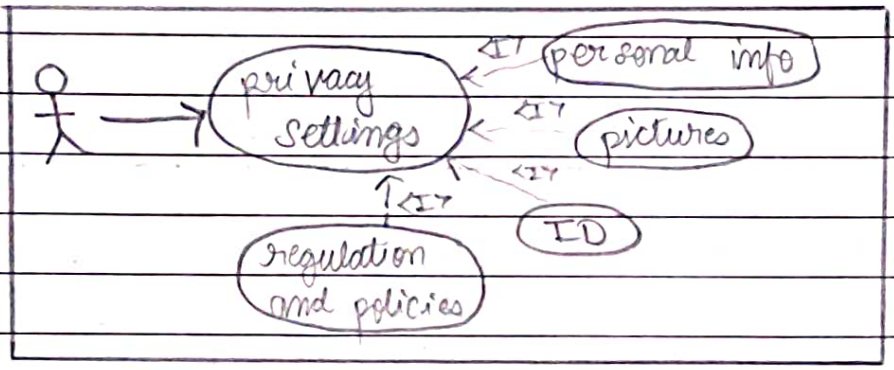
- System Developer - the professionals or team responsible for developing, enhancing and maintaining the machine learning model and its software components.
- Integration with other systems - in certain applications, the model may need to integrate with other systems or databases for additional content or data sharing. This ensures seamless integration.
- Error Handling and Logging - incorporate a use case for error handling and system logging, allowing for the tracking of system issues and performance.
- Generate Alerts - this use case involves the system generating alerts or notifications based on the detected body language cues.
  eg: trigger an alert if suspicious behaviour
- Data Storage and Retention - define how the system handles the storage and retention of body language data, considering data privacy regulations and user preferences.
- User Training and Tutorials - include a use case to provide users with training resources or tutorials to help them understand how to use the system effectively.

6 FEEDBACK PROVIDER



- Feedback provider - individuals who provide feedback on the system's analysis results or accuracy, helping in its continuous improvement.
- Feedback mechanism - the system can include a use case for users to provide feedback or corrections on the analysis results. this feedback loop can help improve the model's accuracy.

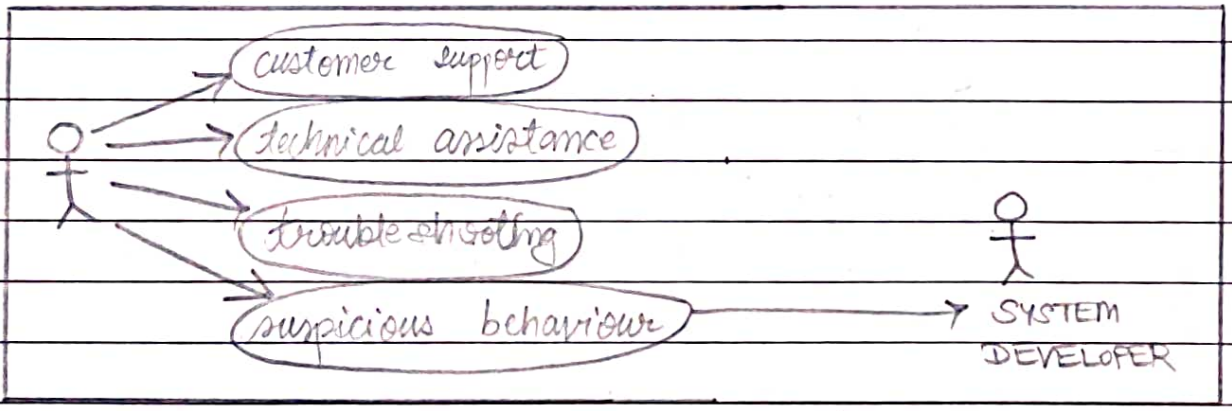
7 PRIVACY OFFICER



- Privacy Officer - in cases where data privacy is crucial, a PO could be responsible for ensuring that the system complies with privacy regulations and policies.

- Privacy settings - users may have the option to set privacy preferences, controlling who has access to their body language data and analysis results.

8 SUPPORT PERSONNEL



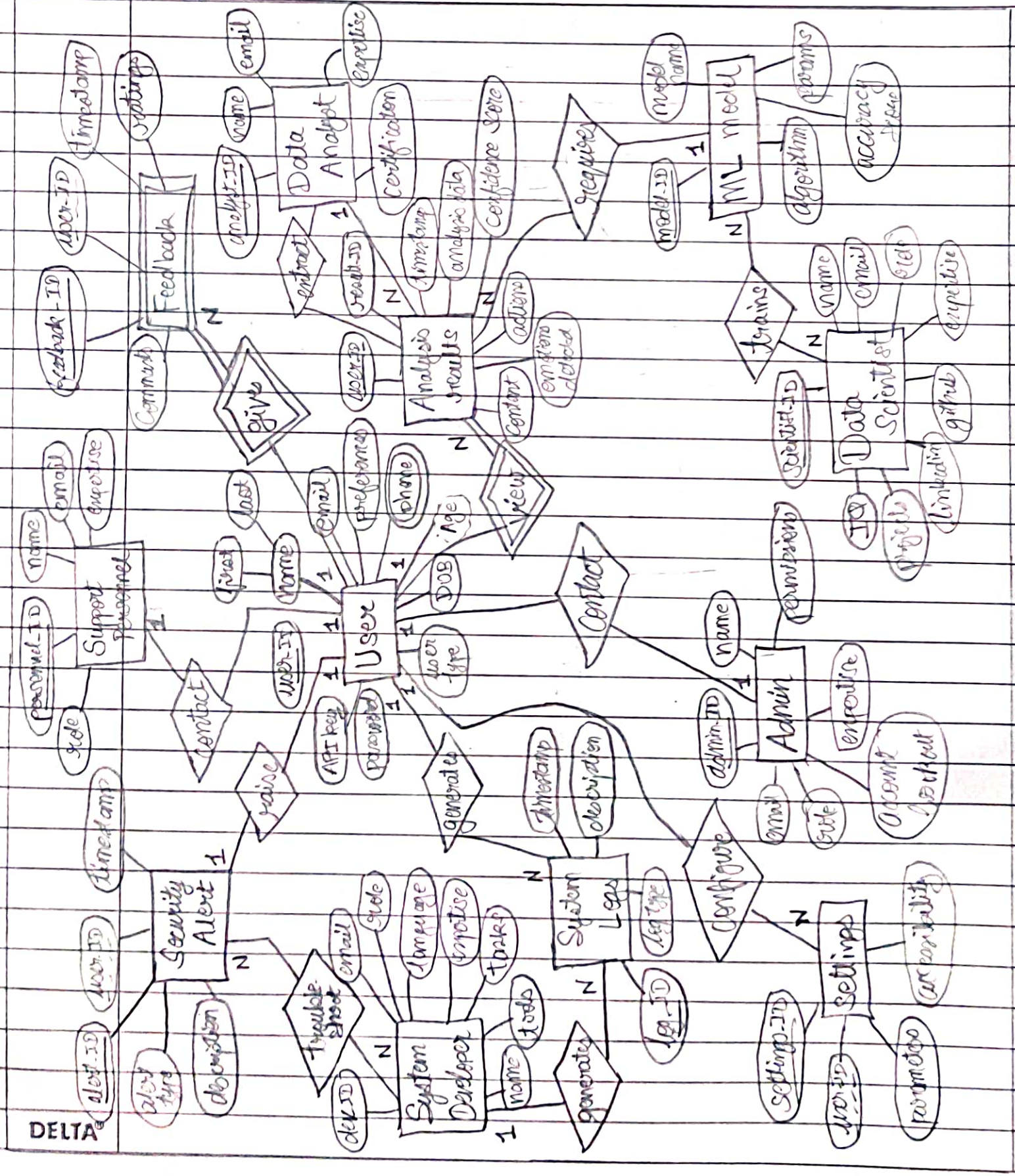
- Support personnel - individuals or a team responsible for providing customer support, technical assistance & troubleshooting for users of the system
- Customer support - aimed at assisting and satisfying customer which involves providing information, addressing inquiries and offering assistance with various aspects of a product or service
- Technical assistance - specialized form of customer support that focuses on resolving technical issues related to a product
- Troubleshooting - process of identifying, analyzing & resolving malfunctions that users encounter with a product.

Title Experiment-5

Date

Page No. 15

Aim: Draw ER diagram for Human Body Language Detector model.



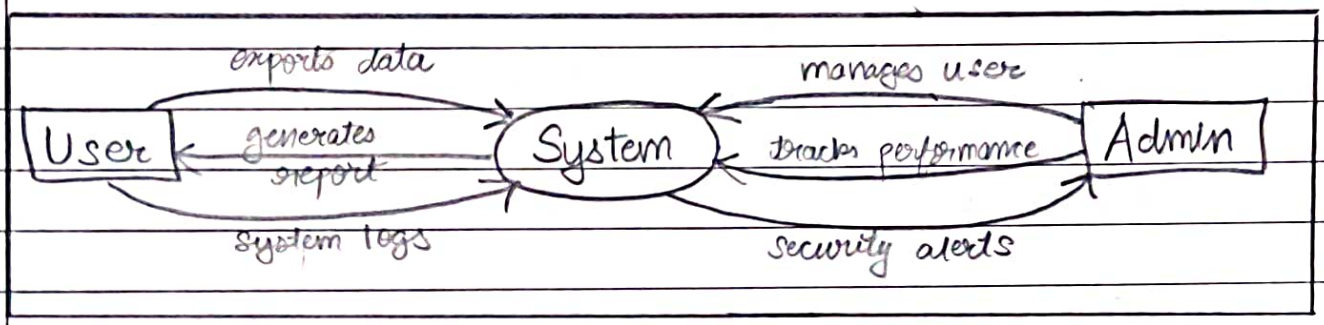
Title Experiment - 6

Date

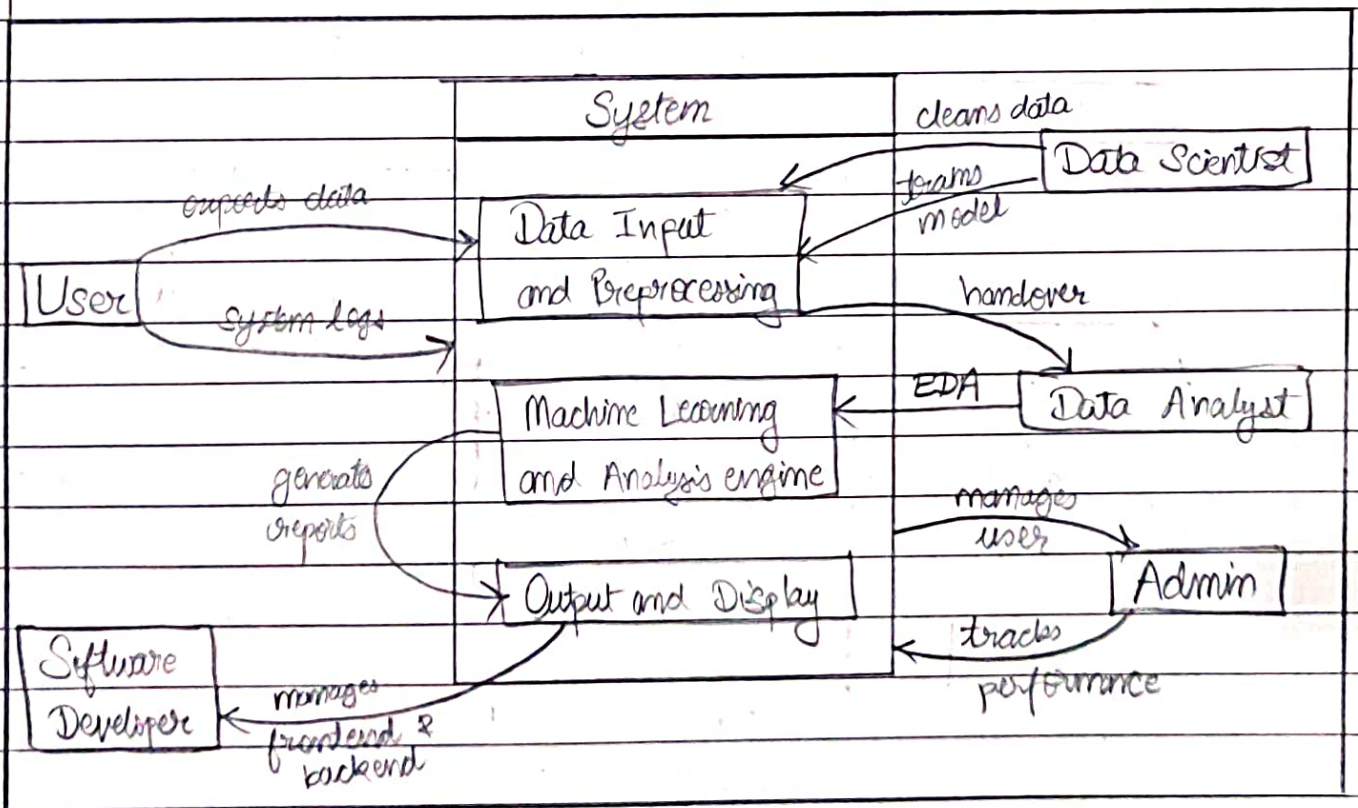
Page No. 16

Aim: Draw different levels of DFD for Human Body Language Detector machine learning model.

- Level 0 DFD

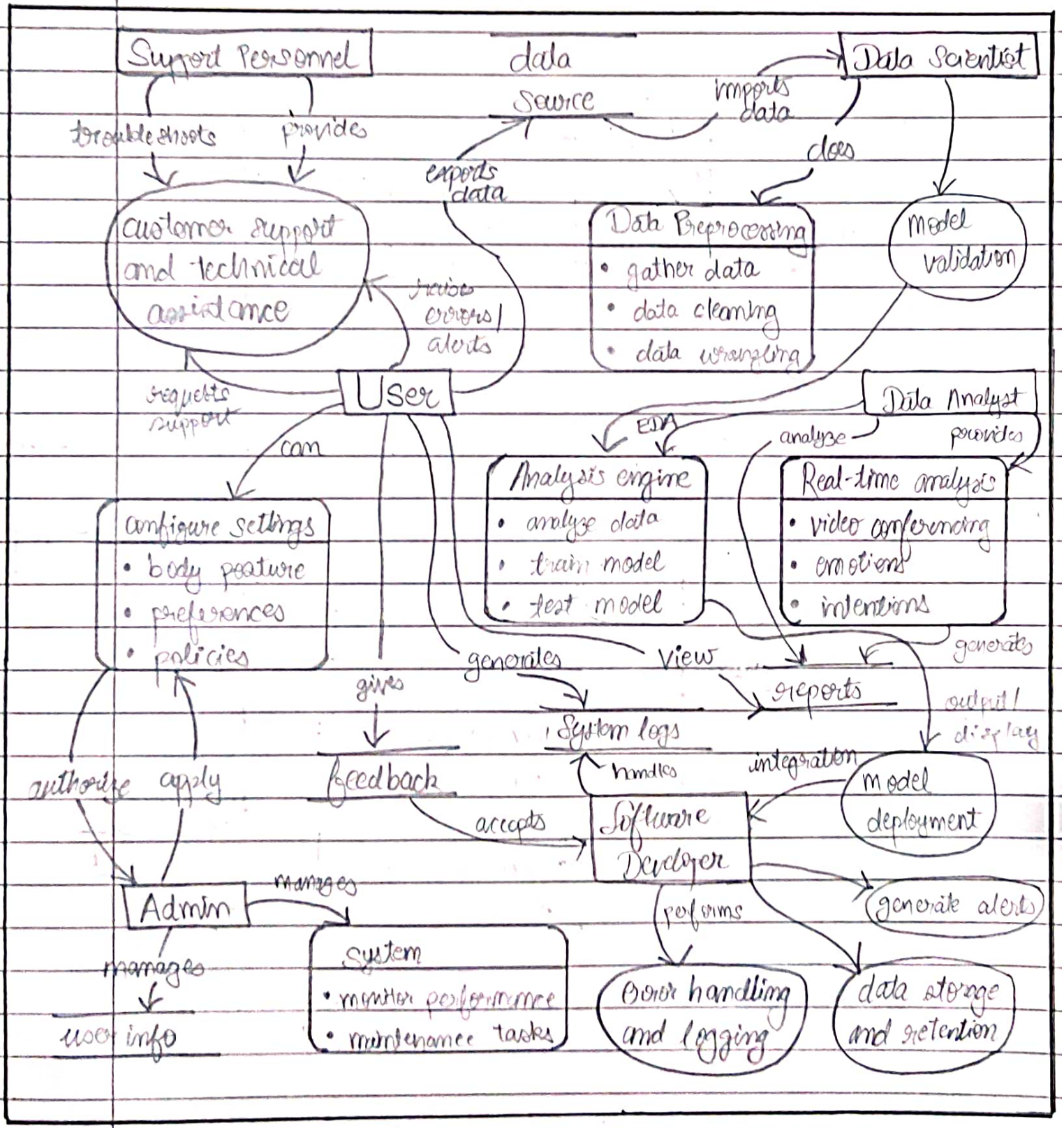


- Level 1 DFD



DELTA®

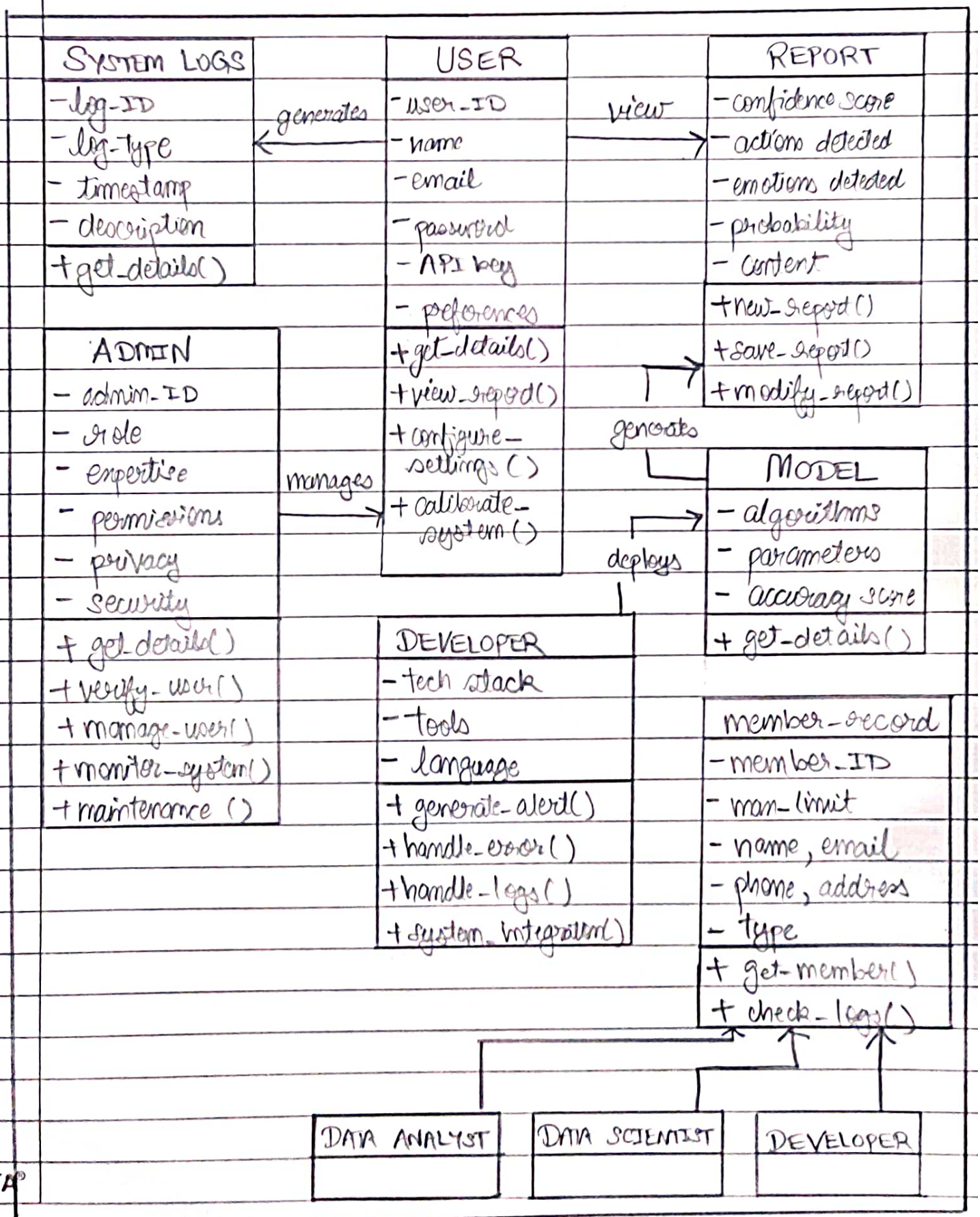
• Level 2 DFD


Support Personnel
troubleshoots
provide
customer support and technical assistance
raises errors/ alerts
data Source
imports data
exports data
Data Scientist
does
Data Preprocessing
• gather data
• data cleaning
• data wrangling
model validation
User
requests support
can
configure settings
• body posture
• preferences
• policies
Data Analyst
analyze
provides
ESN
Analysis engine
• analyze data
• train model
• test model
Real-time analysis
• video conferencing
• emotions
• intentions
generates
View
reports
gives
feedback
System logs
handles
integration
model deployment
output/ display
authorize
apply
accepts
Software Developer
Admin
manages
System
• monitor performance
• maintenance tasks
manages
user info
performs
Error handling and Logging
generate alerts
data storage and retention

Title Experiment - 7 Date

Page No. 18

Aim : Convert DFD to structure chart.



DELTA

Title Experiment-8 Date

Page No. 19

Aim: Develop a code to calculate and effort and time period using different forms of COCOMO model.

BASIC COCOMO

```
def calculate_basic_cocomo_effort(loc, language, project_type):
    # Define language-dependent parameters
    language_params = {
        "organic": (2.4, 1.05, 2.5),
        "semi-detached": (3.0, 1.12, 2.5),
        "embedded": (3.6, 1.20, 2.5)
    }

    if language not in language_params:
        raise ValueError("Invalid project type. Choose from 'organic', 'semi-
detached', or 'embedded'.")

    a, b, c = language_params[language]

    if project_type == "organic":
        exponent = 1.05
    elif project_type == "semi-detached":
        exponent = 1.12
    elif project_type == "embedded":
        exponent = 1.20
    else:
        raise ValueError("Invalid project type. Choose from 'organic', 'semi-
detached', or 'embedded'.")

    effort = a * (loc ** exponent)
    return effort


def calculate_basic_cocomo_time(effort, project_type):
    if project_type == "organic":
        b = 0.38
    elif project_type == "semi-detached":
        b = 0.35
    elif project_type == "embedded":
        b = 0.32
    else:
        raise ValueError("Invalid project type. Choose from 'organic', 'semi-
detached', or 'embedded'.")

    time = 3.0 * (effort ** b)
    return time
```

```
if __name__ == "__main__":
    # Customize these values for your specific project
    lines_of_code = 10000
    development_language = "semi-detached"  # Choose from 'organic', 'semi-
detached', 'embedded'
    project_type = "semi-detached"  # Choose from 'organic', 'semi-detached',
'embedded'

    estimated_effort = calculate_basic_cocomo_effort(lines_of_code,
development_language, project_type)
    estimated_time = calculate_basic_cocomo_time(estimated_effort,
project_type)

    print(f"Estimated Effort: {estimated_effort} person-months")
    print(f"Estimated Time: {estimated_time} months")
```

Intermediate Cocomo

```
def calculate_intermediate_cocomo_effort(loc, scale_factors, cost_drivers):
    # Define constants for the model
    a = 2.94
    b = 0.91

    # Calculate unadjusted function points (UFP)
    ufp = loc

    # Calculate technical complexity factor (TCF)
    tcf = 0.01 * sum(scale_factors)

    # Calculate environmental complexity factor (ECF)
    ecf = 0.01 * sum(cost_drivers)

    # Calculate adjusted function points (AFP)
    afp = ufp * (0.65 + 0.01 * tcf * ecf)

    # Calculate effort in person-months
    effort = a * (afp ** b)

    return effort
```

```
def calculate_intermediate_cocomo_time(effort):
    # Define constants for the model
    c = 3.67
    d = 0.28

    # Calculate time in months
    time = c * (effort ** d)

    return time


if __name__ == "__main__":
    # Customize these values for your specific project
    lines_of_code = 10000
    scale_factors = [1.03, 1.07, 1.12, 1.20, 0.85]
    cost_drivers = [1.10, 1.05, 0.95, 0.90, 1.07, 1.15, 1.10]

    estimated_effort = calculate_intermediate_cocomo_effort(lines_of_code,
scale_factors, cost_drivers)
    estimated_time = calculate_intermediate_cocomo_time(estimated_effort)

    print(f"Estimated Effort: {estimated_effort} person-months")
    print(f"Estimated Time: {estimated_time} months")
```

ADVANCED COCOMO

```
def calculate_advanced_cocomo_effort(size, scale_factors, cost_drivers,
modes):
    # Define constants for the model
    a = 2.94
    b = 0.91

    # Calculate adjusted size based on scale factors
    adjusted_size = size * sum(scale_factors)

    # Calculate effort in person-months
    effort = a * (adjusted_size ** b)

    # Apply mode scaling factor
    mode_scale = {
        "organic": 1.0,
        "semi-detached": 1.3,
        "embedded": 1.6
    }
    effort *= mode_scale[modes]

    # Apply cost drivers
    for driver in cost_drivers:
        effort *= driver

    return effort

def calculate_advanced_cocomo_time(effort, modes):
    # Define constants for the model
    c = 3.67
    d = 0.28

    # Apply mode scaling factor
    mode_scale = {
        "organic": 1.0,
        "semi-detached": 1.1,
        "embedded": 1.2
    }
    effort *= mode_scale[modes]

    # Calculate time in months
    time = c * (effort ** d)

    return time
```

```
if __name__ == "__main__":
    # Customize these values for your specific project
    lines_of_code = 10000
    scale_factors = [1.03, 1.07, 1.12, 1.20, 0.85]
    cost_drivers = [1.24, 0.89, 0.89, 0.89, 1.21, 1.23, 1.21, 1.14]
    project_modes = "semi-detached"  # Choose from 'organic', 'semi-detached',
'embedded'

    estimated_effort = calculate_advanced_cocomo_effort(lines_of_code,
scale_factors, cost_drivers, project_modes)
    estimated_time = calculate_advanced_cocomo_time(estimated_effort,
project_modes)

    print(f"Estimated Effort: {estimated_effort} person-months")
    print(f"Estimated Time: {estimated_time} months")
```

Title Experiment-9 Date

Page No. 25

Aim: Develop test cases using boundary value analysis for the area of a triangle and factorial of a number n.

Boundary value analysis is a testing technique that focuses on test cases at the boundaries or extremes of input domains. Here, we'll create test cases for two specific functions: calculating the area of a triangle and calculating the factorial of a number n.

① Area of a Triangle (Boundary Value Analysis)

assuming we have a function that calculates the area of a triangle based on its three sides (a, b and c) we'll create test cases for boundary values of the sides:

- Test Case-1 (Minimum Side Values)

  a=0, b=0, c=0

  ⇒ Expected result: The function should return an error or an invalid result since a triangle cannot have sides with zero length.

- Test Case-2 (Maximum Side Values)

  a=10000, b=10000, c=10000

  ⇒ Expected result: The function should calculate and return the area of the equilateral triangle with sides of maximum length.

- Test Case-3 (One Side is zero)

  a=0, b=5, c=7

  ⇒ Expected result: The function should return an error or an invalid result as a triangle cannot exist with one side being zero.

DELTA®

- **Test Case-4** (Two Sides Equal Maximum)
  a=10000, b=10000, c=5000
  ⇒ Expected result: The function should calculate and return the area of the triangle with two sides equal to the maximum length.

- **Test Case-5** (One Side Much Larger Than Others)
  a=100, b=150, c=5000
  ⇒ Expected result: The function should return an error or an invalid result as the side 'c' is much larger than 'a' and 'b', making it impossible to form a triangle.

(2) **Factorial of a number 'n'** (Boundary Value Analysis)
assuming we have a function to calculate the factorial of a number n, will create test cases for boundary values.

- **Test Case-1** (Minimum Value)
  n=0
  Expected result ⇒ the function should return 1 as 0! = 1.

- **Test Case-2** (Positive Integer)
  n=5
  Expected result ⇒ The function should return after calculating the value of 5! which is 120.

- **Test Case-3** (Maximum Value within range)
  n=12
  Expected result ⇒ the function should return 12! = 479001600

- Test Case - 4 (Value Beyond the range)

n = 20

Expected result → The function should handle or raise an error for values beyond the supported range, as factorials of large numbers may result in integer overflow.

- Test Case - 5 (Negative Value)

n = -5

Expected result → The function should raise an error or return an invalid result because factorials are not defined for negative numbers.

Title Experiment - 10

Date

Page No. 28

Aim: Develop a program to find cyclomatic complexity using DD path for problems like root of quadratic equation.

```
import re

def find_roots(a, b, c):
    """
    Function to find the roots of a quadratic equation of the form: ax^2 + bx
+ c = 0.
    """
    discriminant = b**2 - 4*a*c
    if discriminant > 0:
        root1 = (-b + discriminant**0.5) / (2*a)
        root2 = (-b - discriminant**0.5) / (2*a)
        return "Two distinct real roots:", root1, root2
    elif discriminant == 0:
        root1 = root2 = -b / (2*a)
        return "One real root:", root1
    else:
        realPart = -b / (2*a)
        imaginaryPart = (abs(discriminant)**0.5) / (2*a)
        return "Complex roots:", realPart, "+", imaginaryPart, "i and",
realPart, "-", imaginaryPart, "i"
```

```
def cyclomatic_complexity(code):
    # Count the number of nodes (N)
    nodes = len(re.findall(r'\bif\b|\belif\b|\belse\b|\bwhile\b', code))

    # Count the number of edges (E)
    edges = len(re.findall(r'\bif\b|\belif\b', code)) + nodes

    # For a single connected component, P = 1
    p = 1

    # Calculate Cyclomatic Complexity
    complexity = edges - nodes + 2 * p
    return complexity
```

DELTA

```
if __name__ == "__main__":
    code = """
if discriminant > 0:
    # Two distinct real roots
    root1 = (-b + discriminant**0.5) / (2*a)
    root2 = (-b - discriminant**0.5) / (2*a)
elif discriminant == 0:
    # One real root
    root1 = root2 = -b / (2*a)
else:
    # Complex roots
    realPart = -b / (2*a)
    imaginaryPart = (abs(discriminant)**0.5) / (2*a)
    """

    print("Cyclomatic Complexity:", cyclomatic_complexity(code))
```

Title Experiment-11 Date

Page No. 31

Aim: Develop equivalence class test cases for classification of a triangle and calculating weekdays for a given date and next date.

① Triangle Problem (BVA)

It accepts three integers - a, b, c as three sides of the triangle. We define a range for the sides as [l, r] where l > 0. It returns the type of triangle (Scalene, Isosceles, Equilateral, not a Triangle) formed by a, b, c.

For a, b, c to form a triangle the following conditions should be satisfied

- a < b+c
- b < a+c
- c < a+b

If any of these conditions is violated, output is not a triangle.

Range = [1, 100]
Nominal value = 50

Total test cases = 4n+1 = 4(3)+1 = 13

Test Case ID	a	b	c	Expected Output
T_1	1	50	50	Isosceles
T_2	2	50	50	Isosceles
T_3	99	50	50	Isosceles
T_4	100	50	50	Not a triangle
T_5	50	50	50	Equilateral
T_6	50	1	50	Isosceles
T_7	50	2	50	Isosceles
T_8	50	99	50	Isosceles
T_9	50	100	50	Not a Triangle
T_{10}	50	50	1	Isosceles
T_{11}	50	50	2	Isosceles
T_{12}	50	50	99	Isosceles
T_{13}	50	50	100	Not a Triangle

(2) Next Date Problem (BVA)

given a day in the format of DD-MM-YYYY, you need to find the next date for the given date.

Conditions

D : $1 \leq day \leq 31$

M : $1 \leq month \leq 12$

Y : $1800 \leq year \leq 2048$

no: of test cases = $4n+1 = 4 \times 3 + 1 = 13$

Test Case ID	Day	Month	Year	Expected Output
T1	1	6	2000	2/6/2000
T2	2	6	2000	3/6/2000
T3	15	6	2000	16/6/2000
T4	30	6	2000	1/7/2000
T5	31	6	2000	Invalid date
T6	15	1	2000	16/1/2000
T7	15	2	2000	16/2/2000
T8	15	11	2000	16/11/2000
T9	15	12	2000	16/12/2000
T10	15	6	1800	16/6/1800
T11	15	6	1801	16/6/1801
T12	15	6	2047	16/6/2047
T13	15	6	2048	16/6/2048

③ Triangle Problem (ECT)

$I_1 = \{0 < a \leq 10\}$
$I_2 = \{a < 0\}$
$I_3 = \{a > 10\}$
$I_4 = \{0 < b \leq 10\}$
$I_5 = \{b < 0\}$
$I_6 = \{b > 10\}$
$I_7 = \{0 < c \leq 10\}$
$I_8 = \{c < 0\}$
$I_9 = \{c > 10\}$
$I_{10} = \{a = b = c\}$
$I_{11} = \{a = b, b != c\}$
$I_{12} = \{b = c, c != a\}$
$I_{13} = \{a = c, c != b\}$
$I_{14} = \{a != b != c\}$
$I_{15} = \{a + b = c\}$
$I_{16} = \{a + b < c\}$
$I_{17} = \{b + c = a\}$
$I_{18} = \{b + c < a\}$
$I_{19} = \{c + a = b\}$
$I_{20} = \{c + a > b\}$

O_1 = not a triangle
O_2 = equilateral
O_3 = isosceles
O_4 = scalene

4 outcomes & 20 possible i/p classes
⇒ 24 test cases

Classes	a	b	c	Expected Output	Program Output	Tested Outcome
O_1	10	5	5	not a triangle	not a triangle	pass
O_2	5	5	5	equilateral	equilateral	pass
O_3	1	5	5	isosceles	isosceles	pass
O_4	10	9	5	Scalene	Scalene	pass
I_1	5	5	5	equilateral	equilateral	pass
I_2	0	5	5	invalid	invalid	pass
I_3	11	5	5	invalid	invalid	pass
I_4	5	5	5	equilateral	equilateral	pass
I_5	5	0	5	invalid	invalid	pass
I_6	5	11	5	invalid	invalid	pass
I_7	5	5	5	equilateral	equilateral	pass
I_8	5	5	0	invalid	invalid	pass
I_9	5	5	11	invalid	invalid	pass
I_{10}	5	5	5	equilateral	equilateral	pass
I_{11}	5	5	1	isosceles	isosceles	pass
I_{12}	1	5	5	Isosceles	Isosceles	pass
I_{13}	5	1	5	isosceles	isosceles	pass
I_{14}	9	5	10	Scalene	Scalene	pass
I_{15}	5	5	10	not a triangle	not a triangle	pass
I_{16}	1	5	10	not a triangle	not a triangle	pass
I_{17}	10	5	5	not a triangle	not a triangle	pass
I_{18}	10	5	1	not a triangle	not a triangle	pass
I_{19}	5	10	5	not a triangle	not a triangle	pass
I_{20}	5	10	1	not a triangle	not a triangle	pass

(4) Next Date (ECT)

Input classes

Day:

D1 : day between 1 to 28

D2 : 29

D3 : 30

D4 : 31

Month:

M1 : has 30 days

M2 : has 31 days

M3 : is February

Year :

Y1 : leap year

Y2 : normal year

Output classes

Increment Day

Reset Day and Increment Month

Increment Year

Invalid Date

∴ 24 test cases

Test Case ID	Day	Month	Year	Expected Output
E1	15	4	2004	16/4/2004
E2	15	4	2003	16/4/2003
E3	15	1	2004	16/1/2004
E4	15	1	2003	16/1/2003
E5	15	2	2004	16/2/2004
E6	15	2	2003	16/2/2003
E7	29	4	2004	30/4/2004
E8	29	4	2003	30/4/2003
E9	29	1	2004	30/1/2004
E10	29	1	2003	30/1/2003
E11	29	2	2004	1/3/2004
E12	29	2	2003	invalid date
E13	30	4	2004	1/5/2004
E14	30	4	2003	1/5/2003
E15	30	1	2004	31/1/2004
E16	30	1	2003	31/1/2003
E17	30	2	2004	invalid date
E18	30	2	2003	invalid date
E19	31	4	2004	invalid date
E20	31	4	2003	invalid date
E21	31	1	2004	1/2/2004
E22	31	1	2003	1/5/2003
E23	31	2	2004	invalid date
E24	31	2	2003	invalid date

Aim: Develop a program to predict reliability metric using different models.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset into a Pandas DataFrame
data = pd.read_csv("your_reliability_data.csv")  # Replace with your dataset
file

# Define the features (independent variables) and target (reliability metric)
X = data[["feature1", "feature2", "feature3"]]  # Customize with your features
y = data["reliability_metric"]  # Replace with your target column
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Choose and train different regression models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest Regressor": RandomForestRegressor()
}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Print the results
    print(f"Model: {model_name}")
    print(f"Mean Squared Error: {mse:.2f}")
    print(f"R-squared (R2) Score: {r2:.2f}")
    print("\n")
```

```
# Choose the best-performing model for prediction
best_model = models["Linear Regression"]  # Replace with the best-performing
model

# Predict reliability metric for a new set of features
new_features = [[value1, value2, value3]]  # Customize with your new feature
values
predicted_reliability = best_model.predict(new_features)

print("Predicted Reliability Metric:", predicted_reliability[0])
```

Title Experiment-13 Date

Page No. 39

Aim : Develop a program to calculate maintenance metrics using different models.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Load your time-series maintenance data into a Pandas DataFrame
data = pd.read_csv("your_time_series_maintenance_data.csv")  # Replace with
your dataset file
# Assuming your dataset has a 'date' column and a 'maintenance_metric' column
data['date'] = pd.to_datetime(data['date'])
data.set_index('date', inplace=True)

# Visualize your time-series data
data['maintenance_metric'].plot()
plt.xlabel('Date')
plt.ylabel('Maintenance Metric')
plt.show()

# Decompose the time series (optional)
decomposition = sm.tsa.seasonal_decompose(data['maintenance_metric'],
model='additive')
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Fit an ARIMA model
order = (1, 1, 1)  # Define ARIMA(p, d, q) parameters
model = sm.tsa.ARIMA(data['maintenance_metric'], order=order)
results = model.fit()

# Make predictions
forecast_steps = 10  # Customize the number of steps to forecast
forecast, stderr, conf_int = results.forecast(steps=forecast_steps)

# Print the forecasted maintenance metrics
print("Forecasted Maintenance Metrics:")
print(forecast)
# Visualize the forecast
plt.plot(data['maintenance_metric'], label='Observed')
plt.plot(pd.date_range(start=data.index[-1], periods=forecast_steps + 1,
closed='right'), forecast, label='Forecast', color='red')
plt.xlabel('Date')
plt.ylabel('Maintenance Metric')
plt.legend()
plt.show()
```

Source Code

body_lang_detector_jupyter_notebook.ipynb

C: > Users > HP > Desktop > PROJECTS > Human_Body_Language_Detector > body_lang_detector_jupyter_notebook.ipynb >

+ Code + Markdown ···

```
import mediapipe as mp # Import mediapipe
import cv2 # Import opencv
```

[1]

```
mp_drawing = mp.solutions.drawing_utils # Drawing helpers
mp_holistic = mp.solutions.holistic # Mediapipe Solutions
```

[2]

```
results.face_landmarks.landmark[0].visibility
```

[4]

··· 0.0

```
import csv
import os
import numpy as np
```

[5]

```
num_coords = len(results.pose_landmarks.landmark)+len(results.face_landmarks.landmark)
num_coords
```

[6]

··· 501

```
landmarks = ['class']
for val in range(1, num_coords+1):
    landmarks += ['x{}'.format(val), 'y{}'.format(val), 'z{}'.format(val), 'v{}'.format(val)]
```

[7]

```
with open('coords.csv', mode='w', newline='') as f:
    csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    csv_writer.writerow(landmarks)
```

[19]

```
class_name = "Quiet"
```

[30]

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

[31]

```
df = pd.read_csv('coords.csv')
```

[32]

body_lang_detector_jupyter_notebook.ipynb

C: > Users > HP > Desktop > PROJECTS > Human_Body_Language_Detector > body_lang_detector_jupyter_notebook.ipynb > in

+ Code + Markdown ...

```
df.head()
```

[33]

	class	x1	y1	z1	v1	x2	y2	z2	v2	x3
0	Happy	0.599210	0.421257	-0.965923	0.999864	0.600432	0.363008	-0.898745	0.999755	0.611158
1	Happy	0.599036	0.416200	-0.757805	0.999838	0.600009	0.360027	-0.701460	0.999716	0.610229
2	Happy	0.603343	0.405795	-0.726382	0.999753	0.601946	0.353116	-0.673239	0.999596	0.610694
3	Happy	0.604033	0.386957	-0.721525	0.999669	0.601428	0.336696	-0.663889	0.999476	0.608700
4	Happy	0.603981	0.380664	-0.744553	0.999604	0.600813	0.330353	-0.684059	0.999379	0.607462

5 rows × 2005 columns

```
X = df.drop('class', axis=1) # features
y = df['class'] # target value
```

[36]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)
```

[37]

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

[38]

```
pipelines = {
    'lr':make_pipeline(StandardScaler(), LogisticRegression()),
    'rc':make_pipeline(StandardScaler(), RidgeClassifier()),
    'rf':make_pipeline(StandardScaler(), RandomForestClassifier()),
    'gb':make_pipeline(StandardScaler(), GradientBoostingClassifier()),
}
```

[39]

```
fit models
```

[41]

```
{'lr': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('logisticregression', LogisticRegression())]),
 'rc': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('ridgeclassifier', RidgeClassifier())]),
 'rf': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('randomforestclassifier', RandomForestClassifier())]),
 'gb': Pipeline(steps=[('standardscaler', StandardScaler()),
                       ('gradientboostingclassifier', GradientBoostingClassifier())])}
```

```
from sklearn.metrics import accuracy_score # Accuracy metrics
import pickle
```

[43]

DELTA®

```
for algo, model in fit_models.items():
    yhat = model.predict(X_test)
    print(algo, accuracy_score(y_test, yhat))
```

```
lr 1.0
rc 1.0
rf 1.0
gb 0.9938271604938271
```

body_lang_detector_jupyter_notebook.ipynb

C: > Users > HP > Desktop > PROJECTS > Human_Body_Language_Detector > body_lang_detector_jupyter_not

+ Code + Markdown ···

```
fit_models['rf'].predict(X_test)
```

[45]

```
array(['Sad', 'Sad', 'Happy', 'Happy', 'Quiet', 'Sad', 'Victorious',
       'Sad', 'Happy', 'Happy', 'Sad', 'Wakanda Forever', 'Victorious',
       'Happy', 'Sad', 'Victorious', 'Wakanda Forever', 'Victorious',
       'Quiet', 'Wakanda Forever', 'Victorious', 'Victorious',
       'Wakanda Forever', 'Sad', 'Wakanda Forever', 'Happy',
       'Wakanda Forever', 'Victorious', 'Happy', 'Victorious',
       'Victorious', 'Sad', 'Happy', 'Sad', 'Wakanda Forever', 'Quiet',
       'Victorious', 'Sad', 'Sad', 'Wakanda Forever', 'Wakanda Forever',
       'Wakanda Forever', 'Sad', 'Victorious', 'Wakanda Forever', 'Sad',
       'Quiet', 'Wakanda Forever', 'Sad', 'Victorious', 'Happy',
       'Wakanda Forever', 'Victorious', 'Quiet', 'Wakanda Forever',
       'Quiet', 'Wakanda Forever', 'Victorious', 'Victorious', 'Sad',
       'Wakanda Forever', 'Victorious', 'Happy', 'Wakanda Forever', 'Sad',
       'Happy', 'Wakanda Forever', 'Victorious', 'Quiet', 'Sad', 'Quiet',
       'Wakanda Forever', 'Wakanda Forever', 'Wakanda Forever',
       'Wakanda Forever', 'Sad', 'Wakanda Forever', 'Wakanda Forever',
       'Wakanda Forever', 'Happy', 'Quiet', 'Wakanda Forever', 'Happy',
       'Happy', 'Victorious', 'Victorious', 'Victorious', 'Sad', 'Sad',
       'Wakanda Forever', 'Happy', 'Quiet', 'Happy', 'Victorious',
       'Wakanda Forever', 'Sad', 'Wakanda Forever', 'Wakanda Forever',
       'Victorious', 'Wakanda Forever', 'Sad', 'Wakanda Forever', 'Happy',
       'Wakanda Forever', 'Happy', 'Wakanda Forever', 'Wakanda Forever',
       'Wakanda Forever', 'Victorious', 'Victorious', 'Happy',
       'Wakanda Forever', 'Victorious', 'Victorious', 'Victorious',
       'Victorious', 'Victorious', 'Sad', 'Victorious', 'Victorious',
  ...
       'Wakanda Forever', 'Victorious', 'Victorious', 'Sad', 'Victorious',
       'Victorious', 'Victorious', 'Victorious', 'Happy', 'Victorious',
       'Victorious', 'Wakanda Forever', 'Happy', 'Victorious', 'Sad',
       'Sad', 'Victorious', 'Sad', 'Happy', 'Happy', 'Victorious', 'Sad',
       'Quiet', 'Sad', 'Quiet', 'Wakanda Forever', 'Sad'], dtype=object)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
y_test
```

```
181                 Sad
138                 Sad
11                Happy
21                Happy
524               Quiet
              ...
484               Quiet
122                 Sad
477               Quiet
469    Wakanda Forever
134                 Sad
Name: class, Length: 162, dtype: object
```

```
with open('body_language.pkl', 'wb') as f:
    pickle.dump(fit_models['rf'], f)
```

```
with open('body_language.pkl', 'rb') as f:
    model = pickle.load(f)
```

```
model
```

Pipeline

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('randomforestclassifier', RandomForestClassifier())])
```

StandardScaler

```
StandardScaler()
```

RandomForestClassifier

```
RandomForestClassifier()
```

Sample Data Collection

A1 fx class

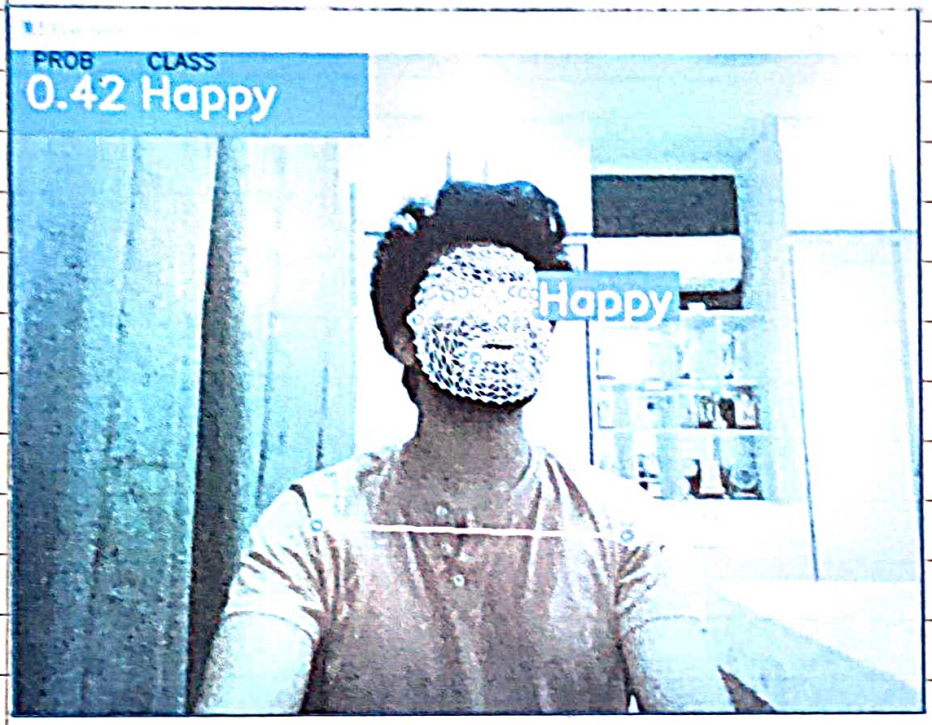
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	class	x1	y1	z1	v1	x2	y2	z2	v2	x3	y3	z3	v3
2	Happy	0.59921	0.421257	-0.96592	0.999864	0.600432	0.363008	-0.89875	0.999755	0.611158	0.361657	-0.8988	0.999712
3	Happy	0.599036	0.4162	-0.75781	0.999838	0.600009	0.360027	-0.70146	0.999716	0.610229	0.358796	-0.7014	0.999659
4	Happy	0.603343	0.405795	-0.72638	0.999753	0.601946	0.353116	-0.67324	0.999596	0.610694	0.352637	-0.67312	0.999517
5	Happy	0.604033	0.386957	-0.72152	0.999669	0.601428	0.336696	-0.66389	0.999476	0.6087	0.336396	-0.66378	0.999387
6	Happy	0.603981	0.380664	-0.74455	0.999604	0.600813	0.330353	-0.68406	0.999379	0.607462	0.330262	-0.68397	0.999277
7	Happy	0.591037	0.382752	-0.80809	0.999626	0.594058	0.331644	-0.73812	0.999411	0.602946	0.331545	-0.73801	0.999312
8	Happy	0.569475	0.384455	-0.98686	0.99965	0.582425	0.332417	-0.91566	0.999443	0.594354	0.331891	-0.9156	0.999351
9	Happy	0.502169	0.38584	-1.00886	0.999673	0.535798	0.333409	-0.95467	0.999466	0.551155	0.335398	-0.95454	0.999379
10	Happy	0.448748	0.384622	-0.76511	0.999683	0.487588	0.333017	-0.75906	0.999461	0.505982	0.335702	-0.75884	0.999381
11	Happy	0.440112	0.385714	-0.79016	0.999685	0.475068	0.334596	-0.77582	0.999431	0.492769	0.337911	-0.77566	0.999358
12	Happy	0.443117	0.386781	-0.79634	0.999679	0.477767	0.334884	-0.80813	0.99938	0.495941	0.338184	-0.80796	0.999314
13	Happy	0.458964	0.387007	-0.83149	0.999681	0.491137	0.335608	-0.82042	0.999344	0.508477	0.339101	-0.8203	0.999289
14	Happy	0.484354	0.389063	-0.97611	0.999701	0.514425	0.337066	-0.91759	0.999381	0.532041	0.34001	-0.91763	0.999328
15	Happy	0.571521	0.386686	-0.96273	0.99967	0.571447	0.334426	-0.8598	0.999336	0.580813	0.335719	-0.86003	0.999288
16	Happy	0.594254	0.386949	-0.7319	0.99967	0.589876	0.33472	-0.66589	0.999347	0.59714	0.336054	-0.66585	0.999297
17	Happy	0.59752	0.387078	-0.70511	0.999669	0.593856	0.334938	-0.64717	0.999356	0.6011	0.336311	-0.6471	0.999304
18	Happy	0.584233	0.401835	-0.86961	0.999683	0.586766	0.345494	-0.8022	0.999385	0.595796	0.345673	-0.80218	0.999336
19	Happy	0.582142	0.435397	-0.86576	0.999696	0.589635	0.373466	-0.80178	0.999414	0.599453	0.373238	-0.80177	0.999365
20	Happy	0.585539	0.444225	-0.87673	0.999701	0.594191	0.383041	-0.81475	0.999429	0.603683	0.38274	-0.81472	0.999377
21	Happy	0.58602	0.445794	-0.83535	0.999707	0.595418	0.38597	-0.77433	0.99945	0.605022	0.385708	-0.77432	0.999394
22	Happy	0.579956	0.447949	-0.829	0.999727	0.594252	0.388493	-0.77095	0.999486	0.602957	0.388358	-0.77093	0.999432
23	Happy	0.504199	0.445374	-0.86883	0.999741	0.538646	0.387808	-0.82221	0.999509	0.553332	0.387066	-0.82225	0.999453
24	Happy	0.458368	0.441288	-0.78826	0.999732	0.496823	0.383685	-0.77895	0.999478	0.515011	0.384665	-0.77887	0.999412
25	Happy	0.437448	0.43302	-0.70155	0.999741	0.476269	0.376034	-0.69831	0.999488	0.495177	0.37784	-0.69826	0.999424
26	Happy	0.429043	0.431004	-0.70596	0.999737	0.46552	0.373534	-0.70354	0.999467	0.483726	0.375241	-0.7035	0.999399
27	Happy	0.452654	0.428046	-0.94022	0.999745	0.48073	0.371153	-0.89245	0.999473	0.49695	0.372773	-0.89252	0.999409
28	Happy	0.538177	0.427397	-0.94665	0.999758	0.536776	0.367233	-0.88448	0.999505	0.548476	0.367588	-0.88451	0.999445

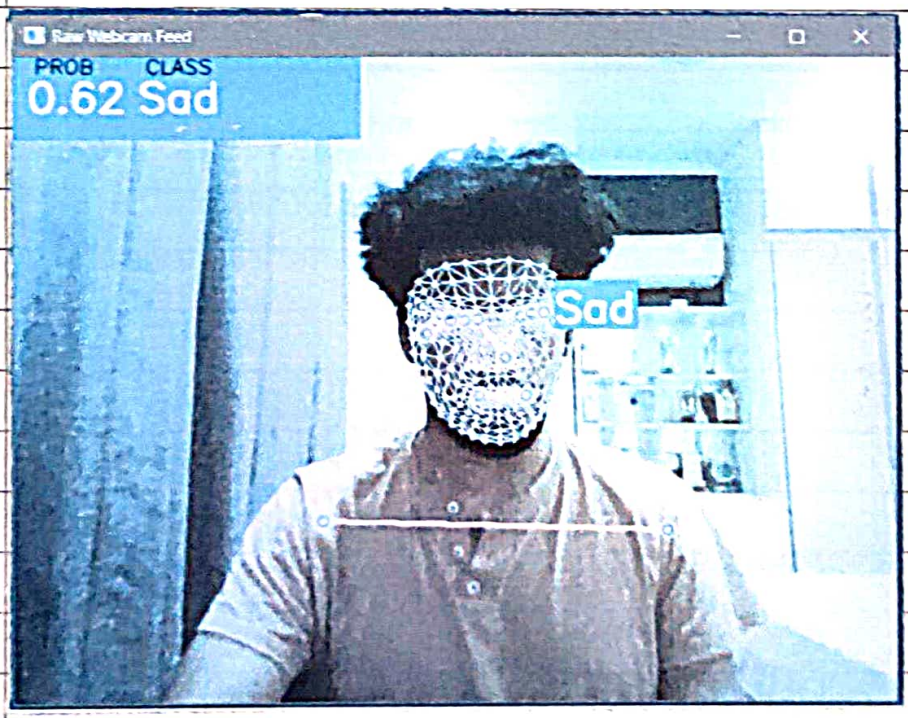
Entire Project Link

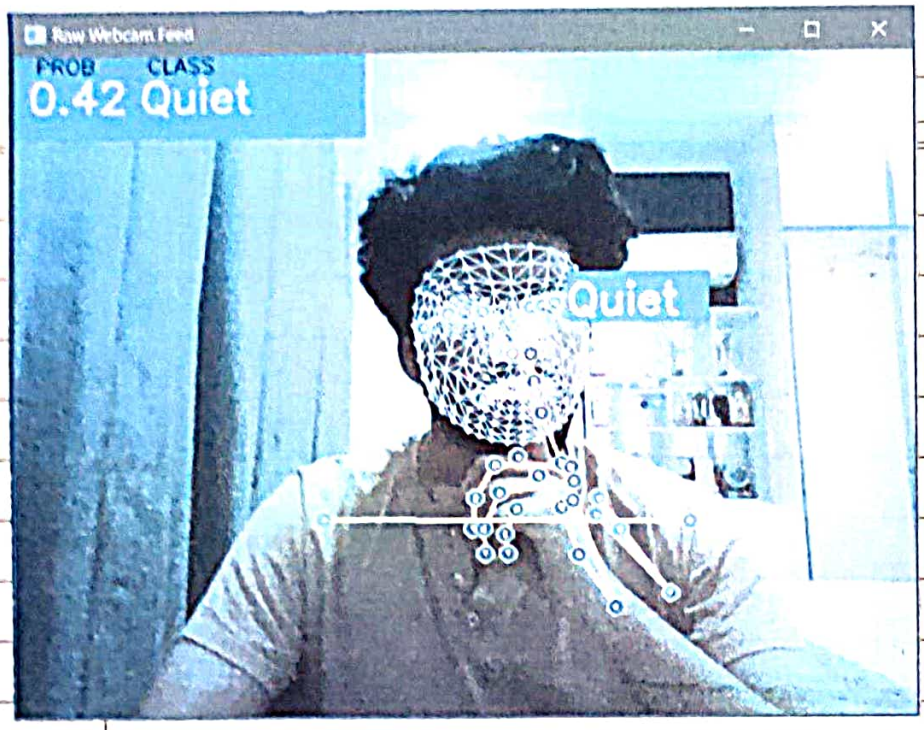
ThePunisher04/Human-Body-Language-Detector- (github.com)

DELTA®

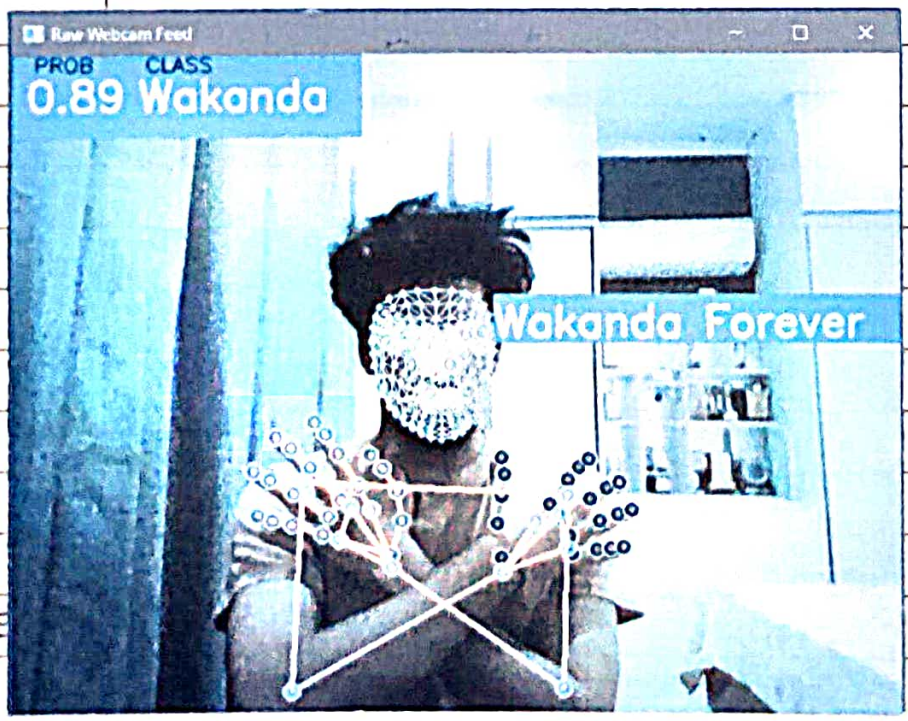
Live Body Language Detection Results with probability






Raw Webcam Feed
PROB CLASS
0.42 Quiet
Quiet


Raw Webcam Feed
PROB CLASS
0.94 Victorious
Victorious


Raw Webcam Feed
PROB CLASS
0.89 Wakanda
Wakanda Forever