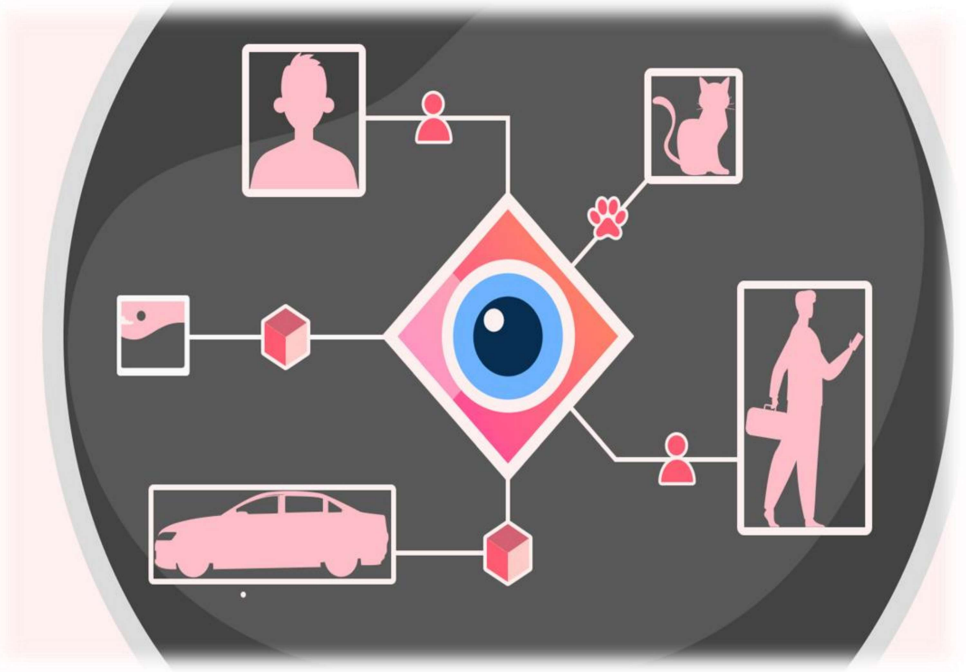
INTRODUCTION

Hey, I'm **Harsh Raj**, and this document is a journey through the projects that paved my way to success at DataIntelliage. In each project, I've gone beyond the surface, diving deep into data analysis, machine learning, and computer vision. This isn't just a showcase; it's the story of how these projects secured my spot in the DataIntelliage interview. Get ready for a quick tour of innovation and problem-solving!



Project Documentation

--Multi-Camera Face Recognition System --

1. Project Overview and Libraries Used

1.1 Libraries Used

This project utilizes several external libraries to achieve its functionality:

- **OpenCV (cv2):** Used for capturing video feeds, image processing, and computer vision tasks.
- **Requests:** Enables communication with a smartphone camera via HTTP requests to obtain video frames.
- **NumPy:** Facilitates array manipulation for efficient image processing.
- **Dlib:** Employs a frontal face detector for face recognition in captured frames.
- **Imutils:** Provides convenience functions for resizing and processing frames.
- **Time:** Manages time-related operations, such as cooldown periods and timestamping.
- **CSV:** Handles the writing of information to a CSV file for logging purposes.

1.2 Project Creation

The project involves the integration of two video feeds (webcam and smartphone camera) to detect and track faces in real-time. The following steps outline the project's structure:

1. **Camera Initialization:**
    - Initialize the webcam and smartphone camera using OpenCV.
2. **Face Detection:**
    - Utilize Dlib's frontal face detector to identify faces in both camera feeds.
3. **Data Logging:**
    - Record relevant information, including camera source, person identifier, count, and timestamp, to a CSV file ('info.csv').
4. **Real-time Display:**
    - Display the processed frames with face bounding boxes, counts, and timestamps for both camera feeds using OpenCV.
5. **User Interruption:**
    - Allow the user to exit the application by pressing 'q.'

2. Project Functionality and Application

2.1 Project Description

This project combines the feeds from a webcam and a smartphone camera to detect and track faces in real-time. Each recognized face is assigned a unique identifier, and relevant information is logged to a CSV file. The project is structured as follows:

- **Webcam Feed:**
  - Faces detected in the webcam feed are assigned identifiers starting with "camera1_person" followed by a unique number.
  - Information such as person identifier, count, and timestamp is logged to 'info.csv.'
  - Bounding boxes are drawn around detected faces, displaying count and timestamp information.
- **Phone Camera Feed:**
  - Faces detected in the smartphone camera feed are assigned identifiers starting with "camera2_person" followed by a unique number.
  - Information is logged to 'info.csv' similarly to the webcam feed.
  - Bounding boxes are drawn around detected faces, displaying count and timestamp information.

2.2 Everyday Application

This project finds application in scenarios requiring real-time face detection and tracking from multiple camera sources. Potential use cases include:

- **Security Systems:**
  - Surveillance systems that employ multiple cameras to monitor and track individuals.
- **Attendance Tracking:**
  - Efficiently tracking and recording attendance in educational or corporate settings.
- **Access Control:**
  - Implementing face recognition for secure access to facilities or devices.
- **Event Management:**
  - Monitoring and managing crowds during events for safety and security.

"Source Code Unveiled: A Comprehensive Look into Implementation"

```
import cv2
import requests
import numpy as np
import dlib
from imutils import face_utils
import time
import csv
detector = dlib.get_frontal_face_detector()
cap_webcam = cv2.VideoCapture(0)
url_phone = 'http://(ip address):8080/video'
cap_phone = cv2.VideoCapture(url_phone)
if not cap_webcam.isOpened() or not cap_phone.isOpened():
    print("Error: Could not open one or both cameras.")
    exit()
width = 400
height = 400
detected_faces_webcam = {}
count_webcam = 0
detected_faces_phone = {}
count_phone = 0
cooldown_time = 10
with open('info.csv', 'w', newline='') as csvfile:
    fieldnames = ['Camera', 'Person', 'Count', 'Time']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
while True:
        ret_webcam, frame_webcam = cap_webcam.read()
        response = requests.get(url_phone)
```

```
    frame_phone = cv2.imdecode(np.array(bytearray(response.content),
dtype=np.uint8), -1)
print(f"ret_webcam: {ret_webcam}, frame_webcam shape: {frame_webcam.shape}")
    print(f"response status: {response.status_code}, frame_phone shape:
{frame_phone.shape}")
if not ret_webcam or response.status_code != 200:
        print("Error: Could not read one or both frames.")
        break
    frame_phone = cv2.resize(frame_phone, (width, height))
    gray_webcam = cv2.cvtColor(frame_webcam, cv2.COLOR_BGR2GRAY)
    gray_phone = cv2.cvtColor(frame_phone, cv2.COLOR_BGR2GRAY)
    faces_webcam = detector(gray_webcam)
    faces_phone = detector(gray_phone)
    current_time = time.time()
    for i, face in enumerate(faces_webcam):
        (x, y, w, h) = face_utils.rect_to_bb(face)
        face_name = f"camera1_person{i + 1}"
        if i not in detected_faces_webcam or (current_time -
detected_faces_webcam[i]['last_update']) >= cooldown_time:
            detected_faces_webcam[i] = {'name': face_name, 'count': count_webcam,
'last_update': current_time}
            count_webcam += 1
            writer.writerow({'Camera': 'Webcam', 'Person': face_name, 'Count':
count_webcam, 'Time': time.ctime(current_time)})
        cv2.rectangle(frame_webcam, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame_webcam, f"{detected_faces_webcam[i]['name']}:
Count={detected_faces_webcam[i]['count']} Time={time.ctime(current_time)}",
                    (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    for i, face in enumerate(faces_phone):
        (x, y, w, h) = face_utils.rect_to_bb(face)
        face_name = f"camera2_person{i + 1}"
```

```
        if i not in detected_faces_phone or (current_time - detected_faces_phone[i]['last_update']) >= cooldown_time:
            detected_faces_phone[i] = {'name': face_name, 'count': count_phone, 'last_update': current_time}
            count_phone += 1
            writer.writerow({'Camera': 'Phone', 'Person': face_name, 'Count': count_phone, 'Time': time.ctime(current_time)})
        cv2.rectangle(frame_phone, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame_phone, f"{detected_faces_phone[i]['name']}: Count={detected_faces_phone[i]['count']} Time={time.ctime(current_time)}",
                    (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    cv2.imshow('Webcam Feed', frame_webcam)
    cv2.imshow('Phone Camera Feed', frame_phone)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap_webcam.release()
cap_phone.release()
cv2.destroyAllWindows()
```

-- RealMask Guardian --

1. Project Overview and Libraries Used

1.1 Used Libraries

This project leverages the power of several key libraries:

- **OpenCV (cv2):** Utilized for image processing, face detection, and video stream handling.
- **TensorFlow and Keras:** Employed for loading a pre-trained MobileNetV2 model and making predictions.
- **Imutils:** Facilitates resizing and handling of video streams efficiently.
- **NumPy:** Enables array manipulation for streamlined image processing.
- **Time:** Manages time-related operations for synchronization and timestamping.

1.2 Project Development

The project is built to detect face masks in real-time video streams. It utilizes a pre-trained face detection model alongside a MobileNetV2 model for mask predictions. The implementation includes:

- Initialization of face detection and mask prediction models.
- Integration of video streams from the webcam using OpenCV and Imutils.
- Real-time face detection and mask prediction using TensorFlow and Keras.
- Display of results, including bounding boxes and labels, in the video stream.

2. Project Functionality and Application

2.1 Project Overview

This project aims to detect individuals wearing face masks in a real-time video stream. It utilizes a pre-trained face detection model to locate faces and a MobileNetV2 model to predict whether a person is wearing a mask or not. The core steps include:

1. **Face Detection:** Utilizes a pre-trained face detection model to identify faces in each frame.
2. **Mask Prediction:** Applies a MobileNetV2 model to predict whether a detected face is wearing a mask.

3. **Result Display:** Displays bounding boxes and labels in the video stream based on mask predictions.

2.2 Real-world Application

This project finds practical applications in scenarios requiring real-time monitoring of individuals wearing face masks. Key use cases include:

- **Health and Safety:** Ensuring compliance with face mask guidelines in public spaces.
- **Security Systems:** Integration into security systems for access control.
- **Public Events:** Monitoring face mask usage during events for safety.

"Source Code Unveiled: A Comprehensive Look into Implementation"

```
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os
def detect_and_predict_mask(frame, faceNet, maskNet):
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224), (104.0, 177.0, 123.0))
	faceNet.setInput(blob)
	detections = faceNet.forward()
	faces = []
	locs = []
	preds = []
	for i in range(0, detections.shape[2]):
		confidence = detections[0, 0, i, 2]
```

```
            if confidence > 0.5:
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
                (startX, startY) = (max(0, startX), max(0, startY))
                (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
                face = frame[startY:endY, startX:endX]
                face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                face = cv2.resize(face, (224, 224))
                face = img_to_array(face)
                face = preprocess_input(face)
                faces.append(face)
                locs.append((startX, startY, endX, endY))
    if len(faces) > 0:
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)
    return (locs, preds)
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
maskNet = load_model("mask_detector.model")
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
while True:
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
    for (box, pred) in zip(locs, preds):
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred
```

```
                label = "Mask" if mask > withoutMask else "No Mask"
                color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
                label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
                cv2.putText(frame, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
                cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
                break
cv2.destroyAllWindows()
vs.stop()
```

--Movie Recommendor System --

1. Project Overview and Libraries Used

1.1 Libraries Used

- **NumPy (np):** Fundamental package for scientific computing.
- **Pandas (pd):** Data manipulation and analysis library.
- **OS:** Operating system interface for file operations.
- **Ast (ast):** Abstract Syntax Tree module for evaluating literal expressions.
- **CountVectorizer from sklearn.feature_extraction.text:** Converts text to a matrix of token counts.
- **cosine_similarity from sklearn.metrics.pairwise:** Calculates cosine similarity.
- **Pickle:** Serializes and deserializes objects for data storage.

1.2 Project Creation

- **Data Loading:** Loading movie dataset from Kaggle using Pandas.
- **Data Merging:** Merging movie data with credits data on the 'title' column.
- **Column Selection:** Keeping relevant columns for analysis.
- **Data Processing:** Converting textual data and handling missing values.
- **Feature Engineering:** Extracting information from text and credits.
- **Text Vectorization:** Utilizing CountVectorizer for text-to-matrix conversion.
- **Similarity Calculation:** Computing cosine similarity for movie vectors.
- **Model Persistence:** Storing processed data and similarity matrix with Pickle.
- **Recommendation Function:** Providing movie recommendations based on similarity.
- **Example Usage:** Demonstrating the system with a sample movie, 'Gandhi.'

2. Project Functionality and Daily Life Application

2.1 Project Description

- **Data Processing:** Extracting and structuring movie information.
- **Text Vectorization:** Converting textual data into numerical vectors.
- **Similarity Calculation:** Utilizing cosine similarity for movie comparisons.
- **Recommendation:** Suggesting movies based on input title and similarity scores.

2.2 Daily Life Application

- **Personalized Entertainment:** Tailoring movie recommendations to user preferences.
- **Enhanced User Experience:** Improving satisfaction on streaming platforms.
- **Time Efficiency:** Efficiently discovering new movies, saving user time.

By understanding user preferences and content similarity, the project contributes to a more enjoyable and personalized movie-watching experience.

"Source Code Unveiled: A Comprehensive Look into Implementation"

```
import numpy as np
import pandas as pd
import os
import ast
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pickle
movies = pd.read_csv('/kaggle/input/tmdb-movie-metadata/tmdb_5000_movies.csv')
credits = pd.read_csv('/kaggle/input/tmdb-movie-metadata/tmdb_5000_credits.csv')
movies = movies.merge(credits, on='title')
movies = movies[['movie_id', 'title', 'overview', 'genres', 'keywords', 'cast', 'crew']]
def convert(text):
  return [i['name'] for i in ast.literal_eval(text)]
def fetch_director(text):
  return [i['name'] for i in ast.literal_eval(text) if i['job'] == 'Director']
def collapse(L):
  return [i.replace(" ","") for i in L]
movies.dropna(inplace=True)
movies['genres'] = movies['genres'].apply(convert)
movies['keywords'] = movies['keywords'].apply(convert)
movies['cast'] = movies['cast'].apply(convert)
```

```
movies['cast'] = movies['cast'].apply(lambda x: x[0:3])
movies['crew'] = movies['crew'].apply(fetch_director)
movies['crew'] = movies['crew'].apply(collapse)
new = movies.drop(columns=['overview', 'genres', 'keywords', 'cast', 'crew'])
new['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']
new['tags'] = new['tags'].apply(lambda x: " ".join(x))
cv = CountVectorizer(max_features=5000, stop_words='english')
vector = cv.fit_transform(new['tags']).toarray()
similarity = cosine_similarity(vector)
pickle.dump(new, open('movie_list.pkl', 'wb'))
pickle.dump(similarity, open('similarity.pkl', 'wb'))
def recommend(movie):
   index = new[new['title'] == movie].index[0]
   distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda x: x[1])
   recommended_movies = [new.iloc[i[0]].title for i in distances[1:6]]
   return recommended_movies
recommendations = recommend('Gandhi')
print("Recommended Movies for 'Gandhi':", recommendations)
```

 -- GuardianFace: Instant Face Recognition & Alerts --

1. Project Overview and Libraries Used

1.1 Libraries Used

This project leverages the following external libraries to implement its functionalities:

- **OpenCV (cv2):** Employed for video capturing, image processing, and real-time computer vision tasks.
- **NumPy:** Utilized for efficient array manipulations, crucial for image processing operations.
- **face_recognition:** A powerful library for face recognition, providing facial detection and encoding capabilities.
- **Pushbullet:** Utilized for sending notifications to a smartphone device.

1.2 Project Creation

This project integrates OpenCV, NumPy, face_recognition, and Pushbullet to accomplish real-time face detection and recognition. The essential steps in creating this project include:

- **Camera Initialization:**
  - Initialize the webcam using OpenCV for video capture.
- **Face Recognition:**
  - Utilize the face_recognition library to detect and recognize faces in real-time.
- **Notification Sending:**
  - Send notifications to a smartphone using Pushbullet when a specific face is detected.

2. Project Functionality and Application

2.1 Project Description

This project focuses on real-time face detection and recognition using a webcam. The key components and functionalities are:

- **Face Detection:**

    - Identify faces in the webcam feed using the face_recognition library.
- **Real-time Recognition:**
    - Recognize known faces (e.g., "Harsh") based on pre-encoded reference images.
- **Notification System:**
    - Send notifications to a smartphone via Pushbullet when a specific face (e.g., "Harsh") is detected.

2.2 Everyday Application

This project finds practical applications in various scenarios, including:

- **Security and Monitoring Systems:**
    - Implementing real-time face recognition for enhanced security and surveillance.
- **Personalized Notifications:**
    - Notifying users when specific individuals (e.g., family members) are detected.
- **Access Control:**
    - Enhancing access control systems by recognizing authorized individuals.

"Source Code Unveiled: A Comprehensive Look into Implementation"

```
import cv2
import numpy as np
import face_recognition
from pushbullet import Pushbullet
PUSHBULLET_API_KEY = "YOUR_API_KEY"
known_face_encodings = []
known_face_names = ["Harsh"]
pb = Pushbullet(PUSHBULLET_API_KEY)
def send_notification(message):
  pb.push_note("Face Detection", message)
harsh_image = face_recognition.load_image_file("path/to/reference_image.jpg")
```

```
harsh_face_encoding = face_recognition.face_encodings(harsh_image)[0]
known_face_encodings.append(harsh_face_encoding)
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    rgb_frame = frame[:, :, ::-1]
    face_locations = face_recognition.face_locations(rgb_frame)
    face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)
    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
        matches = face_recognition.compare_faces(known_face_encodings,
face_encoding)
        name = "Unknown"
        if True in matches:
            first_match_index = matches.index(True)
            name = known_face_names[first_match_index]
        cv2.rectangle(frame, (left, top), (right, bottom), (255, 0, 0), 2)
        cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255,
0, 0), 2)
        if name == "Harsh":
            send_notification("Harsh face detected!")
    cv2.imshow('Face Recognition', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

-- Real-time Gender Detection --

1. Project Overview and Libraries Used

1.1 Libraries Used

This project leverages the following libraries to achieve its functionality:

- **OpenCV (cv2):** Employed for capturing video feeds, image processing, and real-time face detection.
- **face_recognition:** Utilized for face detection and recognition tasks.
- **deepface:** Integrated for additional facial analysis, including gender detection.

1.2 Project Creation

The project is designed to perform real-time face detection and gender classification. Key steps in the project's construction include:

1. **Face Detection:**
    - Utilizes the `face_recognition` library to identify face locations in each frame.
    - Extracts detected faces and performs gender analysis using the `deepface` library.
2. **Gender Classification:**
    - DeepFace is employed to analyze each detected face for gender, providing gender scores.
3. **Visualization:**
    - Draws rectangles around detected faces on the frame.
    - Displays the gender label with corresponding confidence scores.
4. **Real-time Processing:**
    - The project operates continuously, processing frames in real-time from the webcam.

2. Project Functionality and Application

2.1 Project Description

This project combines the capabilities of OpenCV, face_recognition, and deepface to achieve real-time face detection and gender classification. The primary features include:

- **Face Detection:**
  - Detects faces in each video frame, outlining them with rectangles.
- **Gender Classification:**
  - Utilizes deepface to analyze facial features and classify gender.
- **Real-time Display:**
  - Displays the processed frames with detected faces and gender labels.

2.2 Everyday Application

The project finds applications in scenarios requiring real-time face detection and gender classification. Potential uses include:

- **Security Systems:**
  - Enhances surveillance systems with gender classification for improved monitoring.
- **Audience Analytics:**
  - Provides valuable insights in audience composition during events or presentations.
- **User Profiling:**
  - Useful for user-specific interactions or experiences based on detected gender.

"Source Code Unveiled: A Comprehensive Look into Implementation"

```
import cv2
import face_recognition
from deepface import DeepFace
def detect_faces(frame):
  face_locations = face_recognition.face_locations(frame)
  for (top, right, bottom, left) in face_locations:
    face_image = frame[top:bottom, left:right]
    result = DeepFace.analyze(face_image, actions=['gender'],
enforce_detection=False)
    gender_scores = result[0]['gender']
    gender_label = max(gender_scores, key=gender_scores.get)
    cv2.rectangle(frame, (left, top), (right, bottom), (255, 0, 0), 2)
```

```
        cv2.putText(frame, f'Gender: {gender_label}', (left, top - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
    return frame
def main():
    cap = cv2.VideoCapture(0)
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            print("Error reading frame from webcam")
            break
        processed_frame = detect_faces(frame)
        cv2.imshow('Face Detection with Gender', processed_frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
if __name__ == "__main__":
    main()
```