**Overview**

This report provides an in-depth look at the design and implementation of a Vehicle Safety Assessment System, which comprises three core modules:

1. **Number Plate OCR**
   * Purpose: Accurately extract license plate characters from images.
   * Techniques: Image preprocessing (contrast enhancement, denoising, thresholding), sharpening, and Tesseract OCR with custom configs.
2. **Model Training**
   * Purpose: Learn patterns in historical traffic violation data to classify vehicle safety.
   * Techniques: Data cleaning, feature engineering, label encoding, ensemble learning with Random Forests, hyperparameter tuning, performance evaluation.
3. **AI Prediction**
   * Purpose: Integrate OCR output with the trained classifier to compute a real-time safety probability.
   * Techniques: Database lookup, feature encoding, probability extraction, risk interpretation.

Additionally, we cover:

* **Dependencies & Environment Setup**
* **Detailed Data Schema & Feature Overview**
* **Error Handling & Logging Strategies**
* **Deployment & Future Enhancements**

**Dependencies & Environment Setup**

* **Language & Frameworks**: Python 3.8+, OpenCV, pytesseract, scikit-learn, pandas, NumPy, joblib, requests
* **External Tools**: Tesseract OCR v5.x installed at C:\Program Files\Tesseract-OCR\tesseract.exe
* **Directory Structure**:
* project\_root/
* ├── data.csv
* ├── encoders/ # Saved LabelEncoders (.pkl)
* ├── vehicle\_safety\_model.pkl
* ├── model\_features.pkl
* └── main.py # Unified entry-point script
* **Installation**:
* pip install opencv-python pytesseract numpy pandas scikit-learn joblib requests

**Data Schema & Feature Overview**

The data.csv file holds historical records of vehicles and their violation history. Key columns:

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Number Plate | string | Unique license plate identifier |
| Vehicle Type | category | e.g., Car, Truck, Motorcycle |
| Owner Age | integer | Driver/owner age in years |
| License Type | category | e.g., Learner, Permanent, Commercial |
| Violation Type | category | e.g., Speeding, Signal Violation, Overloading |
| Severity | category | e.g., Minor, Major, Critical |
| Repeated Offense | category | Yes/No |
| Total Violations | integer | Count of past violations |
| Days Since Last Violation | integer | Time span from last violation (in days) |
| Weather | category | e.g., Clear, Rainy, Foggy |
| Safe | category | Target label: Safe (1) or Unsafe (0) |

Continuous features (Owner Age, Total Violations, Days Since Last Violation) are used directly; categorical features are label-encoded.

**1. Number Plate OCR (Detailed)**

**Workflow Steps**

1. **Image Retrieval**
   * Use requests.get(url) with a 5-second timeout to fetch PNG/JPG bytes.
   * Validate status code and content length.
2. **Image Decoding**
   * Convert raw bytes into a NumPy array and decode with cv2.imdecode to BGR color.
3. **Grayscale Conversion**  
   Simplifies color channels into intensity values:
4. gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)
5. **Contrast Enhancement**  
   Amplifies pixel intensity differences:
6. gray = cv2.convertScaleAbs(gray, alpha=1.8, beta=0)
   * *alpha* (>1) increases contrast; *beta* adds a bias.
7. **Noise Reduction**  
   Smooths out minor artifacts:
8. gray = cv2.GaussianBlur(gray, (5, 5), 0)
9. **Adaptive Thresholding**  
   Binarizes under varying illumination:
10. thresholded = cv2.adaptiveThreshold(
11. gray, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,
12. cv2.THRESH\_BINARY, blockSize=11, C=2)
    * *blockSize* defines local neighborhood; *C* subtracts from weighted mean.
13. **Sharpening**  
    Enhances character edges with a high-pass filter:
14. kernel = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
15. sharpened = cv2.filter2D(thresholded, -1, kernel)
16. **OCR Configuration & Execution**
    * **Page Segmentation Mode (PSM)**: 7 (single text line)
    * **OCR Engine Mode (OEM)**: 3 (LSTM only)
    * **Whitelist**: Alphanumerics to reduce misclassifications.
17. config = '--psm 7 --oem 3 ' \
18. '-c tessedit\_char\_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
19. text = pytesseract.image\_to\_string(sharpened, config=config)
20. **Post-processing**
    * Strip spaces/newlines: clean = text.strip().replace("\n", "")
    * Remove unexpected characters via regex:
21. import re
22. clean = re.sub(r'[^A-Z0-9]', '', clean)

# Full OCR function

import cv2, pytesseract, numpy as np, requests, re

def detect\_plate\_from\_url(image\_url: str) -> str:

resp = requests.get(image\_url, timeout=5)

resp.raise\_for\_status()

arr = np.frombuffer(resp.content, np.uint8)

img = cv2.imdecode(arr, cv2.IMREAD\_COLOR)

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

gray = cv2.convertScaleAbs(gray, alpha=1.8, beta=0)

gray = cv2.GaussianBlur(gray, (5,5), 0)

thresh = cv2.adaptiveThreshold(

gray, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

cv2.THRESH\_BINARY, 11, 2)

kernel = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])

sharp = cv2.filter2D(thresh, -1, kernel)

cfg = ('--psm 7 --oem 3 ' \

'-c tessedit\_char\_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')

raw = pytesseract.image\_to\_string(sharp, config=cfg)

cleaned = re.sub(r'[^A-Z0-9]', '', raw)

return cleaned

**2. Model Training (In-Depth)**

**Data Preprocessing & Encoding**

1. **Load CSV & Cleanup**:
2. df = pd.read\_csv('data.csv')
3. df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
4. df.ffill(inplace=True)
5. **Label Encoding**: Convert 7 categorical fields + target.
6. cats = ['Vehicle Type', 'License Type', 'Violation Type',
7. 'Severity', 'Weather', 'Repeated Offense']
8. for col in cats + ['Safe']:
9. le = LabelEncoder()
10. df[col] = le.fit\_transform(df[col])
11. joblib.dump(le, f'encoders/{col}.pkl')
12. **Feature Engineering (Optional)**:
    * **Normalization** of continuous variables (e.g., Owner Age) via MinMax or StandardScalers.
    * **Derived Features**: Violation frequency = Total Violations / Days Since Last Violation.

**Train/Test Split & Validation**

* **70/30 split**, random\_state=42 to ensure reproducibility.
* **Stratify** on Safe to maintain class balance.

X = df[features]

y = df['Safe']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, stratify=y, random\_state=42)

**Model Selection & Hyperparameters**

* **Algorithm**: Random Forest (robust to outliers, handles categorical via encoding).
* **Key Hyperparameters**:
  + n\_estimators=100 (number of trees)
  + max\_depth=None (full growth)
  + min\_samples\_split=2

*Note*: You can further tune with grid search over n\_estimators, max\_depth, and min\_samples\_leaf.

**Training & Performance**

model = RandomForestClassifier(

n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

**Evaluation Metrics** on test set:

* **Accuracy**: Overall correct classification percentage.
* **Precision/Recall/F1** for each class (Safe vs. Unsafe).

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

Feature importance ranking (to interpret model):

importances = model.feature\_importances\_

for feat, imp in sorted(zip(features, importances), key=lambda x: -x[1]):

print(f"{feat}: {imp:.3f}")

**Artifact Persistence**

joblib.dump(model, 'vehicle\_safety\_model.pkl')

joblib.dump(features, 'model\_features.pkl')

**3. AI Prediction & Integration**

**Lookup & Encoding**

1. **Reload Data** (same cleaning as training).
2. **Filter** for Number Plate; if missing, return an informative error.
3. **Extract Features** into a dict.
4. **Load Encoders & Model**, and transform each category. Handle unseen categories with -1 (optional one-hot fallback).

**Probability Extraction**

* Random Forest’s predict\_proba returns [P(Unsafe), P(Safe)] when classes are [0, 1].
* We take index 1 for the Safe class.

proba = model.predict\_proba(X\_input)[0][1]

**Risk Interpretation**

| **Score Range** | **Category** |
| --- | --- |
| 90% - 100% | Very Safe |
| 75% - 89% | Safe |
| 60% - 74% | Moderately Safe |
| 40% - 59% | Risky |
| 0% - 39% | Dangerous |

**4. Unified Backend Script**

Below is the consolidated main.py with robust error handling and logging:

import cv2, pytesseract, numpy as np, requests, re

import pandas as pd, joblib, logging

# Configure logging

logging.basicConfig(

filename='app.log', level=logging.INFO,

format='%(asctime)s %(levelname)s:%(message)s')

pytesseract.pytesseract.tesseract\_cmd = (

r"C:\Program Files\Tesseract-OCR\tesseract.exe")

# 1. OCR Function (as above)

...

# 2. Safety Score Function

def get\_safety\_score(plate: str) -> float:

df = pd.read\_csv('data.csv')

df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

df.ffill(inplace=True)

record = df[df['Number Plate'] == plate]

if record.empty:

logging.warning(f"Plate {plate} not found in data.")

return None

features = record.iloc[0][feature\_list].to\_dict()

model = joblib.load('vehicle\_safety\_model.pkl')

encs = {c: joblib.load(f'encoders/{c}.pkl') for c in cat\_columns}

# Encoding

encoded = {}

for c, v in features.items():

if c in encs:

try:

encoded[c] = encs[c].transform([v])[0]

except ValueError:

encoded[c] = -1

else:

encoded[c] = v

inp = pd.DataFrame([encoded]).reindex(columns=feature\_list, fill\_value=0)

return model.predict\_proba(inp)[0][1]

# 3. Main Execution

def main():

url = input("Enter image URL: ").strip()

try:

plate = detect\_plate\_from\_url(url)

logging.info(f"OCR result: {plate}")

except Exception as e:

logging.error(f"OCR failed: {e}")

print("Failed to extract plate. See log for details.")

return

score = get\_safety\_score(plate)

if score is None:

print(f"Vehicle {plate} not found.")

else:

print(f"Safety Score for {plate}: {score:.0%}")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Deployment & Future Enhancements**

* **Containerization**: Dockerize with a slim Python image; include Tesseract.
* **API Layer**: Expose endpoints /ocr and /predict via Flask or FastAPI.
* **Plate Localization**: Use YOLO or OpenALPR to first detect plate region, then OCR.
* **Continuous Learning**: Log new violations and periodically retrain the model.
* **Security & Privacy**: Encrypt data at rest, secure API via OAuth/JWT.