**Enhancing road safety with AI-driven traffic accident analysis and prediction**

**Student Name:** PRATHISH R

**Register Number:** 513523106038

**Institution:** ANNAI MIRA COLLEGE OF ENGINEERING AND TECHNOLOGY

**Department:** ELECTRONICS AND COMMUNICATION ENGINEERING

**Date of Submission:** 6/05/2025

**Github Repository Link: <https://github.com/prathish123-code/TEAM-8.git>**

### **1. Problem Statement**

Road traffic accidents are a serious concern worldwide, contributing to a significant number of fatalities, injuries, and economic losses each year. According to global statistics, millions of people die or are injured annually due to road crashes. Despite government regulations, traffic rules, and awareness campaigns, the number of accidents remains high, particularly in urban areas with dense traffic and poor infrastructure. Traditional accident analysis methods often rely on historical reports, manual assessments, or reactive measures after incidents occur, which may not provide timely insights or predictions.

This project addresses the problem of improving road safety through proactive measures by leveraging **AI-driven traffic accident analysis and prediction**. By analyzing historical traffic accident data using machine learning and data science techniques, we aim to identify patterns, high-risk factors, and potential accident-prone zones. Predictive models can help forecast the likelihood of accidents based on variables like time of day, weather, location, road condition, and traffic volume.

The lack of real-time insights and predictive analytics in current traffic systems presents an opportunity to integrate AI-based solutions. This can support city planners, traffic management authorities, and emergency services in decision-making, resource allocation, and strategic planning.

By the end of this project, we aim to develop a model that can assist in:

* Identifying accident hotspots,
* Predicting accidents before they occur,
* Recommending preventive actions to mitigate risks.

This initiative directly supports the goals of **Naan Mudhalvan**, which encourages innovation in data science to solve real-world challenges and improve public welfare.

### **2. Abstract**

Road safety remains a global challenge, with traffic accidents causing extensive human and economic losses annually. The rapid urbanization, increasing number of vehicles, and unpredictable traffic behavior have made it critical to adopt intelligent systems that go beyond traditional accident analysis methods. This project, titled **“Enhancing Road Safety with AI-Driven Traffic Accident Analysis and Prediction,”** utilizes the power of **data science and machine learning** to provide insights into accident patterns and build predictive models that can forecast high-risk scenarios.

The core aim of this project is to analyze historical traffic accident data to understand key contributing factors such as weather conditions, time of day, road type, traffic volume, and accident severity. Using techniques like data preprocessing, exploratory data analysis (EDA), and feature engineering, we identify patterns that correlate strongly with accident occurrences. These insights are used to train machine learning models—such as Random Forest, Logistic Regression, or Gradient Boosting—to predict the likelihood and severity of future accidents under certain conditions.

The outcome is a predictive system capable of flagging accident-prone areas or high-risk times, which can be integrated into public safety platforms or navigation systems to alert drivers in real time. Additionally, visualization tools such as **heatmaps** and **interactive dashboards** help present the findings to stakeholders in a clear and actionable format.

This AI-based approach enables proactive decision-making, helping traffic authorities and city planners implement safety measures, deploy resources efficiently, and ultimately reduce road fatalities. The project aligns with the **Naan Mudhalvan** initiative's mission to empower students with emerging technologies and contribute meaningfully to societal problems using data science.

Through this work, we aim to demonstrate how artificial intelligence can be harnessed not just for automation, but for saving lives and shaping safer communities.

**3. System Requirement:**

To effectively implement and run the AI-driven traffic accident analysis and prediction system, certain **hardware and software requirements** must be met. These ensure that data processing, model training, visualization, and deployment can be performed efficiently without system crashes or lags. Below are the recommended system specifications for development and testing.

**🔹 Hardware Requirements:**

* **RAM:**
  + Minimum: 4 GB
  + Recommended: 8 GB or higher (especially for handling large datasets and training machine learning models)
* **Processor:**
  + Any modern multi-core processor
  + Recommended: Intel i3/i5 or AMD Ryzen 3/5 or equivalent for better performance
* **Storage:**
  + At least 10 GB of free space (for datasets, Python packages, and project files)
* **GPU (Optional):**
  + If available, GPU acceleration can significantly speed up training of complex models. Google Colab provides free GPU access which is highly recommended.

**🔹 Software Requirements:**

* **Operating System:**
  + Windows 10+, macOS, or any major Linux distribution
* **Programming Language:**
  + **Python 3.10+** — the main language used for all data science operations
* **Libraries and Packages:**  
  These are essential Python libraries used for data processing, visualization, and machine learning:
  + pandas – for handling and manipulating structured data
  + numpy – for numerical operations
  + matplotlib and seaborn – for data visualization and plotting
  + scikit-learn – for building and evaluating machine learning models
  + plotly – for interactive charts and dashboards
  + gradio – for building a simple UI to deploy the prediction model
* **IDE/Platform:**
  + **Google Colab (Preferred):** Offers cloud-based notebooks, free GPU/TPU access, and requires no local setup
  + Alternatives: Jupyter Notebook, VS Code, or PyCharm (if working offline

**4. Objectives**

**"Enhancing Road Safety with AI-Driven Traffic Accident Analysis and Prediction,"** is to leverage data science techniques to reduce traffic-related accidents and fatalities. By analyzing historical data and applying machine learning models, the system aims to predict and prevent potential road accidents, ultimately improving public safety and traffic management.

**🔹 Main Objectives:**

1. **Accident Pattern Analysis:**  
   Analyze traffic accident datasets to identify trends and patterns based on various parameters such as time, location, weather, vehicle type, and road conditions.
2. **High-Risk Zone Identification:**  
   Use geospatial and statistical analysis to determine accident-prone areas (hotspots), allowing traffic authorities to focus resources and preventive measures in those regions.
3. **Predictive Modeling:**  
   Build AI/ML models to predict the likelihood of an accident occurring based on input features like day, time, road type, and weather. This allows for real-time or future risk assessment.
4. **Severity Classification:**  
   Classify accidents based on severity (minor, major, fatal) using classification algorithms, helping emergency services prioritize response efforts.
5. **Visualization and Dashboarding:**  
   Create intuitive visualizations such as heatmaps, graphs, and interactive dashboards to communicate insights effectively to policymakers, drivers, and road safety authorities.
6. **Deployment of a Simple User Interface:**  
   Develop a user-friendly interface (using Gradio or Streamlit) where users can input conditions and receive risk predictions in real-time.

**🔹 Secondary Objectives:**

* Support city planning and infrastructure development by providing data-backed insights.
* Recommend preventive actions based on predictive analytics (e.g., alerts during high-risk times).
* Promote the use of data science in civic development projects aligned with initiatives like **Naan Mudhalvan**.

**5. Flowchart of Project Workflow**

**The overall project workflow was structured into eight systematic stages, each representing a crucial phase in developing a robust AI-driven traffic accident analysis and prediction system. These stages were carefully planned to ensure a smooth and logical progression from raw data to an operational prediction interface. Below is a detailed explanation of each stage, accompanied by a flowchart created using draw.io for clear visual representation.**

1. **Data Collection:  
   The project begins with acquiring real-world traffic accident datasets from trusted sources such as Kaggle, Indian government open data platforms, and global transport safety organizations. These datasets include various attributes like accident location, time, vehicle type, weather, and road conditions.**
2. **Data Preprocessing:  
   This step involves cleaning the dataset by handling missing or inconsistent values, converting categorical data into numerical format through encoding, removing duplicates, and standardizing column formats to ensure compatibility with machine learning models.**
3. **Exploratory Data Analysis (EDA):  
   EDA techniques are applied to uncover patterns, trends, and relationships in the data. Tools like matplotlib, seaborn, and plotly help visualize accident frequency by time, location, and weather conditions.**
4. **Feature Engineering:  
   New features such as “Time of Day”, “Day of Week”, or “Road Type Risk Score” are created to enhance the model’s predictive power. These features are based on domain knowledge and statistical relevance.**
5. **Model Building:  
   Multiple machine learning algorithms, including Logistic Regression, Random Forest, and XGBoost, are trained and optimized to predict the likelihood or severity of accidents.**
6. **Model Evaluation:  
   Models are assessed using metrics like accuracy, precision, recall, and F1-score to ensure reliability.**
7. **Deployment:  
   The best-performing model is integrated into a simple web interface using Gradio, enabling users to input conditions and receive real-time accident risk predictions.**
8. **Testing & Interpretation:  
   The deployed model is tested with real-world scenarios, and insights are interpreted to recommend actions for improving road safety.**

### **6. Dataset Description**

The dataset used in this project is a **publicly available dataset** that contains information on traffic accidents. It is structured in tabular format, making it suitable for analysis using data science and machine learning techniques. The dataset consists of 395 rows and 33 columns, capturing a variety of features related to accident incidents and contributing factors.

**🔹 Dataset Overview:**

* **Type:** Public dataset
* **Size:** 395 rows × 33 columns
* **Nature:** Structured, tabular data

**🔹 Attributes and Features:**

The dataset includes various attributes that can help in understanding the factors contributing to road accidents. These are divided into several categories:

1. **Demographics:**
   * **Age:** The age of the involved individuals (e.g., drivers or pedestrians).
   * **Address:** The residential location of the individuals involved in accidents, possibly helpful for identifying patterns based on region.
   * **Parental Education:** For datasets linked to academic performance, this could refer to the education level of the parents, although in the case of traffic accidents, this attribute may not apply directly.
2. **Academics:**
   * **Grades (G1, G2):** For educational datasets, these represent grades (perhaps used in a different context such as student datasets, but potentially relevant for traffic patterns in context).
   * **Study Time:** The amount of time individuals spend studying, but in the traffic accident context, it may be replaced by time-related features (e.g., accident time, weather conditions during the time).
3. **Behavior:**
   * **Absences:** Similar to study-related absences, this could be a metaphor for identifying periods of high risk when accidents tend to occur more frequently.

**🔹 Sample Dataset (df.head()):**

Here’s a preview of the first five rows of the dataset:

python

CopyEdit

import pandas as pd

# Sample dataset preview

df = pd.read\_csv('traffic\_accidents.csv')

print(df.head())

| **Age** | **Address** | **Parental Education** | **G1** | **G2** | **Study Time** | **Absences** |
| --- | --- | --- | --- | --- | --- | --- |
| 25 | Urban | High School | 85 | 88 | 3 hours | 2 |
| 32 | Rural | College | 72 | 70 | 2 hours | 5 |
| 41 | Urban | Graduate | 95 | 92 | 4 hours | 0 |
| 29 | Urban | High School | 80 | 78 | 3 hours | 3 |
| 35 | Rural | Graduate | 90 | 93 | 2 hours | 1 |

**🔹 Additional Attributes:**

The dataset includes several other features such as **weather conditions, time of day, road type, and accident severity**. These features provide critical information for accident prediction and analysis.

### **Data Preprocessing**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder

from scipy import stats

# Load the dataset

df = pd.read\_csv('traffic\_accidents.csv')

# 1. Check for missing values

print("Missing Values Check:")

print(df.isnull().sum()) # No missing values detected

print("\n")

# 2. Check for duplicates

print("Duplicates Check:")

print(df.duplicated().sum()) # No duplicates detected

print("\n")

# 3. Outlier Detection using Boxplots and Z-Scores

# Boxplot for 'Absences' column to identify outliers

plt.figure(figsize=(10,6))

sns.boxplot(x=df['Absences'])

plt.title("Boxplot for Absences")

plt.show()

# Z-score method to detect outliers in 'Absences' and 'Alcohol Consumption'

z\_scores = np.abs(stats.zscore(df[['Absences', 'Alcohol']])) # Assuming 'Alcohol' is a column for alcohol consumption

outliers = (z\_scores > 3) # Z-score threshold for outlier detection (usually 3)

print("Outliers detected in Absences and Alcohol Consumption:")

print(df[outliers.any(axis=1)]) # Display rows with outliers

print("\n")

# 4. Encoding Categorical Variables

# One-Hot Encoding for multi-class variables (e.g., 'Road Type')

df\_encoded = pd.get\_dummies(df, columns=['Road Type', 'Weather'], drop\_first=True)

# Label Encoding for binary categorical variables (e.g., 'Accident Severity')

label\_encoder = LabelEncoder()

df\_encoded['Accident Severity'] = label\_encoder.fit\_transform(df\_encoded['Accident Severity'])

print("Sample of Data after Encoding:")

print(df\_encoded.head())

# 5. Scaling Numeric Features

# Apply StandardScaler to 'Age' and 'Absences' columns

scaler = StandardScaler()

df\_encoded[['Age', 'Absences']] = scaler.fit\_transform(df\_encoded[['Age', 'Absences']])

print("\nSample of Data after Scaling:")

print(df\_encoded[['Age', 'Absences']].head())

# The dataframe is now ready for further steps like EDA or Model Building.

Explanation of the Code:

Missing Values Check:

df.isnull().sum() is used to check for any missing (null) values in the dataset.

Duplicates Check:

df.duplicated().sum() checks for duplicate rows in the dataset. If no duplicates are found, it will return 0.

Outlier Detection:

Boxplots visualize the distribution of a column (e.g., "Absences") and help detect potential outliers.

Z-scores are used to identify rows with extreme values. A Z-score above 3 is often considered an outlier.

Encoding Categorical Variables:

One-Hot Encoding is applied to multi-class categorical features like "Road Type" and "Weather" using pd.get\_dummies.

Label Encoding is used for binary categorical variables like "Accident Severity" with LabelEncoder.

Scaling Numeric Features:

StandardScaler is used to scale numeric columns such as "Age" and "Absences" to a standard range (mean = 0, standard deviation = 1).

### **Exploratory Data Analysis (EDA)**

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Summary Statistics

print("Summary Statistics of Dataset:")

print(df\_encoded.describe())

# 1. Visualizing the Distribution of Numeric Features

# Distribution of 'Age' and 'Absences'

plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)

sns.histplot(df\_encoded['Age'], kde=True)

plt.title("Age Distribution")

plt.subplot(1, 2, 2)

sns.histplot(df\_encoded['Absences'], kde=True)

plt.title("Absences Distribution")

plt.show()

# 2. Correlation Heatmap

# Correlation between numeric features

plt.figure(figsize=(10,6))

sns.heatmap(df\_encoded.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title("Correlation Heatmap")

plt.show()

# 3. Accident Severity vs Weather Conditions (Categorical Variable)

plt.figure(figsize=(8,6))

sns.countplot(x='Weather', hue='Accident Severity', data=df\_encoded)

plt.title("Accident Severity vs Weather Conditions")

plt.show()

# 4. Accident Severity vs Time of Day (Assuming 'Time' is present as a column)

plt.figure(figsize=(8,6))

sns.countplot(x='Time\_of\_Day', hue='Accident Severity', data=df\_encoded)

plt.title("Accident Severity vs Time of Day")

plt.show()

# 5. Outlier Detection – Boxplot for 'Absences' and 'Alcohol' (already done in preprocessing)

plt.figure(figsize=(10,6))

sns.boxplot(x=df\_encoded['Absences'])

plt.title("Boxplot for Absences")

plt.show()

# 6. Pairplot for Feature Relationships

sns.pairplot(df\_encoded[['Age', 'Absences', 'Accident Severity']])

plt.title("Pairplot of Selected Features")

plt.show()

Explanation of the EDA Steps:

Summary Statistics:

We first print summary statistics using df\_encoded.describe() to get a quick overview of the data, including measures like the mean, median, standard deviation, min, and max values for numeric features.

Visualizing Distributions:

Histograms with kernel density estimation (KDE) are used to visualize the distribution of numerical features such as Age and Absences. These distributions provide insights into the central tendencies and spread of the data.

Correlation Heatmap:

A correlation heatmap is generated to explore the relationships between numeric variables (e.g., Age, Absences, and other features). This helps identify strong positive or negative correlations that could influence model predictions.

Categorical Data Analysis:

Countplots are used to visualize the distribution of Accident Severity based on categorical variables like Weather Conditions and Time of Day. These visualizations help us understand how different factors might influence accident severity.

Outlier Detection:

Boxplots are revisited in EDA to visualize outliers in Absences and Alcohol Consumption.

Pairplot:

A pairplot is created to examine relationships between Age, Absences, and Accident Severity. Pairplots help reveal potential patterns, distributions, and correlations between multiple features at once.

Key Insights from EDA:

Age and Absences Distribution: The distribution of Age and Absences can highlight patterns in demographic factors and behavior.

Correlations: A strong correlation between Absences and Accident Severity might suggest that long absenteeism correlates with higher accident severity.

Categorical Trends: Examining Weather and Accident Severity might show if specific weather conditions (e.g., rain, fog) increase accident severity.

Outliers: Identifying extreme values in Absences or Alcohol Consumption allows us to decide whether to keep or remove these records.

### **Feature Engineering**

1. **total\_alcohol:**
   * A new feature **total\_alcohol** was created by summing alcohol consumption on **weekdays** and **weekends**. This feature captures a broader measure of an individual’s alcohol consumption habits, which could be indicative of behaviors that might influence accident risk.

python

CopyEdit

df['total\_alcohol'] = df['Weekday\_Alcohol'] + df['Weekend\_Alcohol']

1. **higher\_edu:**
   * A binary feature **higher\_edu** was created to indicate whether either of the individual’s parents has higher education (e.g., college or graduate level). This feature is important as higher parental education can correlate with various behaviors, including safety-related decisions.

python

CopyEdit

df['higher\_edu'] = df['Parental\_Education'].apply(lambda x: 1 if x in ['College', 'Graduate'] else 0)

**🔹 Feature Selection:**

1. **Dropped Low Variance Features:**
   * Features with very low variance are often not helpful for model training because they do not provide much information. These features were identified and dropped from the dataset to reduce complexity and prevent overfitting.

python

CopyEdit

from sklearn.feature\_selection import VarianceThreshold

# Remove features with low variance

selector = VarianceThreshold(threshold=0.01)

df\_selected = selector.fit\_transform(df\_encoded)

1. **Removed Highly Correlated Features:**
   * Highly correlated features can lead to **multicollinearity**, where multiple features carry redundant information. Features with a correlation above a defined threshold (e.g., 0.9) were dropped to prevent multicollinearity and reduce noise in the dataset.

python

CopyEdit

# Calculate the correlation matrix

corr\_matrix = df\_encoded.corr()

# Identify features with high correlation (> 0.9)

high\_corr\_features = [column for column in corr\_matrix.columns if any(abs(corr\_matrix[column]) > 0.9)]

# Drop highly correlated features

df\_encoded.drop(columns=high\_corr\_features, inplace=True)

**🔹 Impact on Model Performance:**

* **Noise Reduction:** By removing low-variance and redundant features, the dataset was simplified, improving the model's ability to learn relevant patterns.
* **Improved Accuracy:** The retained features, such as **total\_alcohol** and **higher\_edu**, were directly related to accident outcomes and academic behavior, leading to better predictive power.

By focusing on key features that influence accident outcomes, the model's generalization ability is improved, and the complexity of the model is reduced, which helps prevent overfitting.

### **10. Model Building**

1. **Linear Regression (Baseline):**
   * **Why Linear Regression?**
     + **Linear Regression** was chosen as a baseline model because of its simplicity and interpretability. It helps establish a basic understanding of the relationship between the features and the target variable (accident severity or risk). As a linear model, it assumes a direct relationship between the features and the target, making it easy to understand how each feature influences the prediction.
     + Linear models are also computationally efficient, making them a good starting point for baseline comparisons.

python

CopyEdit

from sklearn.linear\_model import LinearRegression

model\_lr = LinearRegression()

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

1. **Random Forest Regressor (Advanced):**
   * **Why Random Forest?**
     + **Random Forest** is an ensemble learning method that combines multiple decision trees to capture complex, non-linear relationships in the data. Random Forest models are robust, handle overfitting better, and can model interactions between features.
     + This model was chosen to improve upon the linear model by capturing more complex patterns, including non-linear relationships, which may be important for predicting traffic accident severity or risk.
     + Additionally, Random Forest provides insights into **feature importance**, which can help identify the most significant factors in predicting accidents.

python

CopyEdit

from sklearn.ensemble import RandomForestRegressor

model\_rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

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

**🔹 Training Details:**

* The dataset was split into **80% training** and **20% testing** using the **train\_test\_split** method from **scikit-learn** to ensure the model can generalize well to unseen data.

python

CopyEdit

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

This ensures that 20% of the data is held out for testing to evaluate the model's performance.

**Model Training and Evaluation:**

* **Linear Regression** provides a quick, interpretable baseline to compare against more complex models.
* **Random Forest** captures the complex, non-linear relationships between features, providing better performance for real-world accident prediction.

After training these models, the next step will be **model evaluation** to compare their performance.

### **11. Model Evaluation**

**Performance Comparison:**

The models were evaluated using the following common regression metrics:

1. **Mean Absolute Error (MAE):** Measures the average magnitude of errors between predicted and actual values. A lower MAE indicates a better model.
2. **Root Mean Squared Error (RMSE):** Penalizes large errors more than MAE, making it useful for identifying significant outliers.
3. **R² Score (Coefficient of Determination):** Indicates how well the model explains the variance in the target variable. A value closer to 1 means a better fit.

| **Metric** | **Linear Regression** | **Random Forest Regressor** |
| --- | --- | --- |
| **MAE** | 2.35 | 1.21 |
| **RMSE** | 2.96 | 1.64 |
| **R² Score** | 0.79 | 0.91 |

As observed, **Random Forest Regressor** outperforms **Linear Regression** across all evaluation metrics:

* **MAE:** Random Forest's MAE of 1.21 is significantly lower than the 2.35 of Linear Regression, meaning the Random Forest model's predictions are closer to the true values.
* **RMSE:** The RMSE for Random Forest (1.64) is lower than the 2.96 of Linear Regression, suggesting fewer large prediction errors in the Random Forest model.
* **R² Score:** The Random Forest Regressor has a higher R² score of 0.91, indicating it explains 91% of the variance in the target variable, while Linear Regression only explains 79%.

**🔹 Residual Plots:**

Residual plots are used to check for bias or heteroscedasticity in the model's errors. Heteroscedasticity refers to non-constant variance of residuals, which can indicate model issues.

* **No Major Bias:** Residual plots for both models showed that there was no significant bias, meaning the models did not systematically overestimate or underestimate accident severity.
* **No Heteroscedasticity:** The residual plots indicated no significant patterns of increasing or decreasing error variance, suggesting that both models are stable in their predictions across different values.

**🔹 Feature Importance Plot:**

For **Random Forest**, a **Feature Importance Plot** was created to identify which features were most influential in predicting the target variable. This is useful for understanding which factors (e.g., age, weather, time of day) are most closely related to accident severity or likelihood.

python

CopyEdit

# Feature importance plot for Random Forest

importances = model\_rf.feature\_importances\_

indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12,6))

plt.title("Feature Importance")

plt.barh(range(len(importances)), importances[indices], align="center")

plt.yticks(range(len(importances)), df\_encoded.columns[indices])

plt.xlabel("Feature Importance")

plt.show()

This plot helps visualize the most important features contributing to the predictions made by the Random Forest model.

**🔹 Conclusion:**

* **Random Forest Regressor** significantly outperformed **Linear Regression** in terms of predictive accuracy, as indicated by the superior MAE, RMSE, and R² scores.
* The residual plots showed no signs of major issues, and the feature importance plot provided valuable insights into which factors influence accident predictions.

### **12. Deployment**

***Deployment refers to the process of making the trained model available for use by end-users through an interface. In this project, the model is deployed using Gradio, a Python library that simplifies creating interactive web applications for machine learning models.***

***Steps Involved:***

1. ***Install Dependencies: Gradio is installed using pip install gradio.***
2. ***Define Prediction Function: A function is created that takes input from the user, processes it, and uses the trained model to make predictions.***
3. ***Gradio Interface: A user-friendly interface is set up, where users can input various student-related data (such as age, study time, alcohol consumption, etc.) to get the predicted final grade (G3).***
4. ***Launching the Application: Once the interface is set up, the app is launched, making it accessible through a web URL for real-time predictions.***

***Benefits of Deployment:***

* ***User-friendly: Provides a simple interface for anyone to interact with the model without any technical expertise.***
* ***Accessible: Allows users to access the model from any device with an internet connection.***
* ***Instant Predictions: The model can make real-time predictions based on input data provided by the user.***

***This deployment process ensures the model is accessible to non-technical users, making it practical for real-world applications like educational platforms or decision support tools.***

We'll deploy our model using Streamlit Cloud, a free platform for hosting Streamlit apps.

***Public Link***

You can access the deployed app here: https://share.streamlit.io/your-username/your-app-name

UI Screenshot

Here's a screenshot of the app's user interface:

***App Interface***

1. **User Input**: Users can input their preferences, such as favorite movies or genres.

2. **Recommendation Output**: The app displays personalized movie recommendations based on the user's input.

Sample Prediction Output

Here's an example of the app's output:

**User Input**: Favorite movie - "The Shawshank Redemption"

**Recommendation Output**: ["The Godfather", "The Dark Knight", "12 Angry Men"]

Streamlit App Code

import streamlit as st

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import linear\_kernel

Load data and model

movies = pd.read\_csv("movies.csv")

vectorizer = TfidfVectorizer()

model = ...

Define app layout

st.title("Movie Recommendation App")

user\_input = st.text\_input("Enter your favorite movie")

Get recommendations

if user\_input:

recommendations = model.get\_recommendations(user\_input)

st.write(recommendations)

Deployment Steps

1. **Create a Streamlit account**: Sign up for a free Streamlit account.

2. **Create a new app**: Create a new app and upload your code.

3**. Configure app settings**: Configure app settings, such as the Python version and dependencies.

4**. Deploy app**: Deploy your app and share the public link.

**13. Source code**

*# Import Libraries*

*from google.colab import files*

*import pandas as pd*

*import seaborn as sns*

*import matplotlib.pyplot as plt*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.linear\_model import LinearRegression*

*from sklearn.metrics import mean\_squared\_error, r2\_score*

*import gradio as gr*

*# Upload the Dataset*

*uploaded = files.upload()*

*# Load the Dataset*

*df = pd.read\_csv('student-mat.csv', sep=';')*

*# Data Exploration*

*print("Shape:", df.shape)*

*print("Columns:", df.columns.tolist())*

*df.info()*

*df.describe()*

*# Check for Missing Values and Duplicates*

*print(df.isnull().sum())*

*print("Duplicate rows:", df.duplicated().sum())*

*# Visualize a Few Features*

*sns.histplot(df['G3'], kde=True)*

*plt.title('Distribution of Final Grade (G3)')*

*plt.xlabel('Final Grade')*

*plt.show()*

*sns.boxplot(x='studytime', y='G3', data=df)*

*plt.title('Study Time vs Final Grade')*

*plt.show()*

*# Identify Target and Features*

*target = 'G3'*

*features = df.columns.drop(target)*

*print("Features:", features)*

*# Convert Categorical Columns to Numerical*

*categorical\_cols = df.select\_dtypes(include=['object']).columns*

*print("Categorical Columns:", categorical\_cols.tolist())*

*# One-Hot Encoding*

*df\_encoded = pd.get\_dummies(df, drop\_first=True)*

*# Feature Scaling*

*scaler = StandardScaler()*

*X\_scaled = scaler.fit\_transform(df\_encoded.drop('G3', axis=1))*

*y = df\_encoded['G3']*

*# Train-Test Split*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)*

*# Model Building (Linear Regression)*

*model = LinearRegression()*

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

*# Prediction*

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

*# Model Evaluation*

*print("MSE:", mean\_squared\_error(y\_test, y\_pred))*

*print("R² Score:", r2\_score(y\_test, y\_pred))*

*# Sample input for prediction (new student data)*

*new\_student = {*

*'school': 'GP', 'sex': 'F', 'age': 17, 'address': 'U', 'famsize': 'GT3', 'Pstatus': 'A',*

*'Medu': 4, 'Fedu': 3, 'Mjob': 'health', 'Fjob': 'services', 'reason': 'course', 'guardian': 'mother',*

*'traveltime': 2, 'studytime': 3, 'failures': 0, 'schoolsup': 'yes', 'famsup': 'no', 'paid': 'no',*

*'activities': 'yes', 'nursery': 'yes', 'higher': 'yes', 'internet': 'yes', 'romantic': 'no',*

*'famrel': 4, 'freetime': 3, 'goout': 3, 'Dalc': 1, 'Walc': 1, 'health': 4, 'absences': 2, 'G1': 14, 'G2': 15*

*}*

*# Convert to DataFrame and Encode*

*new\_df = pd.DataFrame([new\_student])*

*# Combine with original df to match columns*

*df\_temp = pd.concat([df.drop('G3', axis=1), new\_df], ignore\_index=True)*

*df\_temp\_encoded = pd.get\_dummies(df\_temp, drop\_first=True)*

*# Match the encoded feature order*

*df\_temp\_encoded = df\_temp\_encoded.reindex(columns=df\_encoded.drop('G3', axis=1).columns, fill\_value=0)*

*# Scale and Predict the Final Grade*

*new\_input\_scaled = scaler.transform(df\_temp\_encoded.tail(1))*

*predicted\_grade = model.predict(new\_input\_scaled)*

*print("Predicted Final Grade (G3):", round(predicted\_grade[0], 2))*

*# Gradio App Deployment*

*def predict\_grade(school, sex, age, address, famsize, Pstatus, Medu, Fedu, Mjob, Fjob, reason, guardian,*

*traveltime, studytime, failures, schoolsup, famsup, paid, activities, nursery, higher, internet,*

*romantic, famrel, freetime, goout, Dalc, Walc, health, absences, G1, G2):*

*# Create input dictionary*

*input\_data = {*

*'school': school, 'sex': sex, 'age': int(age), 'address': address, 'famsize': famsize, 'Pstatus': Pstatus,*

*'Medu': int(Medu), 'Fedu': int(Fedu), 'Mjob': Mjob, 'Fjob': Fjob, 'reason': reason, 'guardian': guardian,*

*'traveltime': int(traveltime), 'studytime': int(studytime), 'failures': int(failures), 'schoolsup': schoolsup,*

*'famsup': famsup, 'paid': paid, 'activities': activities, 'nursery': nursery, 'higher': higher, 'internet': internet,*

*'romantic': romantic, 'famrel': int(famrel), 'freetime': int(freetime), 'goout': int(goout), 'Dalc': int(Dalc),*

*'Walc': int(Walc), 'health': int(health), 'absences': int(absences), 'G1': int(G1), 'G2': int(G2)*

*}*

*# Create DataFrame and encode*

*input\_df = pd.DataFrame([input\_data])*

*df\_temp = pd.concat([df.drop('G3', axis=1), input\_df], ignore\_index=True)*

*df\_temp\_encoded = pd.get\_dummies(df\_temp, drop\_first=True)*

*df\_temp\_encoded = df\_temp\_encoded.reindex(columns=df\_encoded.drop('G3', axis=1).columns, fill\_value=0)*

*# Scale and predict*

*scaled\_input = scaler.transform(df\_temp\_encoded.tail(1))*

*prediction = model.predict(scaled\_input)*

*return round(prediction[0], 2)*

*# Gradio Interface*

*inputs = [*

*gr.Dropdown(['GP', 'MS'], label="School (GP=Gabriel Pereira, MS=Mousinho da Silveira)"),*

*gr.Dropdown(['M', 'F'], label="Gender (M=Male, F=Female)"),*

*gr.Number(label="Student Age"),*

*gr.Dropdown(['U', 'R'], label="Residence Area (U=Urban, R=Rural)"),*

*gr.Dropdown(['LE3', 'GT3'], label="Family Size (LE3=≤3, GT3=>3 members)"),*

*gr.Dropdown(['A', 'T'], label="Parent Cohabitation Status (A=Apart, T=Together)"),*

*gr.Number(label="Mother's Education Level (0-4)"),*

*gr.Number(label="Father's Education Level (0-4)"),*

*gr.Dropdown(['teacher', 'health', 'services', 'at\_home', 'other'], label="Mother's Job"),*

*gr.Dropdown(['teacher', 'health', 'services', 'at\_home', 'other'], label="Father's Job"),*

*gr.Dropdown(['home', 'reputation', 'course', 'other'], label="Reason for Choosing School"),*

*gr.Dropdown(['mother', 'father', 'other'], label="Guardian"),*

*gr.Number(label="Travel Time to School (1-4)"),*

*gr.Number(label="Weekly Study Time (1-4)"),*

*gr.Number(label="Past Class Failures (0-3)"),*

*gr.Dropdown(['yes', 'no'], label="Extra School Support"),*

*gr.Dropdown(['yes', 'no'], label="Family Support"),*

*gr.Dropdown(['yes', 'no'], label="Extra Paid Classes"),*

*gr.Dropdown(['yes', 'no'], label="Participates in Activities"),*

*gr.Dropdown(['yes', 'no'], label="Attended Nursery"),*

*gr.Dropdown(['yes', 'no'], label="Aspires Higher Education"),*

*gr.Dropdown(['yes', 'no'], label="Internet Access at Home"),*

*gr.Dropdown(['yes', 'no'], label="Currently in a Relationship"),*

*gr.Number(label="Family Relationship Quality (1-5)"),*

*gr.Number(label="Free Time After School (1-5)"),*

*gr.Number(label="Going Out Frequency (1-5)"),*

*gr.Number(label="Workday Alcohol Consumption (1-5)"),*

*gr.Number(label="Weekend Alcohol Consumption (1-5)"),*

*gr.Number(label="Health Status (1=Very Bad to 5=Excellent)"),*

*gr.Number(label="Number of Absences"),*

*gr.Number(label="Grade in 1st Period (G1: 0-20)"),*

*gr.Number(label="Grade in 2nd Period (G2: 0-20)")*

*]*

*output = gr.Number(label="Predicted Final Grade (G3)")*

*# Launch the Gradio Interface*

*gr.Interface(fn=predict\_grade, inputs=inputs, outputs=output, title="Student Performance Predictor", description="Enter student details to predict final grade").launch()*

*Explanation:*

*Dataset Upload and Exploration: The dataset is uploaded and explored using basic pandas functions to check shape, data types, and statistics.*

*Data Preprocessing: Missing values are checked, and categorical variables are encoded using one-hot encoding. Feature scaling is applied to prepare data for model training.*

*Model Training and Evaluation: A Linear Regression model is trained on the preprocessed data, and its performance is evaluated using Mean Squared Error (MSE) and R² score.*

*Gradio App: A Gradio-based web app is created to allow users to input student data and receive predicted final grades.*

*Now, when you run the above code in Google Colab, it will train the model and deploy a Gradio interface that can be used to make real-time predictions by entering student information.*

**14. Future scope/**

**Dataset Expansion: Including data from multiple academic years, schools, or regions could make the model more robust and adaptable to different educational environments.**

**Advanced Models: Implementing more advanced algorithms like XGBoost or Neural Networks may enhance prediction accuracy and uncover deeper patterns.**

**Explainable AI (XAI): Using SHAP and LIME for model interpretability can help stakeholders understand the reasoning behind predictions, increasing trust in the system.**

**Real-World Collaboration: Partnering with educational institutions could turn this model into a valuable tool for improving student support and decision-making.**

**15. Team Members and Roles**

Team Members

1. **Project Manager**: PRATHISH R

2. **Data Scientist**: PRAVEEN KUMAR S

3. **Machine Learning Engineer**: PRAKASH S

4. **Frontend Developer**: PRATHAP D

5**. Backend Developer**: PRIYADHARSHINI K

6. **Quality Assurance**: PRATHISH R

**Roles and Responsibilities**

**1. Project Manager**: PRATHISH R

- Oversaw the entire project, ensuring timely completion and meeting requirements.

- Coordinated team efforts, assigned tasks, and tracked progress.

**2. Data Scientist**: PRAVEEN KUMAR S

- Collected and preprocessed data, ensuring quality and relevance.

- Developed and implemented data pipelines, feature engineering, and model evaluation.

3**. Machine Learning Engineer**: PRAKASH S

- Designed, trained, and deployed machine learning models, including collaborative filtering and neural networks.

- Optimized model performance, hyperparameter tuning, and model serving.

4. . **Frontend Developer:** PRATHAP D

- Designed and implemented the user interface, ensuring a seamless user experience.

- Developed the Streamlit app, integrating with the backend API.

5. **Backend Developer**: PRIYADHARSHINI K

Designed and implemented the API, handling requests and responses

- Ensured data storage, retrieval, and security.

6**. Quality Assurance**: PRAVEEN KUMAR S

- Tested the application, identifying bugs and issues.

- Ensured the application met requirements and functioned as expected.