**PROJECT REPORT**ON

**"ADVANCED TRAFFIC VOLUME EASTIMATION"**

**Submitted By:**

* Team ID: PNT2025TMID02151
* Team Size: 4

**Team Members:**

* Team Leader: Devratsinh Chauhan
* Team Member: Zeel Patel
* Team Member: Sonu Patel
* Team Member: Mitesh Purohit

**Submitted To:**

Smart Bridge & Smart Interns

**Date of Submission:** 14/03/2025

**INTRODUCTION**

Traffic congestion is a significant challenge in urban areas, leading to increased travel time, fuel consumption, and environmental pollution. Accurate traffic volume estimation is essential for efficient traffic management, urban planning, and the optimization of transportation infrastructure. Traditional methods for estimating traffic volume rely on manual data collection, road sensors, or surveillance cameras, which are often costly and limited in scalability.

With advancements in machine learning and data analytics, traffic volume prediction can be significantly improved by utilizing historical traffic data, weather conditions, time-based patterns, and other influencing factors. This project, TrafficTelligence, aims to develop an intelligent system for traffic volume estimation using machine learning techniques. By leveraging data-driven approaches, we seek to enhance accuracy, reduce dependency on manual methods, and provide actionable insights for urban mobility planning.

**PROBLEM STATEMENT**

Traffic congestion and unpredictable road conditions affect millions of commuters daily, leading to increased delays, fuel wastage, and environmental concerns. Traditional traffic volume estimation methods often lack accuracy, real-time adaptability, and cost efficiency. The key challenges include:

* Lack of real-time and scalable traffic volume estimation methods.
* Influence of external factors like weather, holidays, and time of day on traffic patterns.
* Limited accessibility and high costs associated with traditional data collection techniques.
* Difficulty in predicting future traffic trends for efficient transportation planning.

This project aims to address these challenges by utilizing machine learning models trained on historical traffic data to predict traffic volume accurately. The developed system will enable better traffic forecasting, aiding city planners, commuters, and policymakers in making informed

**TECHNOLOGY**

|  |  |  |
| --- | --- | --- |
| SR NO. | TOOL & TECHNOLOGY | VERSION |
| 1 | Windows 11 | 11 23H2 |
| 2 | Python | 3.12.5 |
| 3 | Excel | 2021 |
| 4 | Pandas | 2.2.2 |
| 5 | NumPy | 1.20.0 |
| 6 | Jupyter Notebook | 7.1 |
| 7 | Html | Html5 |
| 8 | Css | Css3 |
| 9 | Flask | 3.1.0 |

**DATA COLLECTION**

The dataset is obtained from Smart Bridge, which provides data related to traffic volume estimation.

The data can be accessed through the given Google Drive link.

It likely includes historical traffic patterns, weather conditions, and time-based attributes.

The dataset can be accessed using the following link:

<https://drive.google.com/file/d/1iV5PfYAmI6YP0_0S4KYy1ZahHOqMgDbM/view>

**Dataset Overview**

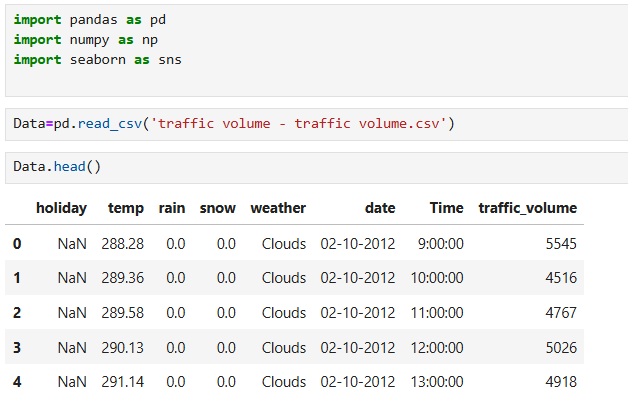
The dataset consists of the following columns:

* holiday: Indicates whether the day is a holiday (missing values suggest non-holidays).
* temp: Temperature (likely in Kelvin).
* rain: Amount of rainfall (in mm or cm).
* snow: Amount of snowfall.
* weather: Weather conditions (e.g., Clouds).
* date: Date of the observation.
* Time: Time of the observation.
* traffic\_volume: Number of vehicles recorded.

**DATA CLEANING AND PREPROCESSING**

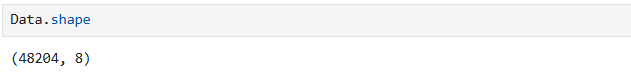
**1. Reads the Dataset**

* pandas (pd): Used for data manipulation and analysis.
* numpy (np): Used for numerical computations.
* seaborn (sns): Used for data visualization.
* pd.read\_csv('traffic volume - traffic volume.csv'): Reads the dataset from a CSV file.
* Data.head(): Displays the first five rows of the dataset.

****

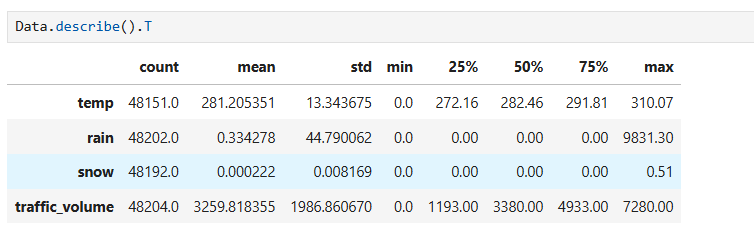
**2. Checking the Dataset Shape**

The output of the command Data.shape is (48204, 8), which provides information about the dataset's dimensions**.**

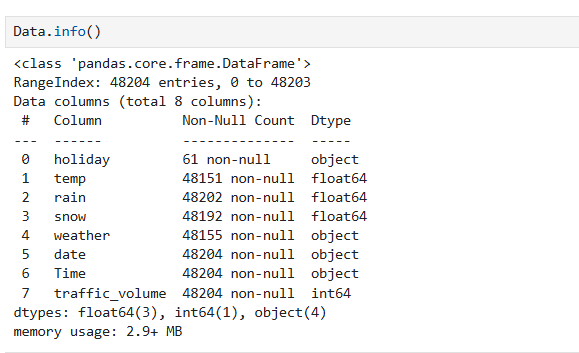
****

**3. Statistical Summary of the Dataset**

The command Data.describe().T provides a summary of the numerical columns in the dataset, giving insights into their distribution.

****

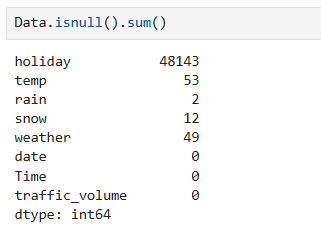
**4. Checking Dataset Information**

The command Data.info() provides an overview of the dataset, including the number of entries, column names, data types, and missing values.****

**5. Checking Missing Values**

The command Data.isnull().sum() counts the number of missing (null) values in each column of the dataset.

* holiday: 48,143 missing values → Most values are missing, meaning this column may not be useful.
* temp: 53 missing values → Some temperature records are missing.
* rain: 2 missing values → Very few missing values, can be handled easily.
* snow: 12 missing values → A small number of missing values.
* weather: 49 missing values → Some weather conditions are missing.

****

**6. Handling Missing Values in the Dataset**

This code is used to handle missing values in specific columns of the dataset.

**1. Filling Missing Values in Numerical Columns**

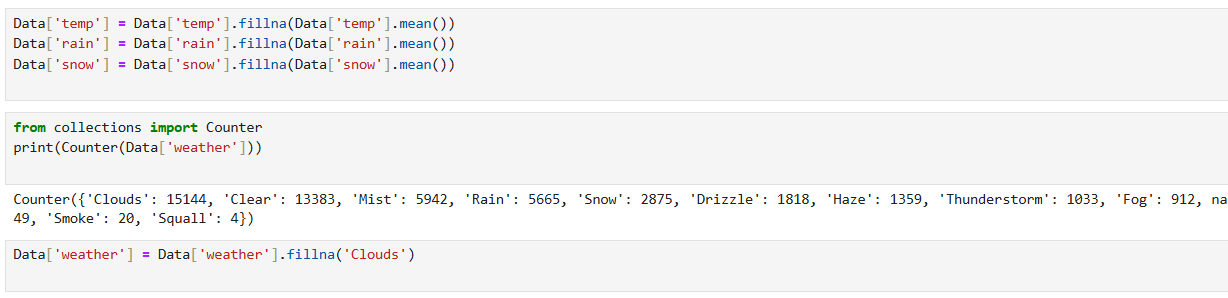
* The missing values in the temp, rain, and snow columns are replaced with their respective mean values.
* This is a common approach for handling missing numerical data, as it helps maintain the overall distribution of values.

**2. Counting Unique Values in the 'weather' Column**

* The Counter function from the collections module is used to count occurrences of each unique value in the weather column.
* The output shows different weather conditions and their frequencies, including some missing (nan) values.

**3. Filling Missing Values in the 'weather' Column**

* The missing values in the weather column are replaced with "Clouds".
* This decision is based on the fact that "Clouds" is the most frequently occurring value in the column (as seen in the previous step).
* Using the mode (most common value) for categorical data is a standard technique to handle missing values.

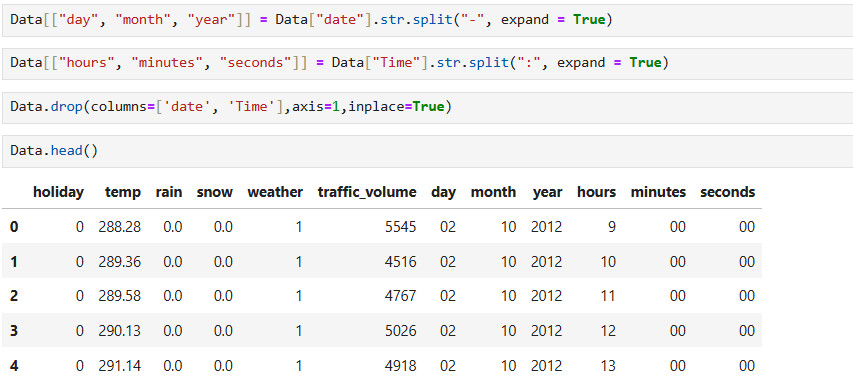
****

**7. Handling Missing Values in the 'holiday' Column**

1. **Filling Missing Values:**
   * The missing values in the 'holiday' column were replaced with 'Not Holiday', assuming that if a holiday was not recorded, it was a regular day.
2. **Converting 'holiday' to Binary Format:**
   * The column was transformed into a binary format:
     + 0 → Represents a regular (non-holiday) day.
     + 1 → Represents a holiday.
3. **Ensuring Numeric Data Type:**
   * The 'holiday' column was converted into a numeric format to facilitate further analysis.
4. **Checking for Missing Values:**
   * After processing, the dataset was checked for missing values, confirming that all columns now have zero missing values.

****

1. **Splitting the 'date' Column into Day, Month, and Year:**
   * The 'date' column, which stores dates in the format DD-MM-YYYY, is split into three separate columns: 'day', 'month', and 'year' using the str.split("-") function.
   * expand=True ensures that the split results are stored in new separate columns.
2. **Splitting the 'Time' Column into Hours, Minutes, and Seconds:**
   * The 'Time' column, formatted as HH:MM:SS, is split into three separate columns: 'hours', 'minutes', and 'seconds' using str.split(":") with expand=True.
3. **Dropping the Original 'date' and 'Time' Columns:**
   * Since the 'date' and 'Time' columns have now been split into their respective components, they are dropped from the dataset using drop(columns=['date', 'Time'], axis=1, inplace=True) to remove unnecessary redundancy.
4. **Displaying the First Few Rows (Data.head()):**
   * The transformed dataset is displayed with separate numerical columns for day, month, year, hours, minutes, and seconds instead of the original 'date' and 'Time' columns.
   * This makes it easier to analyze time-based trends and perform numerical operations.

****

**8. Converting Object Data Types to Integers**

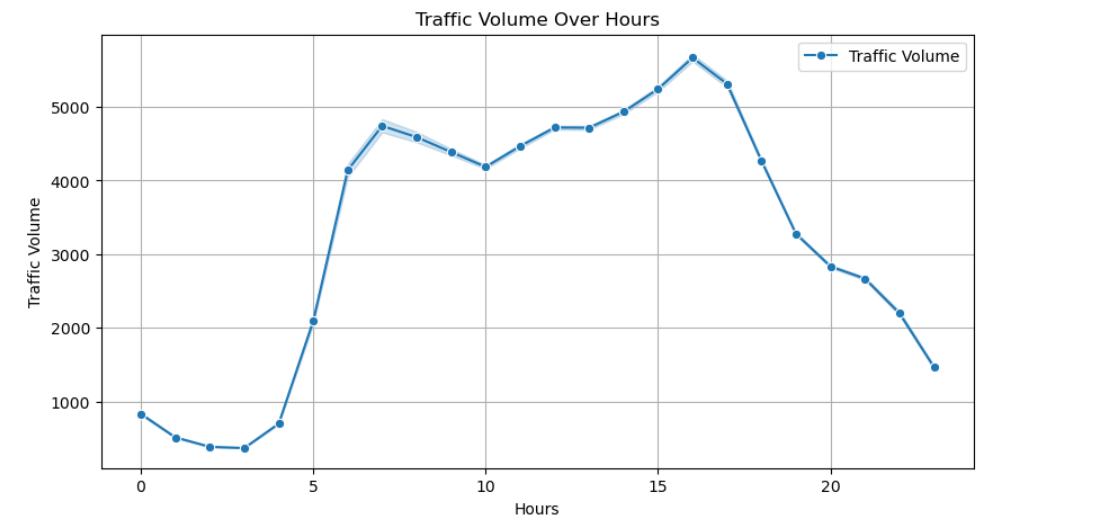
1. **Initial Data Types Check:**
   * The dataset contains various data types, including integer (int64), float (float64), and object.
   * Columns like 'day', 'month', 'year', 'hours', 'minutes', and 'seconds' are stored as object (string) types, which need to be converted to integers.
2. **Conversion of Object Columns to Integers:**
   * The selected columns ('day', 'month', 'year', 'hours', 'minutes', and 'seconds') were converted from object to integer (int32) format.
   * This conversion ensures that the data can be used for numerical analysis and calculations efficiently.
3. **Verifying Data Types After Conversion:**
   * The updated data types were checked, confirming that all selected columns were successfully converted to int32.
   * This step helps optimize memory usage and improves computational efficiency when performing further operations.

****

**EXPLORATORY DATA ANALYSIS**

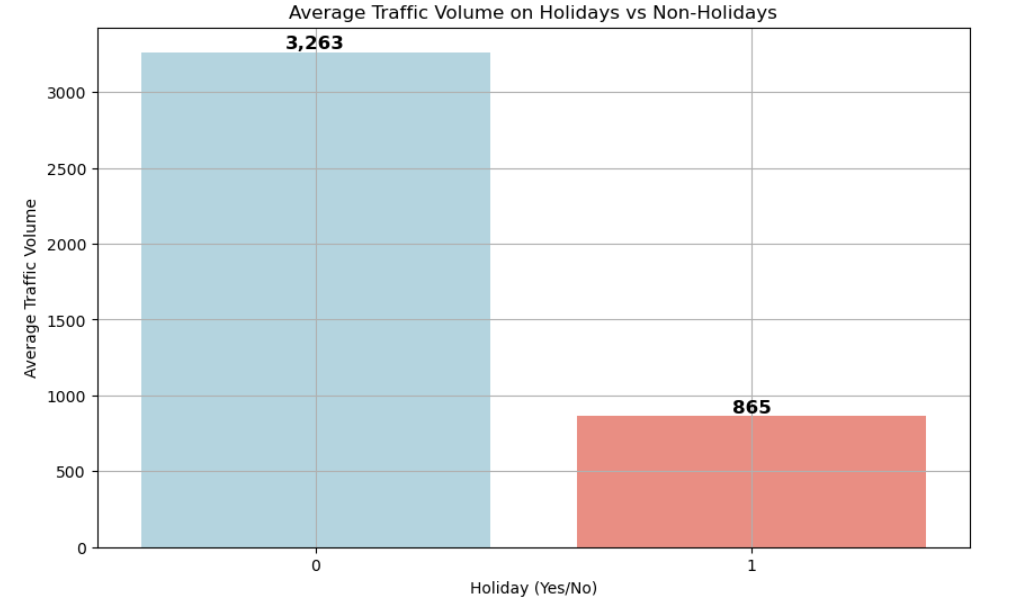
**1. Traffic Volume Over Hours**

1. **Peak Traffic Hours:**
   * The chart highlights two major peaks in traffic volume: one in the morning (6-9 AM) and another in the evening (4-6 PM).
   * These peaks suggest high commuter activity during rush hours.
2. **Off-Peak Hours:**
   * Late Night to Early Morning (12-5 AM): Traffic volume is at its lowest, indicating minimal road activity.
   * Midday (10 AM - 3 PM): Traffic is moderate, but not as high as peak hours.
3. **Traffic Patterns:**
   * The traffic volume follows a bimodal distribution, with morning and evening peaks, likely due to work and school commutes.
   * After 6 PM, traffic volume steadily declines as nighttime approaches.



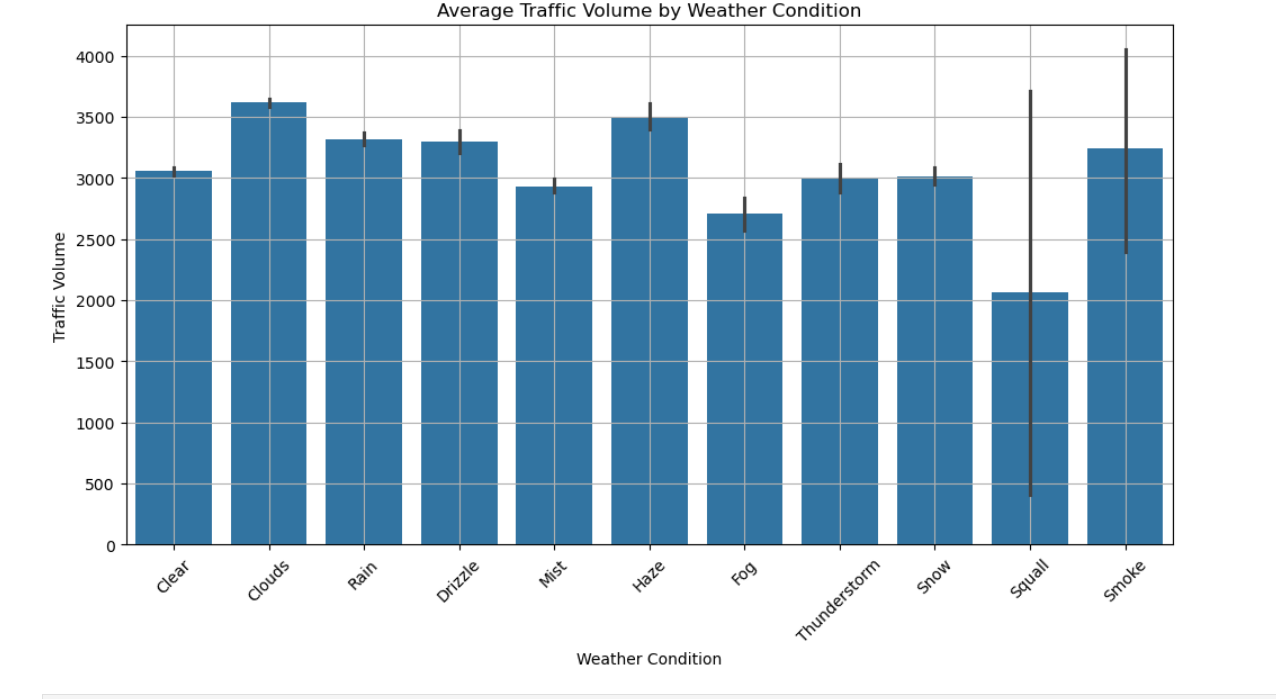
**2. Average Traffic Volume on Holidays vs Non-Holidays**

1. **Traffic Disparity:**
   * The chart shows a significant difference in traffic volume between holidays and non-holidays.
   * On non-holidays (0), the average traffic volume is 3,263, which is much higher than on holidays.
2. **Traffic on Holidays:**
   * On holidays (1), the average traffic volume drops significantly to 865.
   * This suggests that fewer people are commuting on holidays, likely due to reduced work and school-related travel.
3. **Key Insight:**
   * The data indicates that workdays contribute to the majority of traffic congestion, while holidays see a substantial decline in road usage.
   * This trend is expected, as fewer people travel for work or business on holidays, leading to lighter traffic.



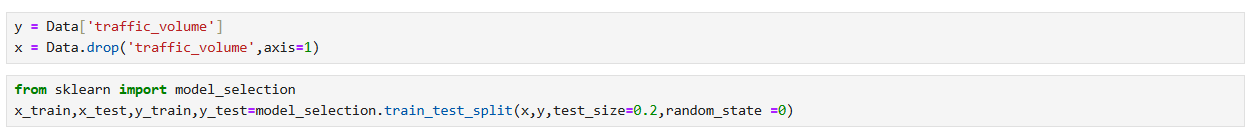
**3. Analysis of Average Traffic Volume by Weather Condition**

* Overall Trend:  
  The traffic volume varies across different weather conditions, with most conditions having an average traffic volume between 3000-3500 vehicles.
* Weather Conditions and Their Impact:
  + High Traffic Volume:
    - Clouds and Haze have the highest traffic volume, averaging around 3500-3600 vehicles, indicating minimal impact on travel.
  + Moderate Traffic Volume:
    - Rain, Drizzle, Mist, Thunderstorm, and Snow have a traffic volume in the range of 2900-3300 vehicles, suggesting a slight reduction in movement.
  + Low Traffic Volume:
    - Fog and Squall significantly reduce traffic volume, with Squall dropping below 2200 vehicles, likely due to hazardous road conditions and visibility issues.



**Model Development**

1. **Separating the Target Variable ('traffic\_volume') from Features:**
   * The 'traffic\_volume' column is selected as the target variable (y), which means the model will predict this value.
   * The rest of the dataset, excluding 'traffic\_volume', is assigned to X, making it the feature set used for prediction.
2. **Splitting the Data into Training and Testing Sets:**
   * The dataset is divided into two parts:
     + 80% for training (used to train the machine learning model).
     + 20% for testing (used to evaluate how well the model generalizes to new data).
   * The random\_state=0 ensures that the data split remains consistent across different runs, making results reproducible.

****

1. **Importing Required Libraries:**
   * linear\_model (for Linear Regression).
   * ensemble (for Random Forest Regressor, an ensemble learning method).
   * svm (for Support Vector Regressor).
   * xgboost (for XGBoost Regressor, a gradient boosting algorithm).
2. **Initializing Regression Models:**

* lin\_reg = linear\_model.LinearRegression() → Linear Regression
  + A simple regression model that assumes a linear relationship between independent and dependent variables.
* Rand = ensemble.RandomForestRegressor() → Random Forest Regressor
  + An ensemble learning method that builds multiple decision trees and merges their predictions for better accuracy.
* svr = svm.SVR() → Support Vector Regressor (SVR)
  + A regression model that uses Support Vector Machines (SVMs) to find a function that best fits the data while minimizing error.
* XGB = xgboost.XGBRegressor() → XGBoost Regressor
  + An optimized gradient boosting algorithm that is widely used for high-performance predictive modeling.

****

1. **Training the Linear Regression Model:**

* The Linear Regression model is trained using the provided dataset.
* It assumes a straight-line relationship between independent and dependent variables.
* The model tries to minimize the difference between actual and predicted values by finding the best-fit line.

1. **Training the Random Forest Regressor:**

* The Random Forest Regressor is trained using multiple decision trees.
* It is an ensemble learning technique that reduces overfitting by averaging predictions from different trees.
* This model is robust and performs well on complex datasets with non-linear relationships.

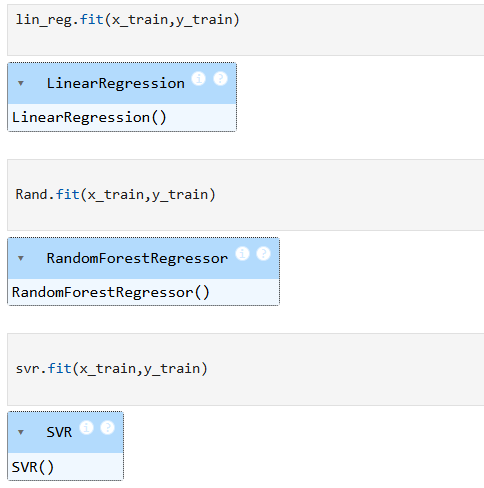
1. **Training the Support Vector Regressor (SVR):**

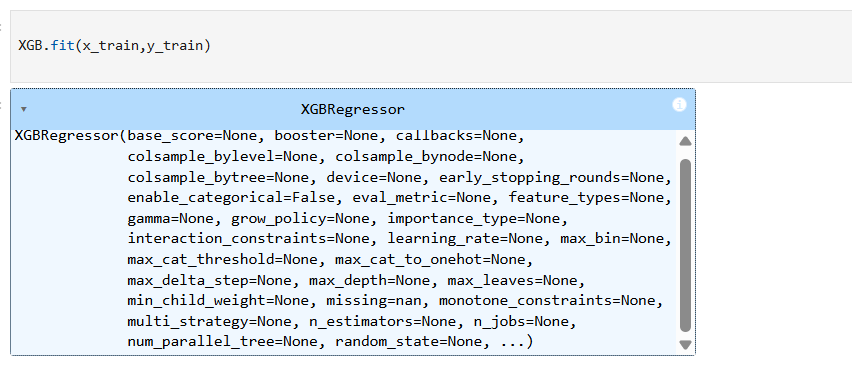
* The Support Vector Regressor (SVR) is trained to predict values while maintaining a margin of error.
* It works well for datasets where relationships between variables are not strictly linear.
* SVR is useful for handling outliers and small datasets by optimizing a balance between error and complexity.

Explanation of the Code in the Image

1. **Training the XGBoost Regressor:**

* The XGBRegressor model is trained using the provided dataset.
* XGBoost (Extreme Gradient Boosting) is an optimized gradient boosting library designed for high performance and efficiency.
* It improves accuracy by iteratively training on errors made by previous models (boosting technique).

****

****

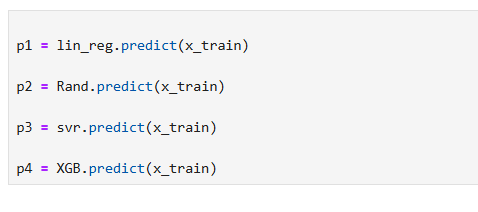
**Evaluation**

This code snippet is performing predictions using four different machine learning models that were previously trained.

Explanation:

Each line of code is using the .predict() method to generate predictions based on the training data (x\_train):

1. p1 = lin\_reg.predict(x\_train)
   * This applies the trained Linear Regression model (lin\_reg) to predict values for x\_train.
2. p2 = Rand.predict(x\_train)
   * This uses the trained Random Forest Regressor (Rand) to predict values for x\_train.
3. p3 = svr.predict(x\_train)
   * This applies the trained Support Vector Regressor (SVR) (svr) to predict values for x\_train.
4. p4 = XGB.predict(x\_train)
   * This uses the trained XGBoost Regressor (XGB) to generate predictions for x\_train.

****

**Model Evaluation Using R² Score**

This Python script evaluates four different regression models (Linear Regression, Random Forest, Support Vector Regressor (SVR), and XGBoost) by calculating their R² scores and determining the best-performing model.

**1. What is R² Score?**

The R² score (coefficient of determination) measures how well a model explains the variance in the target variable (y\_train).

* R² = 1 → Perfect fit
* R² = 0 → Model performs like a simple mean predictor
* R² < 0 → Model performs worse than predicting the mean

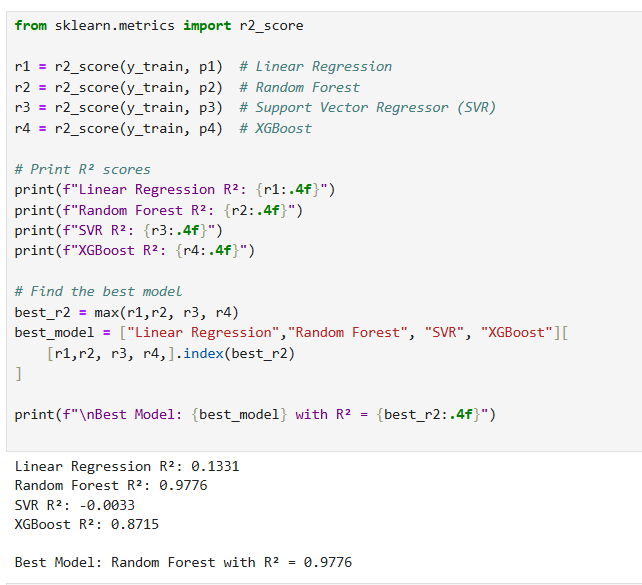
**2️. How the Code Works?**

1. Computes R² scores for each model using the r2\_score() function.
2. Prints the R² scores to compare model performance.
3. Finds the best model by selecting the highest R² score.
4. Displays the best model and its R² value.

**3️. R² Scores from the Output**

|  |  |
| --- | --- |
| **Model** | **R² Score** |
| Linear Regression | 0.1331 |
| Random Forest | 0.9776 |
| SVR | -0.0033 |
| XGBoost | 0.8715 |

* Random Forest performed the best with an R² score of 0.9776.
* SVR performed the worst (R² = -0.0033), meaning it failed to fit the data.

****

**Saving and Deploying a Machine Learning Model Using Flask**

**1️. Saving the Model Using Pickle**

After training a machine learning model, we need to save it so that we can use it later without retraining. This is done using the pickle module in Python.

Explanation of the Image:

* The image shows a Python script that saves a trained model (Rand) using pickle.
* pickle.dump(Rand, open("model.pkl", 'wb')):
  + Serializes and saves the model as model.pkl.
  + 'wb' mode means "write binary," ensuring the model is stored efficiently.

****

Why Save the Model?

* Saves time by avoiding retraining.
* Allows easy deployment in applications.

**2️. Deploying the Model Using Flask**

app.py (Flask Backend)

The app.py file creates a Flask web application that loads the saved model (model.pkl) and makes predictions.

Steps in app.py:

1. Load the saved model using pickle.load().
2. Create a Flask web app and define routes.
3. Accept user input from a web form.
4. Use the model to predict based on input data.
5. Display the prediction on the webpage.

index.html (Webpage UI)

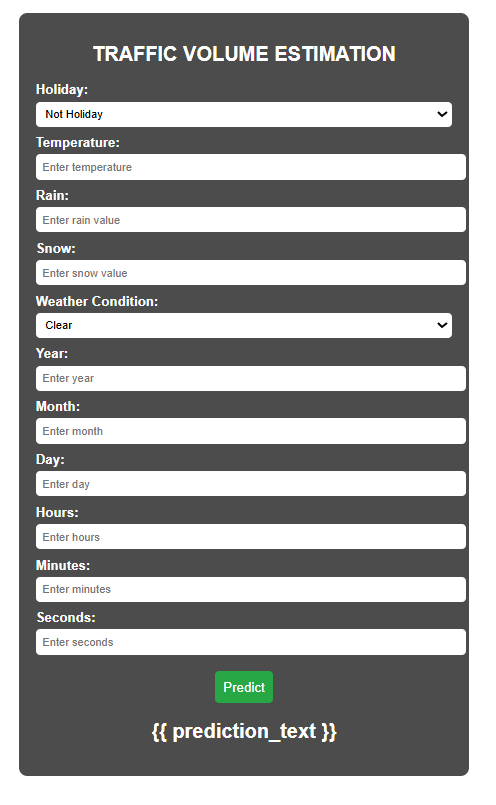
A simple HTML webpage allows users to enter input values and get predictions.

Webpage Features:

* A form where users enter values.
* A "Predict" button to submit input.

**3️. Workflow Summary**

1. Train & Save Model → Use pickle to save (model.pkl).
2. Create Flask App → Load the model and create a web API.
3. Design Web Page → Collect user input and display predictions.
4. Run Flask App → Start the server and access via a browser.



**CONCLUSION**

This project successfully demonstrated the complete workflow of developing and deploying a machine learning model using Flask, starting from data collection to final deployment.

The process began with **data collection and preprocessing**, ensuring that the dataset was clean and suitable for training. Various **machine learning models** were tested, and their performance was evaluated using appropriate metrics. The best-performing model was then selected and **saved using the pickle module**, allowing it to be reused without retraining.

For deployment, a **Flask web application** was developed, providing a user-friendly interface for making predictions. This integration bridges the gap between machine learning models and real-world applications, making the model accessible to users via a web browser.

Overall, this project highlights the end-to-end process of machine learning model deployment, demonstrating how raw data can be transformed into a fully functional predictive system. Future improvements can include enhancing model performance, integrating additional features, and deploying the application on cloud platforms for wider accessibility.