

Final Project Report

1. Introduction
 - 1.1. Project overviews
 - 1.2. Objectives
2. Project Initialization and Planning Phase
 - 2.1. Define Problem Statement
 - 2.2. Project Proposal (Proposed Solution)
 - 2.3. Initial Project Planning
3. Data Collection and Preprocessing Phase
 - 3.1. Data Collection Plan and Raw Data Sources Identified
 - 3.2. Data Quality Report
 - 3.3. Data Exploration and Preprocessing
4. Model Development Phase
 - 4.1. Feature Selection Report
 - 4.2. Model Selection Report
 - 4.3. Initial Model Training Code, Model Validation and Evaluation Report
5. Model Optimization and Tuning Phase
 - 5.1. Hyperparameter Tuning Documentation
 - 5.2. Performance Metrics Comparison Report
 - 5.3. Final Model Selection Justification
6. Results
 - 6.1. Output Screen shots
7. Advantages & Disadvantages
8. Conclusion
9. Future Scope
10. Appendix
 - 10.1. Source Code
 - 10.2. Git Hub & Project Demo Link

INTRODUCTION

Project overviews :

TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning is an application to develop an advanced traffic management system that utilizes machine learning algorithms for accurate estimation and prediction of traffic volumes. The system aims to enhance traffic management, urban planning, and commuter experiences by providing precise traffic forecasts based on historical data and various influencing factors.

Key Features:

Traffic Volume Prediction: Employ machine learning algorithms to analyze historical traffic data, weather conditions, special events, and other relevant factors to predict future traffic volumes.

Dynamic Traffic Management: Provide real-time traffic estimations to enable transportation authorities to implement adaptive traffic control measures, optimize signal timings, and adjust lane configurations, thereby reducing congestion.

Urban Development Planning: Equip city planners and developers with insights into future traffic patterns, facilitating effective planning of infrastructure projects, road networks, public transit systems, and commercial zones to enhance accessibility and efficiency.

Commuter Guidance and Navigation: Offer accurate traffic volume forecasts to individual commuters and navigation apps, allowing for intelligent route planning, real-time updates, and alternative route suggestions to improve overall travel experiences.

Scenarios of Application:

Public Transit Enhancement: Transit agencies can utilize TrafficTelligence data to optimize bus and train schedules based on expected traffic conditions. By aligning transit services with real-time and predicted traffic flows, agencies can improve on-time performance and enhance the overall commuter experience.

Sustainability Initiatives: Traffic Telligence can support urban sustainability efforts by analyzing traffic patterns to promote alternative modes of transport. By identifying congested areas and peak travel times, cities can implement programs encouraging public transport use, carpooling, or cycling, thereby reducing overall vehicle emissions.

Emergency Response Optimization: TrafficTelligence can assist emergency services by predicting traffic conditions in real-time. During incidents, emergency responders can access live traffic forecasts to identify the quickest routes, ensuring timely responses and potentially saving lives.

Objectives :

The objective of advanced traffic volume prediction using machine learning is to accurately forecast traffic patterns by analyzing historical and real-time data. This enhances traffic management and optimizes resource allocation, reducing congestion and improving road safety. It integrates various data sources for a comprehensive traffic overview, ultimately aiming to improve user experience and minimize environmental impact.

- Facilitate Real-Time Traffic Management:** Provide transportation authorities with real-time traffic estimations to enable dynamic traffic control and reduce congestion.
- Promote Sustainable Transportation Practices:** Provide data that supports initiatives aimed at reducing vehicle emissions and encouraging the use of public transit and alternative modes of travel.
- Enhance Commuter Experience:** Deliver accurate traffic insights to individual commuters and navigation apps, improving route planning and overall travel experiences.
- Accurate Forecasting:** Utilize machine learning algorithms to predict traffic volumes with high accuracy based on historical data and real-time variables (e.g., weather, events, time of day).
- Safety Improvement:** Enhance road safety by identifying potential congestion areas and enabling interventions before issues arise.

Project Initialization and Planning Phase

Define Problem Statement :

Traffic congestion is a significant challenge in urban areas, leading to increased travel times and environmental impact. Traditional traffic management systems often need more real-time adaptability and predictive capabilities. To address this, TrafficTelligence employs machine learning algorithms to analyze historical traffic data and various influencing factors, providing accurate real-time traffic volume estimations. This advanced system aims to reduce congestion through adaptive traffic control, support urban planning with data-driven insights, and enhance commuter navigation experiences. Using libraries like Pandas, NumPy, and Matplotlib, TrafficTelligence enables users to monitor daily congestion patterns, ultimately improving traffic management and overall mobility.



Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	A Traffic Analyst	Solve the negative environmental impact of congestion	The complexity of traffic patterns and external factors are unpredictable	I have a strong foundation in data analysis techniques	Confident, Continuously learning, and adapting to new technologies easily

Project Proposal (Proposed Solution) :

This project proposes developing an advanced traffic volume prediction system using regression algorithms such as Linear Regression, Decision Tree, Random Forest, Support Vector Regression, and XGBoost. We will preprocess a comprehensive data set that includes historical traffic data and influencing factors, then train and evaluate each model to identify the most effective one. The selected model will be serialized in .pkl format and integrated into a user-friendly Flask application, enabling real-time traffic volume predictions.

Project Overview	
Objective	The primary objective is to develop an advanced traffic volume prediction system using machine learning algorithms to optimize urban traffic management.
Scope	The project will focus on developing a traffic volume prediction system using historical data and machine learning algorithms, limited to urban areas with varying traffic conditions. It will include data preprocessing, model training, and deployment of a Flask application , but will not address physical infrastructure changes or broader transportation policy issues.
Problem Statement	
Description	Urban traffic congestion increases travel times and environmental impacts, exacerbated by traditional traffic management systems that lack real-time adaptability. This project aims to leverage machine learning to provide accurate traffic volume predictions, enhancing traffic flow and commuter experiences.
Impact	Solving urban traffic congestion through accurate volume predictions can lead to reduced travel times, lower emissions, and improved overall air quality. Additionally, it enhances commuter experiences and supports more efficient urban planning and resource allocation.
Proposed Solution	
Approach	The approach involves preprocessing historical traffic data and various influencing factors, followed by training and evaluating multiple regression models to identify the most effective one. The selected model will be integrated into a Flask application for real-time predictions.

Key Features	Real-Time Predictions: Delivers accurate traffic volume estimates using advanced machine learning algorithms through a user-friendly Flask application. Data Insights: Leveraging historical data to enhance adaptive traffic control and urban planning.
--------------	--

Initial Project Planning :

Product Backlog, Sprint Schedule, and Estimation :

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Data Collection	SL-1	Understanding, Collecting and Downloading the data	Low	Sri Rama Chandu	23/09/2024	25/09/2024
Sprint-1	Data preprocessing	SL-2	Importing necessary libraries and Analysing the data	High	Yeswant Sarma	25/09/2024	27/09/2024
Sprint-1	Data Preprocessing	SL-3	Visualizing and Feature Scaling	High	Tanveer	28/09/2024	30/09/2024
Sprint-3	Report	SL-4	Report	Medium	Sri Rama Chandu	09/10/2024	09/10/2024
Sprint-2	Model Building	SL-5	Training and Evaluating the model	High	Sai Sathwika	1/10/2024	5/10/2024
Sprint-2	Application Building	SL-6	Building Flask application and Deployment	Medium	Sai Sathwika	6/10/2024	8/10/2024
Sprint-3	Data Collection and Processing Report	SL-7	Report	Medium	Yeswan Sarma	10/10/2024	12/10/2024
Sprint-3	Model Development Phase Report	SL-8	Report	Medium	Tanveer	12/10/2024	13/10/2024
Sprint-3	Model Optimization and Tuning Phase Report	SL-9	Report	Medium	Sai Sathwika	13/10/2024	14/10/2024

Data Collection and Preprocessing Phase

Data Collection Plan and Raw Data Sources Identified :

The plan identifies key data types, including historical traffic data, weather conditions, special events. Each data type specifies its purpose, collection method, and frequency.

Section	Description
Project Overview	The TrafficTelligence project aims to develop a machine-learning a system that accurately predicts traffic volumes by analyzing historical data, weather patterns, and events. Its objective is to enhance traffic management and urban planning, ultimately improving commuter experiences and reducing congestion.
Data Collection Plan	Data for the TrafficTelligence project will be collected from the given platform which is provided in the guided workspace.
Raw Data Sources Identified	There is no raw data set was chosen during the project.

Data Quality Report :

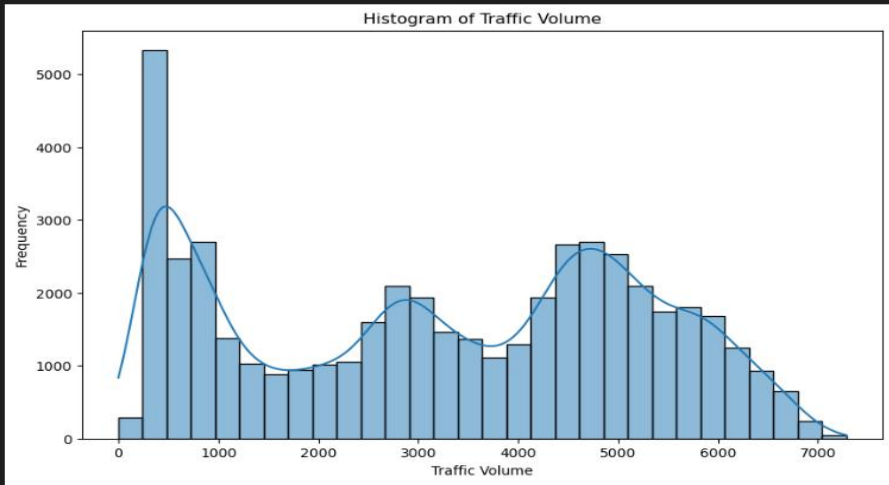
The data set for "TrafficTelligence—Advanced Traffic Volume Estimation with Machine Learning" is sourced from SmartInternz. It summarizes data quality issues related to holidays, weather, and time stamps in the data set, detailing severity levels and resolution plans. It serves as a systematic guide for identifying and rectifying discrepancies to ensure data reliability for traffic volume prediction

Data Source	Data Quality Issue	Severity	Resolution Plan
SmartInternz Data set	Missing values in the “Weather”, Temperature, Rain, Snow, Clouds, columns.	Moderate	Use mean/median imputation
SmartInternz Data set	Categorical data in the data set	Moderate	Encoding has to be done in

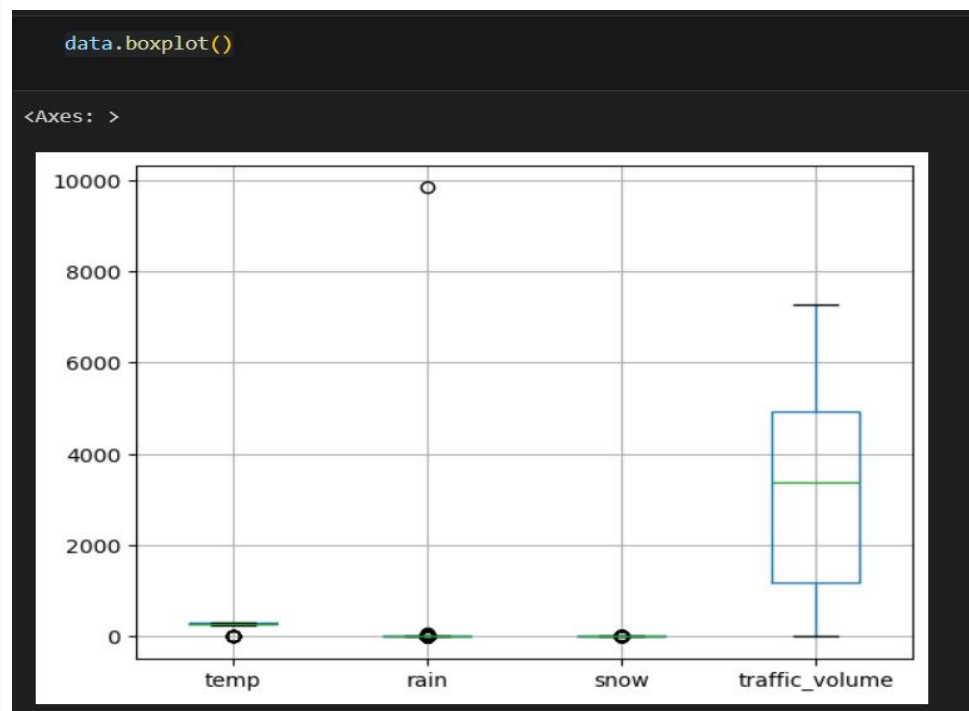
the data.

Data Exploration and Preprocessing :

Data set variables will undergo statistical analysis to uncover patterns and identify outliers, utilizing Python for preprocessing tasks such as normalization and feature engineering. Data cleaning will address missing values and outliers, ensuring high-quality data for subsequent analysis and modeling, ultimately establishing a solid foundation for insights and accurate predictions.

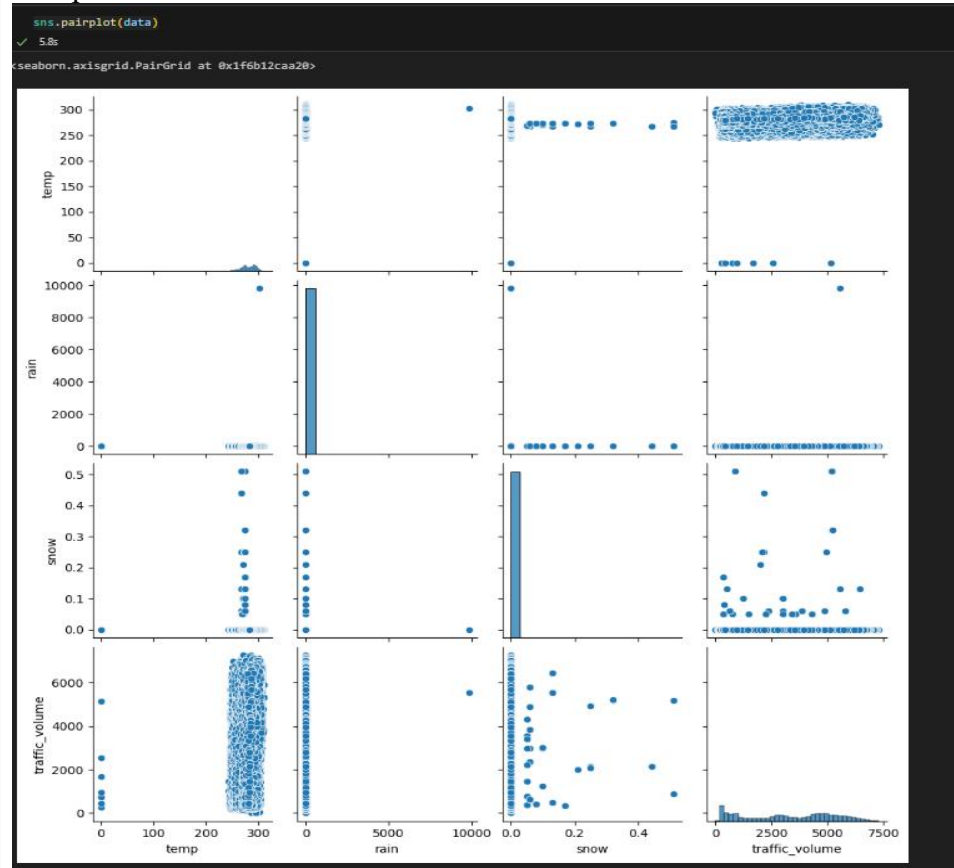
Section	Description																																													
Data Overview	<div><pre>> data.describe()</pre><pre>[8]</pre><pre>...</pre><table><tr><th></th><th>temp</th><th>rain</th><th>snow</th><th>traffic_volume</th></tr><tr><td>count</td><td>48151.000000</td><td>48202.000000</td><td>48192.000000</td><td>48204.000000</td></tr><tr><td>mean</td><td>281.205351</td><td>0.334278</td><td>0.000222</td><td>3259.818355</td></tr><tr><td>std</td><td>13.343675</td><td>44.790062</td><td>0.008169</td><td>1986.860670</td></tr><tr><td>min</td><td>0.000000</td><td>0.000000</td><td>0.000000</td><td>0.000000</td></tr><tr><td>25%</td><td>272.160000</td><td>0.000000</td><td>0.000000</td><td>1193.000000</td></tr><tr><td>50%</td><td>282.460000</td><td>0.000000</td><td>0.000000</td><td>3380.000000</td></tr><tr><td>75%</td><td>291.810000</td><td>0.000000</td><td>0.000000</td><td>4933.000000</td></tr><tr><td>max</td><td>310.070000</td><td>9831.300000</td><td>0.510000</td><td>7280.000000</td></tr></table></div>		temp	rain	snow	traffic_volume	count	48151.000000	48202.000000	48192.000000	48204.000000	mean	281.205351	0.334278	0.000222	3259.818355	std	13.343675	44.790062	0.008169	1986.860670	min	0.000000	0.000000	0.000000	0.000000	25%	272.160000	0.000000	0.000000	1193.000000	50%	282.460000	0.000000	0.000000	3380.000000	75%	291.810000	0.000000	0.000000	4933.000000	max	310.070000	9831.300000	0.510000	7280.000000
	temp	rain	snow	traffic_volume																																										
count	48151.000000	48202.000000	48192.000000	48204.000000																																										
mean	281.205351	0.334278	0.000222	3259.818355																																										
std	13.343675	44.790062	0.008169	1986.860670																																										
min	0.000000	0.000000	0.000000	0.000000																																										
25%	272.160000	0.000000	0.000000	1193.000000																																										
50%	282.460000	0.000000	0.000000	3380.000000																																										
75%	291.810000	0.000000	0.000000	4933.000000																																										
max	310.070000	9831.300000	0.510000	7280.000000																																										
Univariate Analysis	<div><pre>plt.figure(figsize=(10, 6)) sns.histplot(data['traffic_volume'], bins=30, kde=True) plt.title('Histogram of Traffic Volume') plt.xlabel('Traffic Volume') plt.ylabel('Frequency') plt.show()</pre><p>✓ 0.3s</p></div>																																													

Bivariate Analysis



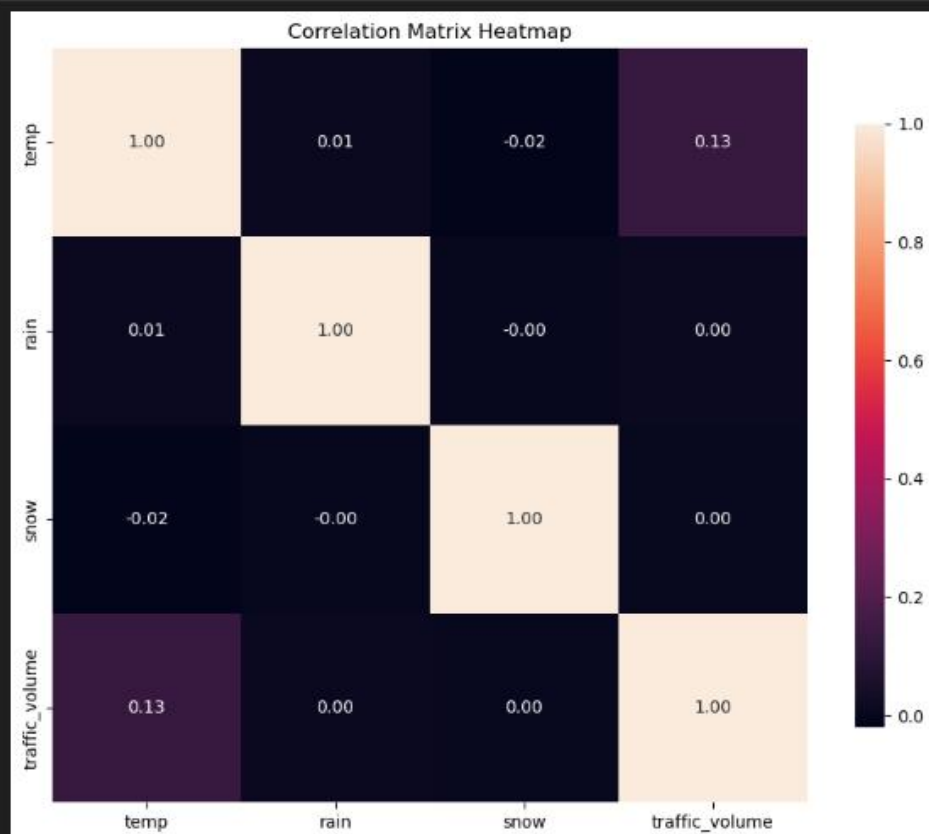
Multivariate Analysis

Pair plot :



Heat Map:


```
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", square=True, cbar_kws={"shrink": .8})
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Outliers and
Anomalies

There are no outliers or anomalies in the dataset.

Data Preprocessing Code Screen shots

Loading Data

```
data = pd.read_csv('traffic_volume.csv')
✓ 0.0s
```

data

```
✓ 0.0s
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	0	288.28	0.0	0.0	1.0	02-10-2012	09:00:00	5545
1	0	289.36	0.0	0.0	1.0	02-10-2012	10:00:00	4516
2	0	289.58	0.0	0.0	1.0	02-10-2012	11:00:00	4767
3	0	290.13	0.0	0.0	1.0	02-10-2012	12:00:00	5026
4	0	291.14	0.0	0.0	1.0	02-10-2012	13:00:00	4918
...
48199	0	283.45	0.0	0.0	1.0	30-09-2018	19:00:00	3543
48200	0	282.76	0.0	0.0	1.0	30-09-2018	20:00:00	2781
48201	0	282.73	0.0	0.0	Thunderstorm	30-09-2018	21:00:00	2159
48202	0	282.09	0.0	0.0	1.0	30-09-2018	22:00:00	1450
48203	0	282.12	0.0	0.0	1.0	30-09-2018	23:00:00	954

48204 rows x 8 columns

Handling Missing Data

```
data['temp'] = data['temp'].fillna(data['temp'].median())
data['rain'] = data['rain'].fillna(data['rain'].median())
data['snow'] = data['snow'].fillna(data['snow'].median())

✓ 0.0s

print(data['weather'].value_counts())

✓ 0.0s

weather
Clouds      15144
Clear       13383
Rain       15842
Rain       5665
Snow       3879
Drizzle     1818
Haze       1059
Thunderstorm 1013
Fog         912
Smoke       70
Squall       4
Name: count, dtype: int64

data['weather'].fillna('Clouds', inplace=True)

✓ 0.0s

C:\Users\saikat\AppData\Local\Temp\ipykernel_15312\1731951281.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the original object.

data['weather'].fillna('Clouds', inplace=True)

# prompt: print NaN in holiday column to 0

data['holiday'].fillna(0, inplace=True)

✓ 0.0s

C:\Users\saikat\AppData\Local\Temp\ipykernel_15312\1731951281.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or 'df[col] = df[col].method(value)' instead, to perform the operation inplace on the original object.

data['holiday'].fillna(0, inplace=True)
```

Data Transformation

```
data[['day', 'month', 'year']] = data['date'].str.split("-", expand=True)
data[['hours', 'minutes', 'seconds']] = data['Time'].str.split(":", expand=True)
data.drop(columns=['date', 'Time'], axis=1, inplace=True)

print(data.head())
```

	holiday	temp	rain	snow	weather	traffic_volume	day	month	year	hours	\
0	0	288.28	0.0	0.0	1.0	5545	02	10	2012	09	
1	0	289.36	0.0	0.0	1.0	4516	02	10	2012	10	
2	0	289.58	0.0	0.0	1.0	4767	02	10	2012	11	
3	0	290.13	0.0	0.0	1.0	5026	02	10	2012	12	
4	0	291.14	0.0	0.0	1.0	4918	02	10	2012	13	

	minutes	seconds
0	00	00
1	00	00
2	00	00
3	00	00
4	00	00

Feature Engineering

```
le = LabelEncoder()

✓ 0.0s

le.fit(data['weather'])

✓ 0.0s

LabelEncoder 1 ?
LabelEncoder ()

#splitting into independant and dependant variables
y=data['traffic_volume']
x=data.drop(columns=['traffic_volume'], axis=1)
x['holiday'] = le.fit_transform(x['holiday'].astype(str))
x['weather'] = le.fit_transform(x['weather'].astype(str))
```

Save Processed Data

```
data.head()

✓ 0.0s
```

	holiday	temp	rain	snow	weather	traffic_volume	day	month	year	hours	minutes	seconds
0	0	288.28	0.0	0.0	1.0	5545	02	10	2012	09	00	00
1	0	289.36	0.0	0.0	1.0	4516	02	10	2012	10	00	00
2	0	289.58	0.0	0.0	1.0	4767	02	10	2012	11	00	00
3	0	290.13	0.0	0.0	1.0	5026	02	10	2012	12	00	00
4	0	291.14	0.0	0.0	1.0	4918	02	10	2012	13	00	00

Model Development Phase

Feature Selection Report :

The Feature Selection Report details the rationale for selecting specific features (e.g., temperature, time of day, weather conditions) for the traffic volume prediction model. It evaluates the relevance, importance, and impact of these features on predictive accuracy, ensuring the inclusion of key factors that influence the model's ability to accurately estimate traffic flow and identify congestion patterns.

Feature	Description	Selected (Yes/No)	Reasoning
holiday	Tells if that particular day is a holiday or not	Yes	Influences traffic volume predictions
temp	It says about the temperature on that day	Yes	Temperature impacts the traffic volume.
rain	It says whether it is raining or not	Yes	Rain impacts the traffic.
snow	It says whether it is snowing or not	Yes	Snow blocks the road so it impacts the traffic congestion.
weather	Tells about the weather on that particular day	Yes	Different weather conditions lead to different effects on traffic patterns, commuter behavior, and travel times.
date	It gives the date of that particular day	Yes	It influences commuting patterns, with variations on weekdays, holidays, and seasons affecting traffic volume.

time	The time at which traffic volume is	Yes	By incorporating time features, we can identify peak hours, optimize traffic management strategies, and improve predictions for traffic volume throughout the day.
Traffic volume	It gives data on traffic volume based on all features	Yes	Analyzing traffic volume helps identify congestion patterns, optimize traffic flow, and inform infrastructure planning, enabling better traffic management and improved commuter experiences.

Model Selection Report :

This report identifies the best predictive model for a specific problem by evaluating various candidates based on performance metrics. It outlines the data set, candidate models, and selection criteria, ultimately recommending the most suitable model. This process ensures transparency and informs decision-making for stakeholders

Model	Description	Hyperparameters	Performance Metric (e.g., Accuracy, F1 Score)
Linear Regression	Linear Regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.	-	14%
Decision Tree	A Decision Tree Regressor predicts continuous outcomes by splitting data into subsets based on feature values, resulting in a tree-like structure. Suitable for initial insights into traffic patterns.	-	78%
Random Forest Regressor	Random Forest Regressor is an ensemble method that combines multiple decision trees to enhance prediction accuracy and reduce overfitting. Predicts traffic flow and congestion by analyzing diverse data sources, improving traffic management and urban mobility, and provides feature importance for	-	84%

	traffic prediction		
Support vector Regression	Support Vector Regression (SVR) in traffic intelligence can handle non-linear relationships and robustness to outliers makes it effective for modeling complex traffic patterns. This enhances traffic management and helps improve urban mobility strategies.	-	59%
Gradient Boosting	Gradient boosting with trees optimizes predictive performance handles complex relationships, and is suitable for accurate traffic predictions.	-	86%

Initial Model Training Code, Model Validation and Evaluation Report :

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include accuracy, and confusion matrices for multiple models like Random Forest Regression, Support Vector Regression, XGBoost, Linear Regression, etc.,, presented through respective screen shots.

Initial Model Training Code:

```

from sklearn import linear_model
from sklearn import tree
from sklearn import ensemble
from sklearn import svm
from sklearn import metrics
import pickle
import xgboost
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

```

✓ 4.2s

```
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2, random_state=42)
```

✓ 0.0s

```
lin_reg = linear_model.LinearRegression()  
Dtree = tree.DecisionTreeRegressor()  
Rand = ensemble.RandomForestRegressor()  
svr = svm.SVR()  
XGB = xgboost.XGBRegressor()
```

✓ 0.0s

```
lin_reg.fit(x_train,y_train)  
Dtree.fit(x_train,y_train)  
Rand.fit(x_train,y_train)  
svr.fit(x_train,y_train)  
XGB.fit(x_train,y_train)
```

✓ 1m 40.9s

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,  
             colsample_bylevel=None, colsample_bynode=None,  
             colsample_bytree=None, device=None, early_stopping_rounds=None,  
             enable_categorical=False, eval_metric=None, feature_types=None,  
             gamma=None, grow_policy=None, importance_type=None,  
             interaction_constraints=None, learning_rate=None, max_bin=None,  
             max_cat_threshold=None, max_cat_to_onehot=None,  
             max_delta_step=None, max_depth=None, max_leaves=None,  
             min_child_weight=None, missing=None, monotone_constraints=None,  
             multi_strategy=None, n_estimators=None, n_jobs=None,  
             num_parallel_tree=None, random_state=None, ...)
```

```
p1 = lin_reg.predict(x_train)  
p2 = Dtree.predict(x_train)  
p3 = Rand.predict(x_train)  
p4 = svr.predict(x_train)  
p5 = XGB.predict(x_train)
```

✓ 2m 25.4s

```
from sklearn import metrics  
print(metrics.r2_score(p1,y_train))  
print(metrics.r2_score(p2,y_train))  
print(metrics.r2_score(p3,y_train))  
print(metrics.r2_score(p4,y_train))  
print(metrics.r2_score(p5, y_train))
```

✓ 0.0s

```
-5.472535413028158  
1.0  
0.9754382687609833  
-12.320314642857845  
0.8469192209869841
```

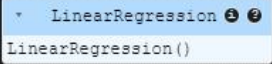
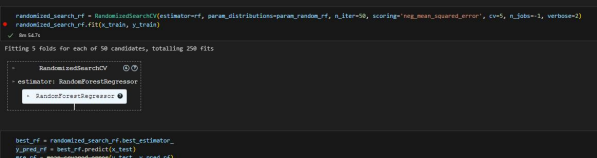
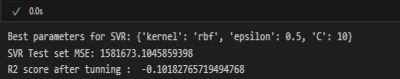

Model Validation and Evaluation Report:

Model	Regression Report	R2_Score
Linear Regression	<pre>p1 = lin_reg.predict(x_test) regression_report(y_test,p1) ✓ 0.0s</pre> <p>Regression Report: Mean Absolute Error: 1636.2917687816025 Mean Squared Error: 3396347.4025021424 Root Mean Squared Error: 1842.9181757479473 R² Score: 0.14092936529824218 Explained Variance Score: 0.1409820946135063</p>	14%
Decision Tree Regressor	<pre>p2 = Dtree.predict(x_test) regression_report(y_test,p2) ✓ 0.0s</pre> <p>Regression Report: Mean Absolute Error: 554.2354527538637 Mean Squared Error: 1099370.4845970336 Root Mean Squared Error: 1048.508695527621 R² Score: 0.7219257078120539 Explained Variance Score: 0.7220542092663976</p>	72%
Random Forest Regressor	<pre>p3 = Rand.predict(x_test) regression_report(y_test,p3) ✓ 0.3s</pre> <p>Regression Report: Mean Absolute Error: 497.7161642983093 Mean Squared Error: 621916.1681831761 Root Mean Squared Error: 788.6166167303198 R² Score: 0.8426927949305771 Explained Variance Score: 0.8428044626755363</p>	84%
SVR	<pre>p4 = svr.predict(x_test) regression_report(y_test,p4) ✓ 33.6s</pre> <p>Regression Report: Mean Absolute Error: 1514.4397776051203 Mean Squared Error: 2997923.7415386634 Root Mean Squared Error: 1731.4513396392817 R² Score: 0.24170647280259672 Explained Variance Score: 0.2435684593747267</p>	24%
XGB	<pre>p5 = XGB.predict(x_test) regression_report(y_test,p5) ✓ 0.0s</pre> <p>Regression Report: Mean Absolute Error: 527.3010232711454 Mean Squared Error: 628595.9247982444 Root Mean Squared Error: 792.8404157194841 R² Score: 0.841003220197812 Explained Variance Score: 0.8410340833901695</p>	84%

Model Optimization and Tuning Phase

Hyperparameter Tuning Documentation :

The Xtream Gradient Boosting (XGBoost) model has been chosen for this project due to its exceptional performance in predictive tasks. Known for its high accuracy and efficiency, XGBoost is particularly adept at handling complex relationships within data. Its built-in mechanisms to reduce overfitting and optimize predictive accuracy make it a suitable choice for our objectives.

Model	Tuned Hyperparameters	Optimal Values
Linear Regression	<pre>linear_reg = LinearRegression() linear_reg.fit(x_train, y_train)</pre>  <pre>y_pred_linear = linear_reg.predict(x_test) mse_linear = mean_squared_error(y_test, y_pred_linear) r2_linear = metrics.r2_score(y_pred_linear, y_test)</pre>	<pre>print("Linear Regression Test set MSE:", mse_linear) print("R2 score after tuning : ", r2_linear)</pre>  <p>Linear Regression Test set MSE: 3396347.4025021424 R2 score after tuning : -5.431530418573976</p>
Decision Tree Regressor	<pre>dt = DecisionTreeRegressor() param_random_dt = { 'max_depth': [10, 20, 30, 40], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4, 8], } random_search_dt = RandomizedSearchCV(estimator=dt, param_distributions=param_random_dt, scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2) random_search_dt.fit(x_train, y_train)</pre>  <pre>test_dt = random_search_dt.best_estimator_ y_pred_dt = test_dt.predict(x_test)</pre>	<pre>mse_dt = mean_squared_error(y_test, y_pred_dt) r2_dt = metrics.r2_score(y_pred_dt, y_test)</pre>  <p>Best parameters for Decision Tree: ('min_samples_split': 2, 'min_samples_leaf': 4, 'max_depth': 10) Decision Tree Test set MSE: 834640.5839893497 R2 score after tuning : 0.7379026129421835</p>
Random Forest Regressor	<pre>rf = RandomForestRegressor() param_random_rf = { 'n_estimators': [100, 200, 300], 'max_depth': [10, 20, 30, 40], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4, 8], } random_search_rf = RandomizedSearchCV(estimator=rf, param_distributions=param_random_rf, scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2) random_search_rf.fit(x_train, y_train)</pre>  <pre>test_rf = random_search_rf.best_estimator_ y_pred_rf = test_rf.predict(x_test)</pre>	<pre>mse_rf = mean_squared_error(y_test, y_pred_rf) r2_rf = metrics.r2_score(y_pred_rf, y_test)</pre>  <p>Best parameters for Random Forest: ('n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': None) Random Forest Test set MSE: 81338.560303944 R2 score after tuning : 0.81087201251262</p>
SVR	<pre>svr = svm.SVR() param_random_svr = { 'C': [0.1, 1, 10], 'epsilon': [0.1, 0.2, 0.5], 'kernel': ['linear', 'rbf'], } random_search_svr = RandomizedSearchCV(estimator=svr, param_distributions=param_random_svr, scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2) random_search_svr.fit(x_train, y_train)</pre>  <pre>test_svr = random_search_svr.best_estimator_ y_pred_svr = test_svr.predict(x_test)</pre>	<pre>mse_svr = mean_squared_error(y_test, y_pred_svr) r2_svr = metrics.r2_score(y_pred_svr, y_test)</pre>  <p>Best parameters for SVR: ('kernel': 'rbf', 'epsilon': 0.5, 'C': 10) SVR Test set MSE: 1501673.1045859398 R2 score after tuning : -0.10182765719494768</p>

XGBoost

```
def fit_xgb_model(X_train, y_train, X_test, y_test):
    """Fit XGBoost model and evaluate performance"""
    # Create XGBoost model
    xgb_model = xgb.XGBRegressor(
        n_estimators=1000,
        max_depth=7,
        learning_rate=0.1,
        subsample=0.8,
        colsample_bytree=0.8,
        seed=42,
        silent=True
    )

    # Fit the model
    xgb_model.fit(X_train, y_train)

    # Predict on test set
    y_pred_xgb = xgb_model.predict(X_test)

    # Calculate metrics
    mse_xgb = metrics.mean_squared_error(y_test, y_pred_xgb)
    r2_xgb = metrics.r2_score(y_test, y_pred_xgb)

    return xgb_model, mse_xgb, r2_xgb
```

```
# Fit the model and evaluate performance
xgb_model, mse_xgb, r2_xgb = fit_xgb_model(X_train, y_train, X_test, y_test)

# Print best parameters for XGBoost
print("Best parameters for XGBoost:", xgb_model.get_params())

# Print XGBoost Test set MSE and R2 score
print("XGBoost Test set MSE: ", mse_xgb)
print("R2 score after tuning: ", r2_xgb)
```

Best parameters for XGBoost: {'n_estimators': 200, 'max_depth': 7, 'learning_rate': 0.2}
XGBoost test set MSE: 534993.4344943126
R2 score after tuning: 0.8495261148959617

Reasons for Selection:

Optimized Predictive Accuracy: Utilizing a gradient boosting framework, XGBoost effectively optimized loss functions, enhancing its predictive capabilities. This allows for fine-tuning to maximize performance, ensuring that TrafficTelligence can adapt to specific traffic forecasting needs and deliver accurate insights for urban planning and traffic management.

High Accuracy: XGBoost consistently demonstrates exceptional performance in various machine learning benchmarks, making it an ideal choice for TrafficTelligence. Its robust nature allows it to deliver precise traffic volume predictions, even in complex urban environments where numerous variables interact.

Complex Relationship Handling: In the context of traffic prediction, XGBoost excels at capturing non-linear relationships and interactions among features, such as time of day, weather conditions, and special events. This capability is crucial for accurately forecasting traffic patterns, where multiple factors influence outcomes

Performance Metrics Comparison Report :

The following metrics were used to evaluate and compare model performance:

R² Score (Coefficient of Determination): The R² score indicates the proportion of variance in the dependent variable that can be explained by the independent variables in the model. For example, an R² score of 0.84 suggests that 84% of the variance in traffic volumes is explained by the model, indicating a strong fit.

Mean Absolute Error (MAE): The average of absolute errors between predicted and actual values. MAE measures the average magnitude of errors in a set of predictions, without considering their direction. It provides an intuitive sense of how much the predictions deviate from actual values in the same units as the target variable. A lower MAE indicates better model performance.

Root Mean Squared Error (RMSE): The square root of the average of squared differences between predicted and actual values. RMSE is the square root of the MSE and provides an error metric in the same units as the target variable. It is useful for understanding the model's prediction error magnitude. A lower RMSE suggests that the model's predictions are closer to the actual values.

Mean Squared Error (MSE): MSE calculates the average of the squares of the errors. the average squared difference between predicted and actual values. It emphasizes larger errors due to squaring, making it sensitive to outliers. A lower MSE signifies improved predictive accuracy.

Explained Variance Score: This metric measures how much of the variability in the target variable is captured by the model. An explained variance score close to 1 indicates that the model accounts for a substantial portion of the variance in the data.

Model	Optimized Metric
Linear Regression	<pre>#linear regression regression_report(y_test,y_pred_linear)</pre> <p>✓ 0.0s</p> <p>Regression Report: Mean Absolute Error: 1636.2917687816025 Mean Squared Error: 3396347.4025021424 Root Mean Squared Error: 1842.9181757479473 R² Score: 0.14092936529824218 Explained Variance Score: 0.1409820946135063</p>
Decision Tree Regressor	<pre>#desicion tree regression_report(y_test,y_pred_dt)</pre> <p>✓ 0.0s</p> <p>Regression Report: Mean Absolute Error: 603.994188573295 Mean Squared Error: 834649.5839893497 Root Mean Squared Error: 913.5915848941198 R² Score: 0.7888840972678335 Explained Variance Score: 0.7889080572908945</p>
Random Forest Regressor	<pre>#random forest regression regression_report(y_test,y_pred_rf)</pre> <p>✓ 0.0s</p> <p>Regression Report: Mean Absolute Error: 494.55865435812325 Mean Squared Error: 611318.5661813644 Root Mean Squared Error: 781.8686374202282 R² Score: 0.8453733477713454 Explained Variance Score: 0.8454733422423539</p>

SVR	<pre>#svr regression_report(y_test,y_pred_svr) ✓ 0.0s Regression Report: Mean Absolute Error: 982.2159131619221 Mean Squared Error: 1581673.1045859398 Root Mean Squared Error: 1257.64585817548 R² Score: 0.5999322928960928 Explained Variance Score: 0.604402149108233</pre>
XGBoost	<pre>#xgb regression_report(y_test,y_pred_xgb) ✓ 0.0s Regression Report: Mean Absolute Error: 474.40938132787414 Mean Squared Error: 534993.0344943126 Root Mean Squared Error: 731.4321803792287 R² Score: 0.8646790945574483 Explained Variance Score: 0.8646957455194769</pre>

Final Model Selection Justification :

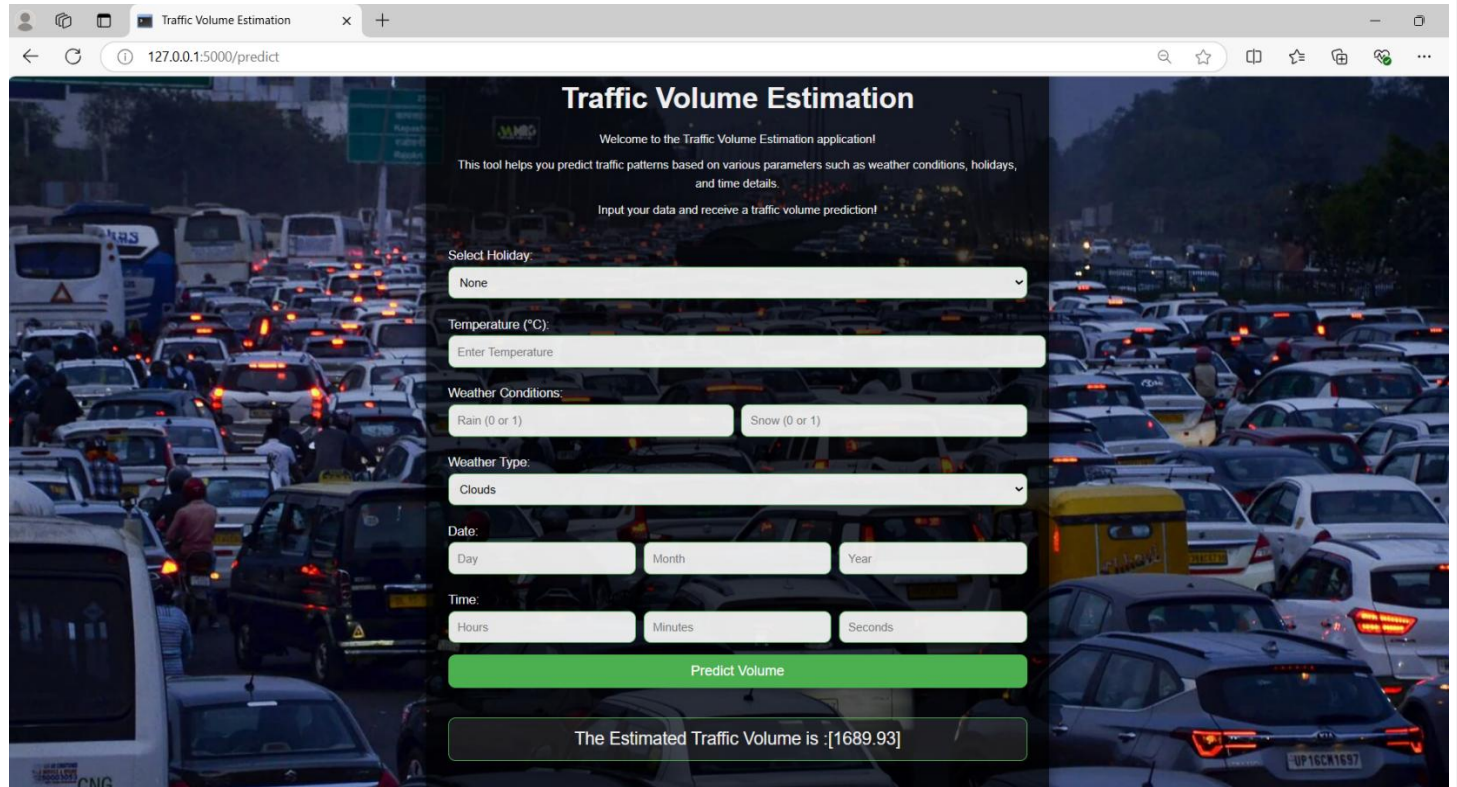
The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Final Model	Reasoning
Extreme Gradient Boosting	XGBoost was chosen as the final optimized model due to its high predictive accuracy and efficiency. Its built-in regularization helped prevent overfitting, while its ability to handle missing values simplified preprocessing. Additionally, XGBoost provides valuable insights into feature importance, enhancing interpretability and model refinement to align with project objectives. Justifying it as a final model.XGBoost is known for its computational efficiency, making it suitable for large data sets typical in traffic analysis.

Results

Output Screenshot :

Initial web-application screen shot:

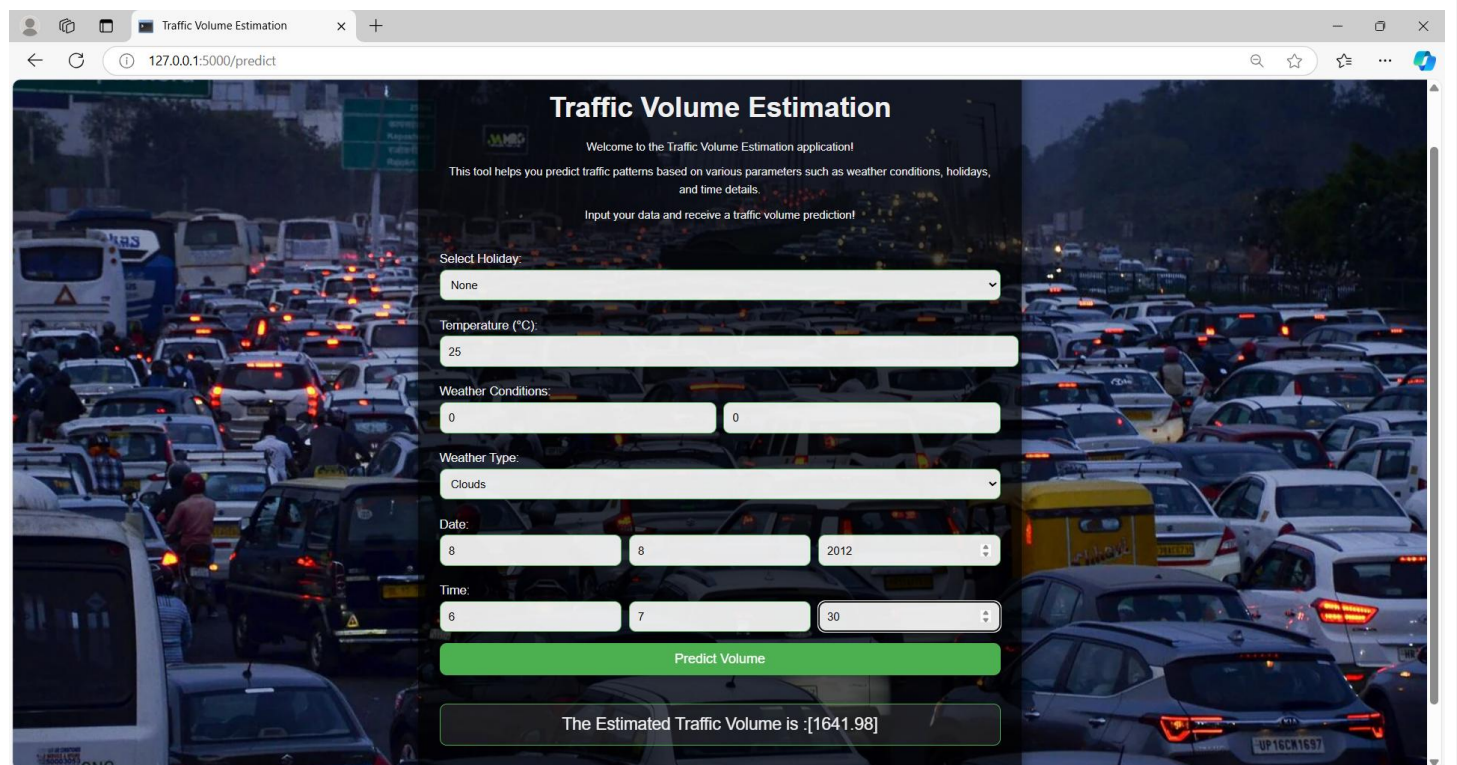


The screenshot shows a web browser window with the title "Traffic Volume Estimation". The URL bar displays "127.0.0.1:5000/predict". The page content includes a header with the title "Traffic Volume Estimation" and a welcome message. Below the header, there is a form with the following fields:

- Select Holiday: A dropdown menu with "None" selected.
- Temperature (°C): A text input field with the placeholder "Enter Temperature".
- Weather Conditions: Two checkboxes, "Rain (0 or 1)" and "Snow (0 or 1)", both of which are unchecked.
- Weather Type: A dropdown menu with "Clouds" selected.
- Date: Three text input fields for "Day", "Month", and "Year".
- Time: Three text input fields for "Hours", "Minutes", and "Seconds".

At the bottom of the form, there is a green button labeled "Predict Volume". Below the button, a message states "The Estimated Traffic Volume is :[1689.93]".

Prediction Screen Shot :



The screenshot shows the same web browser window as the previous one, but with the form fields filled with specific values:

- Select Holiday: A dropdown menu with "None" selected.
- Temperature (°C): A text input field with the value "25".
- Weather Conditions: Two checkboxes, "Rain (0 or 1)" and "Snow (0 or 1)", both of which are unchecked.
- Weather Type: A dropdown menu with "Clouds" selected.
- Date: Three text input fields with the values "8", "8", and "2012".
- Time: Three text input fields with the values "6", "7", and "30".

At the bottom of the form, there is a green button labeled "Predict Volume". Below the button, a message states "The Estimated Traffic Volume is :[1641.98]".

Advantages & Disadvantages

Advantages :

High Prediction Accuracy: Leveraging machine learning algorithms like XGBoost allows for precise traffic volume forecasts, significantly improving planning and management efforts.

Adaptability: The system can adapt to changing traffic patterns, learning from new data and adjusting predictions accordingly. This is crucial for handling dynamic urban environments.

Enhanced Traffic Management: With accurate predictions, transportation authorities can implement better traffic control measures, optimize signal timings, and reduce congestion, leading to improved commuter experiences.

Support for Urban Planning: Traffic predictions aid city planners in designing efficient road networks, public transportation systems, and commercial zones based on expected traffic flows.

Disadvantages:

Data Dependency: The accuracy of predictions relies heavily on the availability and quality of historical and real-time data. Inconsistent or incomplete data can lead to inaccurate forecasts.

Privacy Concerns: The collection and analysis of real-time data may raise privacy issues, particularly when dealing with location-based information from individuals.

Conclusion

The TrafficTelligence project successfully demonstrated the potential of advanced machine learning techniques, specifically using the Xstream Gradient Boosting (XGBoost) model, to predict traffic volumes with high accuracy. The project achieved its objectives of providing actionable insights for traffic management, urban planning, and commuter guidance.

In conclusion, this project effectively utilized the XGBoost algorithm to predict traffic volume, showcasing its ability to handle complex data sets and deliver high accuracy. By integrating critical input features such as weather conditions, time of day, date, and holiday information, the model captured the multifaceted influences on traffic flow.

Furthermore, the insights gained from this project can assist urban planners and transportation agencies in making data-driven decisions to optimize traffic flow, enhance road safety, and improve overall commuter experience. As cities continue to grow and traffic conditions evolve, adopting advanced predictive analytics like XGBoost will be essential in developing smarter, more efficient transportation systems. This project lays the groundwork for future research and applications in traffic volume prediction and management.

Future Scope

The TrafficTelligence project has laid a strong foundation for advancing traffic management and prediction systems. Several avenues for future development and enhancement can be explored to further improve its capabilities and impact :

Enhanced Data Integration: Integrating data from social media, user-generated content, and real-time traffic feeds can provide a richer context for traffic predictions.

Model Improvement and Refinement: Exploring other machine learning algorithms (e.g., deep learning models) or hybrid approaches could improve predictive accuracy and handle more complex patterns.

Scenario Analysis and Simulation: Building capabilities to simulate various scenarios (e.g., road closures, special events) to predict their impact on traffic flow and develop mitigation strategies.

User Personalization and Behavioral Insights: Developing personalized traffic predictions and recommendations based on individual commuter behavior and preferences can enhance user experience. Additionally, analyzing commuter behavior data can provide insights into travel habits and preferences, aiding in the design of more user-centric transportation solutions.

Appendix

Source Code :

Traffic volume.ipynb :

```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn as sk
from sklearn import linear_model
from sklearn import tree
from sklearn import ensemble
from sklearn import svm
from sklearn import metrics
import xgboost
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.model_selection import train_test_split

data = pd.read_csv('traffic volume.csv')

data

print(data.info())

print(data.columns)

data.isnull().sum()

data.shape

data.describe()

data.dtypes

data.info()

data['weather'].value_counts()

data['temp'] = data['temp'].fillna(data['temp'].mean())
data['rain'] = data['rain'].fillna(data['rain'].mean())
data['snow'] = data['snow'].fillna(data['snow'].mean())
data['weather'].fillna('Clouds',inplace=True)

data['weather'] = data['weather'].replace('Clouds', 1.0)
print(data['weather'].value_counts())
data['weather'] = data['weather'].fillna(1)
data['holiday'].fillna(0, inplace=True)
sns.bloxpplot(data=data['temp'])
sns.pairplot(data)
sns.countplot(data['weather'])
sns.boxplot(data['traffic_volume'])
data.boxplot()
plt.figure(figsize=(10, 6))
sns.histplot(data['traffic_volume'],bins=30, kde=True)
```

```

plt.title('Histogram of Traffic Volume')
plt.xlabel('Traffic Volume')
plt.ylabel('Frequency')
plt.show()

numeric_df = data.select_dtypes(include=[np.number])
corr = numeric_df.corr()
corr

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", square=True, cbar_kws={"shrink": .8})
plt.title('Correlation Matrix HeatMap')
plt.show()

data[["day", "month", "year"]] = data["date"].str.split("-", expand=True)
data[["hours", "minutes", "seconds"]] = data["Time"].str.split(":", expand=True)
data.drop(columns=['date', 'Time'], axis=1, inplace=True)

data['weather'] = data['weather'].astype(str)

le = LabelEncoder()
le.fit(data['weather'])

#splitting into independant and dependant variables
y=data['traffic_volume']
x=data.drop(columns=['traffic_volume'],axis=1)
x['holiday'] = le.fit_transform(x['holiday'].astype(str))
x['weather'] = le.fit_transform(x['weather'].astype(str))

feature_names = x.columns
names=x.columns
x=scale(x)
x=pd.DataFrame(x,columns=names)
print(x.head())

from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
from sklearn.preprocessing import StandardScaler

```



```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled = pd.DataFrame(x_scaled, columns=feature_names)

x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2, random_state=42)

lin_reg = linear_model.LinearRegression()
Dtree = tree.DecisionTreeRegressor()
Rand = ensemble.RandomForestRegressor()
svr = svm.SVR()
XGB = xgboost.XGBRegressor()

lin_reg.fit(x_train, y_train)
Dtree.fit(x_train, y_train)
Rand.fit(x_train, y_train)
svr.fit(x_train, y_train)
XGB.fit(x_train, y_train)

p1 = lin_reg.predict(x_train)
p2 = Dtree.predict(x_train)
p3 = Rand.predict(x_train)
p4 = svr.predict(x_train)
p5 = XGB.predict(x_train)

from sklearn import metrics
print(metrics.r2_score(p1, y_train))
print(metrics.r2_score(p2, y_train))
print(metrics.r2_score(p3, y_train))
print(metrics.r2_score(p4, y_train))
print(metrics.r2_score(p5, y_train))

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explained_variance_score
import numpy as np

def regression_report(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
```

```
r2 = r2_score(y_true, y_pred)
explained_variance = explained_variance_score(y_true, y_pred)
print("Regression Report:")
print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R2 Score: {r2}")
print(f"Explained Variance Score: {explained_variance}")

p1 = lin_reg.predict(x_test)
regression_report(y_test,p1)

p2 = Dtree.predict(x_test)
regression_report(y_test,p2)

p3 = Rand.predict(x_test)
regression_report(y_test,p3)

p4 = svr.predict(x_test)
regression_report(y_test,p4)

p5 = XGB.predict(x_test)
regression_report(y_test,p5)

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

rf = RandomForestRegressor()
param_random_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 6],
}
```

```
randomized_search_rf = RandomizedSearchCV(estimator=rf, param_distributions=param_random_rf,
n_iter=50, scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2)

randomized_search_rf.fit(x_train, y_train)

best_rf = randomized_search_rf.best_estimator_
y_pred_rf = best_rf.predict(x_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = metrics.r2_score(y_pred_rf, y_test)

print("Best parameters for Random Forest:", randomized_search_rf.best_params_)
print("Random Forest Test set MSE:", mse_rf)
print("R2 score after tuning : " , r2_rf)

svr = SVR()
param_random_svr = {
    'C': [0.1, 1, 10],
    'epsilon': [0.1, 0.2, 0.5],
    'kernel': ['linear', 'rbf'],
}

random_search_svr = RandomizedSearchCV(estimator=svr,
param_distributions=param_random_svr,scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2)
random_search_svr.fit(x_train, y_train)

best_svr = random_search_svr.best_estimator_
y_pred_svr = best_svr.predict(x_test)
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = metrics.r2_score(y_pred_svr, y_test)

print("Best parameters for SVR:", random_search_svr.best_params_)
print("SVR Test set MSE:", mse_svr)
print("R2 score after tuning : " , r2_svr)

xgb = XGBRegressor()
param_random_xgb = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
```

```

    'learning_rate': [0.01, 0.1, 0.2],
}

random_search_xgb = RandomizedSearchCV(estimator=xgb, param_distributions =
param_random_xgb,n_iter=50,scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=2)
random_search_xgb.fit(x_train, y_train)

best_xgb = random_search_xgb.best_estimator_
y_pred_xgb = best_xgb.predict(x_test)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = metrics.r2_score(y_pred_xgb, y_test)

print("Best parameters for XGBoost:", random_search_xgb.best_params_)
print("XGBoost Test set MSE:", mse_xgb)
print("R2 score after tuning : " , r2_xgb)

linear_reg = LinearRegression()
linear_reg.fit(x_train, y_train)

y_pred_linear = linear_reg.predict(x_test)
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = metrics.r2_score(y_pred_linear, y_test)

print("Linear Regression Test set MSE:", mse_linear)
print("R2 score after tuning : " , r2_linear)

dt = DecisionTreeRegressor()
param_random_dt = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 6],
}

random_search_dt = RandomizedSearchCV(estimator=dt,
param_distributions=param_random_dt,n_iter=50,scoring='neg_mean_squared_error', cv=5, n_jobs=-1,
verbose=2)
random_search_dt.fit(x_train, y_train)

best_dt = random_search_dt.best_estimator_

```

```

y_pred_dt = best_dt.predict(x_test)
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = metrics.r2_score(y_pred_dt, y_test)

print("Best parameters for Decision Tree:", random_search_dt.best_params_)
print("Decision Tree Test set MSE:", mse_dt)
print("R2 score after tuning : ", r2_dt)

#linear regression
regression_report(y_test,y_pred_linear)

#desicion tree
regression_report(y_test,y_pred_dt)

#random forest regression
regression_report(y_test,y_pred_rf)

#svr
regression_report(y_test,y_pred_svr)

#xgb
regression_report(y_test,y_pred_xgb)

import pickle
pickle.dump(Rand, open("model.pkl", 'wb'))
pickle.dump(scaler, open("scale.pkl", 'wb'))

```

Index.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Traffic Volume Estimation</title>
  <style>
    body {
      background-image:

```

```
url('https://images.hindustantimes.com/auto/img/2023/03/16/1600x900/Traffic_jam_1678944991540_1678944991839_1678944991839.jpg');
```

```
background-size: cover;  
background-repeat: no-repeat;  
background-position: center;  
color: #f0f0f0;  
font-family: 'Arial', sans-serif;  
margin: 0;  
padding: 0;
```

```
}
```

```
.container {
```

```
background-color: rgba(0, 0, 0, 0.7);  
padding: 30px;  
border-radius: 15px;  
width: 80%; /* Increased width */  
max-width: 800px; /* Set a maximum width */  
margin: 50px auto;  
box-shadow: 0 4px 20px rgba(0, 0, 0, 0.5);
```

```
}
```

```
h1 {
```

```
text-align: center;  
margin-bottom: 20px;  
font-size: 2.5em;
```

```
}
```

```
section {
```

```
margin-bottom: 40px; /* Space between sections */
```

```
}
```

```
p {
```

```
text-align: center;  
line-height: 1.6;  
margin: 10px 0;
```

```
}
```

```
label {
```

```
display: block;
margin: 10px 0 5px;
font-size: 1.1em;
}
```

```
input, select {
width: 100%;
padding: 12px;
margin-bottom: 15px;
border: 2px solid #4CAF50;
border-radius: 10px;
background: rgba(255, 255, 255, 0.9);
font-size: 1em;
}
```

```
.weather-row, .date-row, .time-row {
display: flex;
justify-content: space-between; /* Space between inputs */
}
```

```
.weather-row input, .date-row input, .time-row input {
flex: 1; /* Equal width for inputs */
margin-right: 10px; /* Space between inputs */
}
```

```
.weather-row input:last-child, .date-row input:last-child, .time-row input:last-child {
margin-right: 0; /* No margin on the last input */
}
```

```
button {
width: 100%;
padding: 12px;
border: none;
border-radius: 10px;
background-color: #4CAF50;
color: white;
font-size: 1.2em;
cursor: pointer;
}
```

```
        transition: background-color 0.3s;
    }

    button:hover {
        background-color: #45a049;
    }

    .output-text {
        font-size: 1.5em;
        text-align: center;
        margin-top: 20px;
        padding: 10px;
        background: rgba(255, 255, 255, 0.1);
        border-radius: 10px;
        border: 1px solid #4CAF50;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>Traffic Volume Estimation</h1>

        <!-- Introduction Section -->
        <section>
            <p>Welcome to the Traffic Volume Estimation application!</p>
            <p>This tool helps you predict traffic patterns based on various parameters such as weather conditions,
            holidays, and time details.</p>
            <p>Input your data and receive a traffic volume prediction!</p>
        </section>

        <!-- Input Form Section -->
        <section>
            <form action="{{ url_for('predict') }}" method="POST">
                <label for="holiday">Select Holiday:</label>
                <select id="holiday" name="holiday">
                    <option value="7">None</option>
                    <option value="1">Columbus Day</option>
                    <option value="10">Veterans Day</option>
```



```
<option value="9">Thanksgiving Day</option>
<option value="0">Christmas Day</option>
<option value="6">New Year's Day</option>
<option value="11">Washington's Birthday</option>
<option value="5">Memorial Day</option>
<option value="2">Independence Day</option>
<option value="8">State Fair</option>
<option value="3">Labour Day</option>
<option value="4">Martin Luther King Jr Day</option>
</select>

<label for="temp">Temperature (°C):</label>
<input type="number" name="temp" placeholder="Enter Temperature" required />

<label>Weather Conditions:</label>
<div class="weather-row">
  <input type="number" min="0" max="1" name="rain" placeholder="Rain (0 or 1)" required />
  <input type="number" min="0" max="1" name="snow" placeholder="Snow (0 or 1)" required />
</div>

<label for="weather">Weather Type:</label>
<select id="weather" name="weather">
  <option value="1">Clouds</option>
  <option value="0">Clear</option>
  <option value="6">Rain</option>
  <option value="2">Drizzle</option>
  <option value="5">Mist</option>
  <option value="4">Haze</option>
  <option value="3">Fog</option>
  <option value="10">Thunderstorm</option>
  <option value="8">Snow</option>
  <option value="9">Squall</option>
  <option value="7">Smoke</option>
</select>

<label>Date:</label>
<div class="date-row">
  <input type="number" min="1" max="31" name="day" placeholder="Day" required />
```

```

        <input type="number" min="1" max="12" name="month" placeholder="Month" required />
        <input type="number" min="2012" max="2022" name="year" placeholder="Year" required />
    </div>

    <label>Time:</label>
    <div class="time-row">
        <input type="number" min="0" max="23" name="hours" placeholder="Hours" required />
        <input type="number" min="0" max="59" name="minutes" placeholder="Minutes" required />
        <input type="number" min="0" max="59" name="seconds" placeholder="Seconds" required />
    </div>

    <button type="submit">Predict Volume</button>
</form>
</section>

<!-- Output Section -->
<section>
    <p id="output-text" class="output-text">{{ prediction_text }}</p>
</section>
</div>
</body>
</html>

```

App.py :

```

import numpy as np
import pickle
import joblib
import matplotlib
import matplotlib.pyplot as plt
import time
import pandas
import os
from flask import Flask, request, jsonify, render_template
from sklearn import preprocessing

app = Flask(__name__)
model = pickle.load(open(r'model.pkl','rb'))

```

```

scale = pickle.load(open(r'scale.pkl','rb'))

@app.route('/')# route to display the home page
def home():
    return render_template('index.html') #rendering the home page
@app.route('/predict',methods=["POST","GET"])# route to show the show predictions in a web UI
def predict():
    # rendering the inputs given by the user
    input_feature=[float(x) for x in request.form.values()]
    features_values= [np.array(input_feature)]
    names = [['holiday','temp','rain','snow','weather','year','month','day','hours','minutes','seconds']]
    data = pandas.DataFrame(features_values,columns=names)

    data_scaled = scale.transform(data)
    data_scaled = pandas.DataFrame(data_scaled, columns = names)
    #predictions using the loaded model file
    prediction=model.predict(data_scaled)
    print(prediction)
    text = "The Estimated Traffic Volume is :'"
    return render_template("index.html",prediction_text = text + str(prediction))
    #showing the predication results in a UI
if __name__=="__main__":
    port=int(os.environ.get('PORT',5000))
    app.run(port=port, debug=True,use_reloader=False)

```

Git Hub & Project Demo Link :

Git Hub Link :

<https://github.com/SaiSathwikaAinampudi/TrafficTelligence-Advanced-Traffic-Volume-Estimation-with-Machine-Learning>

Project Demo Link :

<https://drive.google.com/file/d/1RG15I3vwIMfRRoIT0lyhBrx6uD9HGYK1/view?usp=sharing>