Project Report: TrafficTelligence

Advanced Traffic Volume Estimation with Machine Learning

Team Details

Team ID : LTVIP2025TMID42491

Team Size : 4

Team Leader : M Maheswara Rao

Team member : M P Jeevanandam

Team member : Monika Kupendiran

Team member : N Navya

Phase 1: Brainstorming & Ideation

Objective

TrafficTelligence is an intelligent system that leverages machine learning algorithms to estimate and forecast traffic volume with high accuracy. By integrating historical traffic data, weather patterns, and real-time event inputs, the system aims to support dynamic traffic management, informed urban development, and enhanced commuter navigation.

Scenarios

1. **Dynamic Traffic Management**
   Transportation authorities can utilize real-time traffic predictions to implement adaptive traffic controls, optimize signal timings, and reduce congestion.
2. **Urban Development Planning**
   Urban planners can use forecast data to design roads, transit systems, and commercial areas that accommodate future traffic demands effectively.
3. **Commuter Guidance & Navigation**
   Commuters and navigation applications can access accurate traffic forecasts to plan efficient routes, avoid congested areas, and optimize travel times.

Problem Statement

Urban traffic congestion is intensifying due to population growth, outdated infrastructure, and unpredictable road conditions. Traditional systems rely on static and reactive data, resulting in inefficiencies. There is a clear need for a predictive, data-driven system that facilitates both real-time and strategic decision-making.

Proposed Solution

TrafficTelligence offers a robust, AI-powered platform that processes historical traffic data, real-time sensor input, weather conditions, and event schedules. It delivers precise traffic volume predictions to support immediate responses and long-term planning.

Key Features

- Machine learning-based traffic volume forecasting.
- Real-time adaptive traffic control suggestions.
- Integration with navigation platforms for route optimization.
- Analytical tools for long-term infrastructure planning.

Target Users

- **Traffic Management Authorities** – for live adaptive control.
- **Urban Planners & Government Agencies** – for future-proof design.
- **Navigation App Developers** – for enhanced routing systems.
- **Commuters & Logistics Companies** – for efficient travel and deliveries.

Expected Outcomes

- Reduced congestion through proactive signaling and routing.
- Smarter city planning backed by data insights.
- Improved commuter experience and reduced travel times.
- Environmental gains via emission reduction.
- Cost savings for governments and transportation businesses.

Phase 2: Requirement Analysis

Objective

Define technical and functional specifications for the development of *TrafficTelligence*.

Technical Requirements

- **Languages**: Python (backend & ML), JavaScript (frontend)
- **Frameworks/Libraries**:

    - ML: TensorFlow, Scikit-learn
    - Data Handling: Pandas, NumPy
    - Visualization: Plotly, Matplotlib
    - Web: Flask or Django
    - APIs: Google Maps API, OpenWeatherMap API
- **Tools & Platforms**:
    - Jupyter Notebook / Google Colab
    - Git for version control
    - Cloud Hosting: Heroku / AWS / Google Cloud

Functional Requirements

- Ingest and preprocess multi-source datasets.
- Predict traffic volume using ML models.
- Provide real-time and historical traffic analysis.
- Offer a visual dashboard for monitoring and planning.
- Enable data export for planning departments.

Constraints & Challenges

- **Data Availability**: Accessing consistent and up-to-date datasets.
- **Generalization**: Adapting models to different traffic zones.
- **Integration Complexity**: Merging APIs in real time.
- **Latency**: Ensuring swift real-time responses.
- **Scalability**: Handling growing data and user demands.

<u>Phase 3: Project Design</u>

Objective

Design the architectural and user interaction blueprint for *TrafficTelligence*.

System Architecture

Flow Diagram

(Traffic APIs, Weather APIs, Historical Data)

↓

Data Ingestion & Preprocessing

↓

ML Prediction Engine (Traffic Volume Estimator)

↓

Output Services (Dashboard, API, Reports)

Components

- **Data Ingestion**: Real-time API integrations.
- **ML Model**: Trained regression or time-series model.
- **Backend**: Flask/Django server with RESTful endpoints.
- **Frontend**: Interactive dashboard (Plotly/Dash).
- **Database**: PostgreSQL or SQLite for storing predictions.

User Flow

- **Traffic Authority**: Logs in → Views real-time data → Downloads reports → Adjusts controls.
- **Urban Planner**: Accesses forecast tools → Plans zoning/infrastructure.
- **Commuter**: Receives optimal route suggestions via app/API.

UI/UX Considerations

- Clean dashboard with responsive charts and maps.
- Heatmaps for congestion zones.
- PDF/CSV export.
- API interface for mobile app integration.

Phase 4: Project Planning (Agile)

Objective

Break down project milestones using Agile methodology.

Sprint Planning

- **Sprint 1**: Data collection, cleaning, and EDA. Build baseline model.
- **Sprint 2**: Model training, tuning, and evaluation. Begin UI mockups.
- **Sprint 3**: Backend development and real-time API integration.
- **Sprint 4**: Finalize frontend, deploy system, perform user testing.

Team Roles

- **Data Engineer**: Data collection, preprocessing.
- **ML Engineer**: Model development and evaluation.
- **Backend Developer**: API & server-side logic.
- **Frontend Developer**: Dashboard and UI/UX.
- **Scrum Master**: Task coordination and sprint tracking.

Timeline

Milestone	Week
Data Pipeline Completion	1
Ofline Model Training	2
Functional Dashboard Prototype	
3 Real-time API Integration	
4	
Final Deployment & Demo	5

Phase 5: Project Development

Objective

Implement and integrate all system components into a deployable product.

Technology Stack

- **Languages**: Python, JavaScript
- **ML Libraries**: Scikit-learn, TensorFlow, NumPy, Pandas
- **Visualization**: Plotly, Matplotlib
- **Web Framework**: Flask / Django
- **Frontend**: HTML, CSS, JS (Bootstrap/React optional)
- **Database**: PostgreSQL / SQLite
- **APIs**: Google Maps, OpenWeatherMap

Development Steps

- Set up virtual environment and Git repo.
- Preprocess and align datasets.
- Train ML model for traffic prediction.
- Build REST APIs for model inference.
- Design interactive dashboard with live data.
- Connect frontend to backend APIs.
- Deploy on Heroku or local server.

Challenges & Fixes

Challenge	Fix
Limited real-time data	Used simulated feeds and public APIs
Mismatched timestamps	Standardized timestamps using UTC
Slow prediction performance	Optimized processing pipeline
API rate limits	Implemented caching and retry logic

Phase 6: Functional & Performance Testing

Objective

Ensure system reliability, accuracy, and responsiveness.

Test Cases

- **Prediction Accuracy**: Compared model outputs with known datasets.

- **API Integration**: Verified consistency of external data sources.
- **UI/UX Testing**: Confirmed interactivity and responsiveness.
- **Performance Testing**: Simulated heavy loads on backend APIs.

Bug Fixes

Issue	Resolution
Timezone misalignment	Standardized with UTC
Dashboard not refreshing	Used AJAX for live
updates API latency	Introduced fallback cache

Final Validation

- Fully functional system meeting all initial objectives.
- Accurate predictions and responsive frontend.
- Ready for deployment and demonstration.

Deployment

- **Hosting Platform**: Heroku
- **LiveDemo**:https://screenapp.io/app/#/library/687a17306ddd264021f8aab7/recents/3277b837-0aa7-41c6-95f8571425
- **GitHub Repository**: https://github.com/nnavya570/Traffictelligence

#PROGRAM

```
# importing the necessary libraries
import pandas as pd         # for data manipulation and analysis
import numpy as np          # for numerical computations
import seaborn as sns       # for data visualization
import sklearn as sk        # general scikit-learn import (not typically necessary)
# importing specific modules from sklearn
from sklearn import linear_model # for linear regression, logistic regression, etc.
from sklearn import tree         # for decision tree models
from sklearn import ensemble     # for ensemble methods like RandomForest, GradientBoosting
from sklearn import svm          # for support vector machines

import xgboost                   # for eXtreme Gradient Boosting



data = pd.read_csv(r"/content/traffic volume.csv")


data.head()
```

holiday	temp	rain	snow	weather	date	Time	traffic_volume	
0	NaN	288.28	0.0	0.0	Clouds	02-10-2012	09:00:00	5545
1	NaN	289.36	0.0	0.0	Clouds	02-10-2012	10:00:00	4516
2	NaN	289.58	0.0	0.0	Clouds	02-10-2012	11:00:00	4767

3	NaN	290.13	0.0	0.0	Clouds	02-10-2012	12:00:00	5026

holiday	temp	rain	snow	weather	date	Time	traffic_volume	
4	NaN	291.14	0.0	0.0	Clouds	02-10-2012	13:00:00	4918

data.describe()

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

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 48204 entries, 0 to 48203

Data columns (total 8 columns):

Column Non-Null Count Dtype

--- ------ -------------- -----

0 holiday 61 non-null object

1	temp	48151 non-null	float64
2	rain	48202 non-null	float64
3	snow	48192 non-null	float64
4	weather	48155 non-null	object

5 date 48204 non-null object

6 Time 48204 non-null object

7 traffic_volume 48204 non-null

int64 dtypes: float64(3), int64(1),

object(4)

memory usage: 2.9+ MB

data.isnull().sum()

0	
holiday	48143
temp	53
rain	2
snow	12
weather	49
date	0
Time	0
traffic_volume	0

from collections import Counter # Add this line

data['temp'].fillna(data['temp'].mean(),inplace=True)

data['rain'].fillna(data['rain'].mean(),inplace=True)

data['snow'].fillna(data['snow'].mean(),inplace=True)

print(Counter(data['weather']))

The output shown would typically follow:

Counter({'Clouds': 15144, 'Clear': 13383, 'Mist': 5942, 'Rain': 5665, 'Snow': 2875, 'Drizzle': 1818, 'Fog': 912, 'nan': 49, 'Smoke': 20, 'Squall': 4})

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

Counter({'Clouds': 15144, 'Clear': 13383, 'Mist': 5942, 'Rain': 5665, 'Snow': 2875, 'Drizzle': 1818, 'Haze': 1359, 'Thunderstorm': 1033, 'Fog': 912, nan: 49, 'Smoke': 20, 'Squall': 4})

/tmp/ipython-input-12-2131860540.py:3: 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['temp'].fillna(data['temp'].mean(),inplace=True)

/tmp/ipython-input-12-2131860540.py:4: 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['rain'].fillna(data['rain'].mean(),inplace=True)

/tmp/ipython-input-12-2131860540.py:5: 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['snow'].fillna(data['snow'].mean(),inplace=True)

/tmp/ipython-input-12-2131860540.py:12: 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)

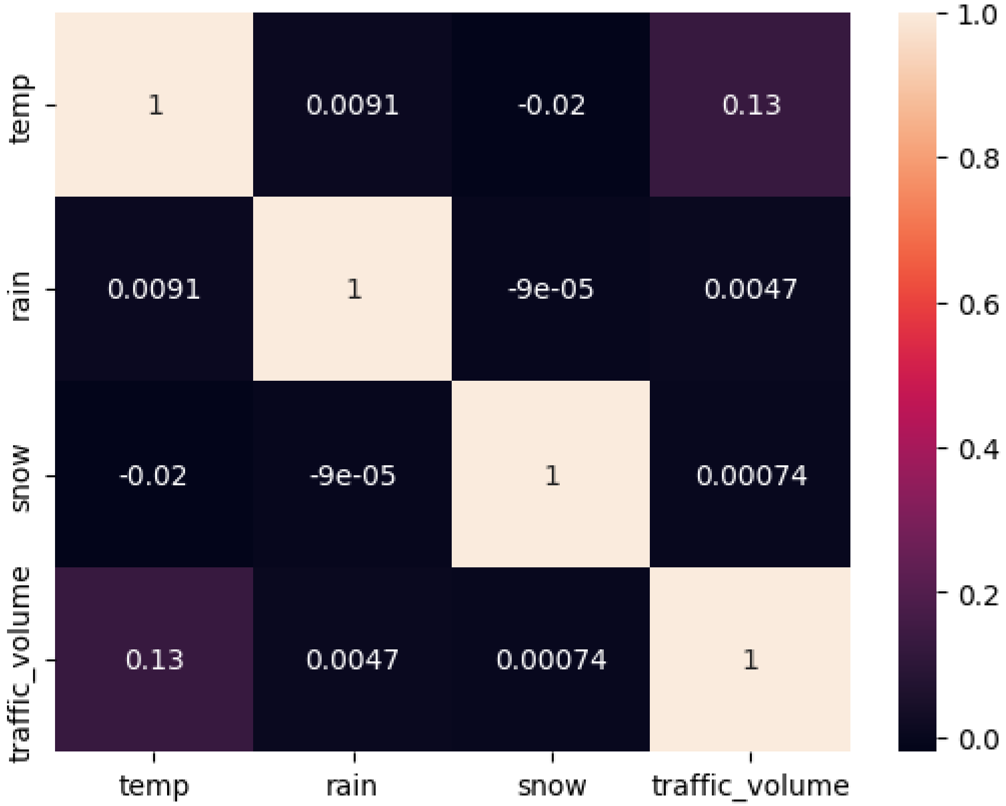
data_numeric = data.select_dtypes(include=[np.number])

correlation_matrix = data_numeric.corr()

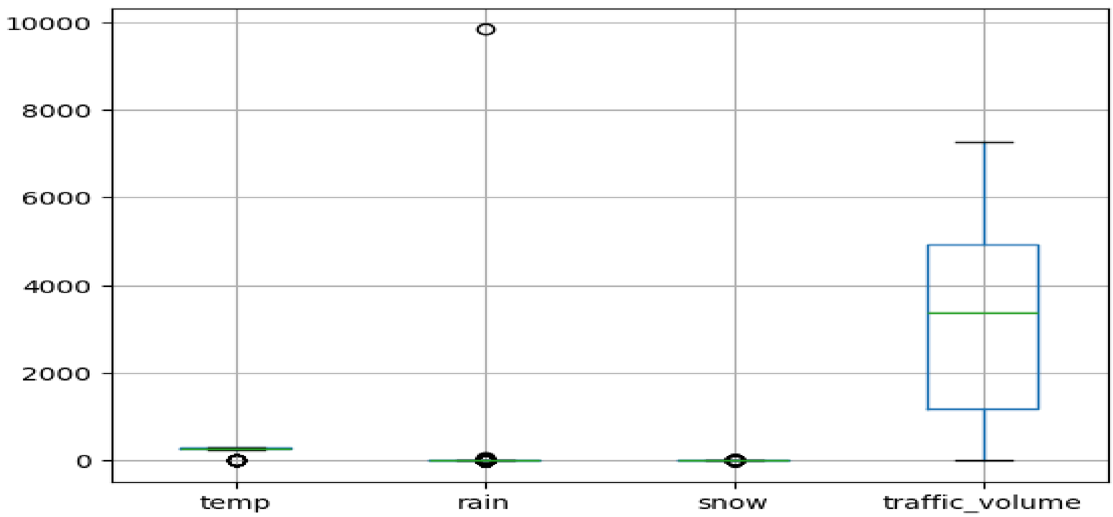
correlation_matrix

temp	**rain**	**snow**	**traffic_volume**	
temp	1.000000	0.009070	-0.019758	0.130034
rain	0.009070	1.000000	-0.000090	0.004714
snow	-0.019758	-0.000090	1.000000	0.000735
traffic_volume	0.130034	0.004714	0.000735	1.000000

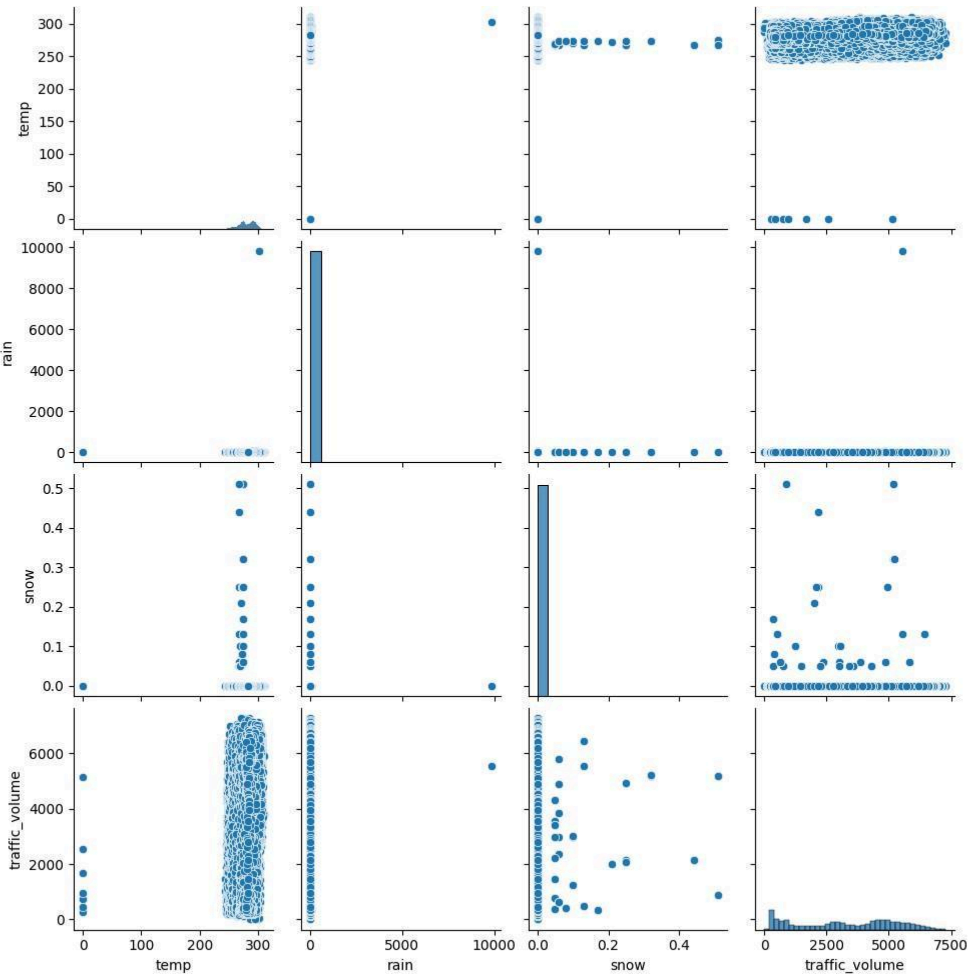
sns.heatmap(correlation_matrix, annot=True)



data.boxplot()



sns.pairplot(data)



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.head()

holi day	te mp	rain	sno w	wea t her	traffic_vo lume	da y	mo nth	ye ar	ho urs	min u tes	seco nds	
0	Na N	288. 28	0.0	0.0	Clouds	55 45	02	10	201 2	09	00	0 0

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

```
y = data['traffic_volume']
x = data.drop(columns=['traffic_volume'], axis=1)
from sklearn.preprocessing import scale

# Encode categorical columns (like 'holiday') using one-hot encoding
x = pd.get_dummies(x, drop_first=True)

# Save column names
names = x.columns

# Scale features
x = scale(x)

# Convert back to DataFrame
x = pd.DataFrame(x, columns=names)

# View top rows
x.head()
```

```
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=0)

from sklearn import linear_model
from sklearn import tree
from sklearn import ensemble
from sklearn import svm
import xgboost

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
# Assuming lin_reg, Dtree, Rand, svr, XGB are already trained
# and x_train, y_train, x_test, y_test are defined.
```

```
# --- Predictions on the Training Set (as per previous context) ---
p1_train = lin_reg.predict(x_train)
p2_train = Dtree.predict(x_train)
p3_train = Rand.predict(x_train)
p4_train = svr.predict(x_train)
p5_train = XGB.predict(x_train)

print("R2 scores on Training Set:")
print(metrics.r2_score(p1_train, y_train))
print(metrics.r2_score(p2_train, y_train))
print(metrics.r2_score(p3_train, y_train))
print(metrics.r2_score(p4_train, y_train))
print(metrics.r2_score(p5_train, y_train))

# --- Predictions on the Test Set ---
# You need to calculate predictions for the test set using x_test
p1_test = lin_reg.predict(x_test) # Corrected: p1 for test data
p2_test = Dtree.predict(x_test)
p3_test =
Rand.predict(x_test) p4_test
= svr.predict(x_test) p5_test =
XGB.predict(x_test)

print("\nR2 scores on Test Set:")
print(metrics.r2_score(p1_test,
y_test))
print(metrics.r2_score(p2_test,
y_test))
print(metrics.r2_score(p3_test,
y_test))
print(metrics.r2_score(p4_test,
```

y_test))

print(metrics.r2_score(p5_test,

y_test))

R2 scores on Training Set:

0.7180334886333546

1.0

0.9687456442212015

-110.267968701714

0.7431479096412659

R2 scores on Test Set:

0.7253816254700689

0.6676879089072176

0.7603567926350155

-109.70984497219168

0.6931703090667725

```
import numpy as np

from sklearn import metrics


# Assuming p3 and y_test are already defined from previous steps

# p3 would be the predictions from the RandomForestRegressor on the test set (Rand.predict(x_test))

# y_test would be the true target values for the test set


# RMSE values

MSE = metrics.mean_squared_error(p3, y_test)


print(np.sqrt(MSE))
```

886.9288543295274

```
from sklearn.preprocessing import LabelEncoder


# Create and fit the encoder

le = LabelEncoder()
```

```
le.fit(y) # 'y' is your target or categorical data
import pickle
pickle.dump(Rand, open("model.pkl", 'wb'))
pickle.dump(le, open("encoder.pkl", 'wb'))
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

app = Flask(_name_)
model = pickle.load(open('/content/model.pkl', 'rb'))
scale = pickle.load(open('/content/model.pkl', 'rb'))

@app.route('/')
def home():
   return render_template('index.html') #rendering the home page


@app.route('/predict',methods=["POST","GET"]) # route to show the predictions in a web
UI def predict():
   # reading 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.reshape(1, -1), columns = names)
    data = scale.fit_transform(data) # Note: scale.fit_tra
data = pandas.DataFrame(data,columns = names)
    # predictions using the loaded model file
    prediction = model.predict(data)
    print(prediction)
    text = "Estimated Traffic Volume is : "
    return render_template('index.html', prediction_text = text + str(prediction)) # showing
the prediction results in a UI


if _name____== "_main_":
    app.run(host='0.0.0.0', port=8000, debug=True) # running the app
    port=int(os.environ.get('PORT',5000))
    app.run(port=port,debug=True,use_reloader=False)
```

Running on all addresses (0.0.0.0)

* Running on http://127.0.0.1:8000

* Running on http://172.28.0.12:8000

INFO:werkzeug:Press CTRL+C to quit

INFO:werkzeug: * Restarting with stat

* Serving Flask app '_main_'

* Debug mode: on

*   Running on

http://127.0.0.1:5000

INFO:werkzeug:Press CTRL+C to

quit

Conclusion

TrafficTelligence exemplifies how machine learning and real-time data integration can address complex urban mobility challenges. The system delivers actionable insights for traffic authorities, strategic planning support for urban developers, and smart navigation tools for commuters.

By following a systematic approach—starting from ideation, requirement analysis, agile development, and testing—the team developed a scalable, efficient, and user-centric solution.

Future Enhancements

- Real-time accident detection via computer vision.
- Multi-modal transit recommendations.
- AI-based adaptive traffic signal control.
- Advanced forecasting using deep learning techniques.

OUTPUT: