TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning

TrafficTelligence is an advanced system that uses machine learning algorithms to estimate and predict traffic volume with precision. By analyzing historical traffic data, weather patterns, events, and other relevant factors, TrafficTelligence provides accurate forecasts and insights to enhance traffic management, urban planning, and commuter experiences.

"Accurate traffic volume estimation is crucial for optimizing traffic management, reducing congestion, and improving travel times. However, traditional methods often rely on manual counting, limited sensor data, and simplistic models, leading to inaccurate estimates and inefficient resource allocation.

"This is where TrafficIntelligence comes in – harnessing the power of Artificial Intelligence (AI) and Machine Learning (ML) to revolutionize traffic volume estimation. Our cutting-edge solution leverages:

- Advanced computer vision techniques to analyze real-time camera feeds
- Machine learning algorithms to identify patterns and anomalies in traffic flow
- Integration with IoT sensors and real-time data feeds for unparalleled accuracy

"With TrafficIntelligence, traffic managers can:

- Accurately estimate traffic volume in real-time
- Identify trends and patterns to inform traffic management decisions
- Optimize traffic signal timing and resource allocation
- Enhance traveler experience and reduce congestion

"Join us in transforming traffic management with AI-powered traffic volume estimation. Discover how TrafficIntelligence can help you make data-driven decisions and create smarter, more efficient transportation systems."

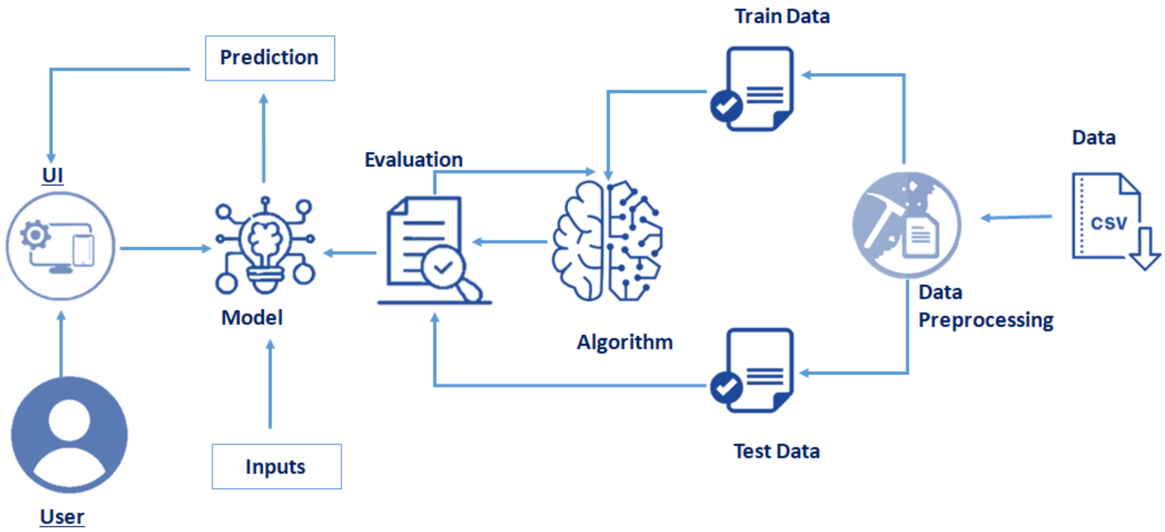
Would you like me to modify the introduction or provide further assistance?

Scenario 1: Dynamic Traffic Management TrafficTelligence enables dynamic traffic management by providing real-time traffic volume estimations. Transportation authorities can use this information to implement adaptive traffic control systems, adjust signal timings, and optimize lane configurations to reduce congestion and improve traffic flow.

Scenario 2: Urban Development Planning City planners and urban developers can leverage TrafficTelligence predictions to plan new infrastructure projects effectively. By understanding future traffic volumes, they can design road networks, public transit systems, and commercial zones that are optimized for traffic efficiency and accessibility.

Scenario 3: Commuter Guidance and Navigation Individual commuters and navigation apps can benefit from TrafficTelligence's accurate traffic volume estimations. Commuters can plan their routes intelligently, avoiding congested areas and selecting optimal travel times based on predicted traffic conditions. Navigation apps can provide real-time updates and alternative routes to improve overall travel experiences.

Technical Architecture



Project Objectives

By the end of this project:

- You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to pre-process/clean the data using different data pre-processing techniques.
- You will able to analyze or get insights into data through visualization.
- Applying different algorithms according to a dataset and based on visualization.
- You will be able to know how to find the accuracy of the model.
- You will be able to know how to build a web application using the Flask framework.

Project Flow

- User interacts with the UI (User Interface) to enter the input values.
- Entered input values are analyzed by the model which is integrated.
- Once the model analyses the input the prediction is showcased on the UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Collect the dataset or Create the dataset
- Data Pre-processing.
  - Import the Libraries.

        - Importing the dataset.
        - Checking for Null Values.
        - Data Visualization.
        - Taking care of Missing Data.
        - Feature Scaling.
        - Splitting Data into Train and Test.
    - Model Building
        - Import the model building Libraries
        - Initializing the model
        - Training and testing the model
        - Evaluation of Model
        - Save the Model
    - Application Building
        - Create an HTML file
        - Build a Python Code
        - Run the App

Project Structure

Create a Project folder that contains files as shown below

Name	Type	Date Modified
.ipynb_checkpoints	File Folder	11-12-2021 13:07
Flask	File Folder	10-02-2022 11:35
templates	File Folder	10-02-2022 11:35
app.py	py File	10-02-2022 11:35
encoder.pkl	pkl File	11-12-2021 13:01
model.pkl	pkl File	11-12-2021 13:01
IBM	File Folder	15-02-2022 14:58
Flask	File Folder	31-01-2022 10:30
traffic volume_Ibm_scoring end point.ipynb	ipynb File	31-01-2022 10:27
Requirements.txt	txt File	15-12-2021 10:57
Traffic volume estimation.docx	docx File	15-02-2022 15:04
traffic volume.csv	csv File	10-12-2021 11:08
traffic volume.ipynb	ipynb File	11-12-2021 13:03

- Flask files consist of template folder which has HTML pages, app.py file and .pkl files which are used for application building
- IBM folder has flask files and scoring endpoint.ipynb- model training code file.
- We need the model which is saved and the saved model in this content is Traffic volume

TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning

- **Import Necessary Libraries**
- It is important to import all the necessary libraries such as pandas, NumPy, matplotlib.

- **Numpy**- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.
- **Pandas-** It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **Seaborn**- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Matplotlib**- Visualisation with python. It is a comprehensive library for creating static,animated, and interactive visualizations in Python
- **Sklearn** – which contains all the modules required for model building.
-

```
# importing the necessary libraries
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
import xgboost
```

- **Importing the Dataset**
- You might have your data in .csv files, .excel files
- Let's load a .csv data file into pandas using read_csv() function.We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- If your dataset is in some other location, Then
- **Data=pd.read_csv(r"File_location/datasetname.csv")**
-

```
# importing the data
data = pd.read_csv(r"G:\AI&ML\ML projects\Traffic_volume\traffic volume.csv")
```

-

  Note: r stands for "raw" and will cause backslashes in the string to be interpreted as actual backslashes rather than special characters.

- If the dataset is in the same directory of your program, you can directly read it, without giving raw as r.
- Our Dataset weatherAus.csv contains the following Columns
- Holiday - working day or holiday
- Temp- temperature of the day
- Rain and snow – whether it is raining or snowing on that day or not
- Weather = describes the weather conditions of the day
- Date and time = represents the exact date and time of the day
- Traffic volume – output column
- The output column to be predicted is Traffic volume.Based on the input variables we predict the volume of the traffic. The predicted output gives them a fair idea of the count of traffic

- # Analyse the data
- **head()** method is used to return top n (5 by default) rows of a DataFrame or series.

```
# displaying first 5 columns of the data
data.head()
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	None	288.28	0.0	0.0	Clouds	02-10-2012	09:00:00	5545
1	None	289.36	0.0	0.0	Clouds	02-10-2012	10:00:00	4516
2	None	289.58	0.0	0.0	Clouds	02-10-2012	11:00:00	4767
3	None	290.13	0.0	0.0	Clouds	02-10-2012	12:00:00	5026
4	None	291.14	0.0	0.0	Clouds	02-10-2012	13:00:00	4918

- **describe()** method computes a summary of statistics like count, mean, standard deviation, min, max, and quartile values.

```
data.describe()
```

- The output is as shown below

```
# used to understand the descriptive analysis of the data
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

- From the data, we infer that there are only decimal values and no categorical values.

info() gives information about the data - paste the image here.

```
# used to display the basic information of the data
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         48204 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
```

- # Analyse the data
- **head()** method is used to return top n (5 by default) rows of a DataFrame or series.

```
# displaying first 5 columns of the data
data.head()
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	None	288.28	0.0	0.0	Clouds	02-10-2012	09:00:00	5545
1	None	289.36	0.0	0.0	Clouds	02-10-2012	10:00:00	4516
2	None	289.58	0.0	0.0	Clouds	02-10-2012	11:00:00	4767
3	None	290.13	0.0	0.0	Clouds	02-10-2012	12:00:00	5026
4	None	291.14	0.0	0.0	Clouds	02-10-2012	13:00:00	4918

- **describe()** method computes a summary of statistics like count, mean, standard deviation, min, max, and quartile values.

```
data.describe()
```

- The output is as shown below

```
# used to understand the descriptive analysis of the data
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

- From the data, we infer that there are only decimal values and no categorical values.

info() gives information about the data - paste the image here.

```
# used to display the basic information of the data
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         48204 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
```

Handling Missing Values

1. The Most important step in data pre-processing is dealing with missing data, the presence of missing data in the dataset can lead to low accuracy.

2. Check whether any null values are there or not. if it is present then the following can be done.

```
# used to display the null values of the data
data.isnull().sum()

holiday            0
temp              53
rain               2
snow              12
weather           49
date               0
Time               0
traffic_volume     0
dtype: int64
```

There are missing values in the dataset, we will fill the missing values in the columns.

3. We are using mean and mode methods for filling the missing values

- Columns such as temp, rain, and snow are the numeric columns, when there is a numeric column you should fill the missing values with the mean/median method. so here we are using the mean method to fill the missing values.

- Weather column has a categorical data type, in such case missing data needs to be filled with the most repeated/ frequent value. Clouds are the most repeated value in the column, so imputing with clouds value.

```
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']))
```

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

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

pa

Handling Missing Values

1. The Most important step in data pre-processing is dealing with missing data, the presence of missing data in the dataset can lead to low accuracy.

2. Check whether any null values are there or not. if it is present then the following can be done.

```
# used to display the null values of the data
data.isnull().sum()
```

```
holiday            0
temp              53
rain               2
snow              12
weather           49
date               0
Time               0
traffic_volume     0
dtype: int64
```

There are missing values in the dataset, we will fill the missing values in the columns.

3. We are using mean and mode methods for filling the missing values

- Columns such as temp, rain, and snow are the numeric columns, when there is a numeric column you should fill the missing values with the mean/median method. so here we are using the mean method to fill the missing values.
- Weather column has a categorical data type, in such case missing data needs to be filled with the most repeated/ frequent value. Clouds are the most repeated value in the column, so imputing with clouds value.

```
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']))
```

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

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

pa

Splitting the Dataset into Dependent and Independent variable

· In machine learning, the concept of the dependent variable (y) and independent variables(x) is important to understand. Here, the Dependent variable is nothing but output in dataset and the independent variable is all inputs in the dataset.

· With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.

To read the columns, we will use iloc of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

Let's split our dataset into independent and dependent variables.

y = data[traffic_volume] - independent

x = data.drop(traffic_volume,axis=1)

```
y = data['traffic_volume']
x = data.drop(columns=['traffic_volume'],axis=1)
```

Feature Scaling

There is a huge disparity between the x values so let us use feature scaling.

Feature scaling is a method used to normalize the range of independent variables or features of data.

```
y = data['traffic_volume']
x = data.drop(columns=['traffic_volume'],axis=1)

names = x.columns

from sklearn.preprocessing import scale

x = scale(x)

x  = pd.DataFrame(x,columns=names)

x.head()
```

	holiday	temp	rain	snow	weather	day	month	year	hours	minutes	seconds
0	0.015856	0.530485	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.345548	0.0	0.0
1	0.015856	0.611467	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.201459	0.0	0.0
2	0.015856	0.627964	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	-0.057371	0.0	0.0
3	0.015856	0.669205	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	0.086718	0.0	0.0
4	0.015856	0.744939	-0.007463	-0.027235	-0.566452	-1.574903	1.02758	-1.855294	0.230807	0.0	0.0

- After scaling the data will be converted into an array form
- Loading the feature names before scaling and converting them back to data frame after standard scaling is applied

Splitting the data into Train and Test

When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test datasets. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

- The train-test split is a technique for evaluating the performance of a machine learning algorithm.

- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.
- In general you can allocate 80% of the dataset to the training set and the remaining 20% to test.
- Now split our dataset into train set and test using train_test_split class from sci-kit learn library.

from sklearn import model_selection x_train,x_test,y_train,y_test=model_selection.train_test_split(x,y,test_size=0.2,random_state =0)

```
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)
```

Model Building

The model building includes the following main tasks

- o Import the model building Libraries
- o Initializing the model
- o Training and testing the model
- o Evaluation of Model
- o Save the Model

Training and Testing the Model

- Once after splitting the data into train and test, the data should be fed to an algorithm to build a model.
- There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms are Regression algorithms.
  1. Linear Regression
  2. Decision Tree Regressor
  3.Random Forest Regressor
  4.KNN
  5.svm
  5.xgboost

Steps in Building the model:-

Initialize the model -

```
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()
```

Fit the models with x_train and y_train -

```
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)
```

Predict the y_train values and calculate the accuracy -

```
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)
```

We're going to use the x-train and y-train obtained above in the train_test_split section to train our Random forest regression model. We're using the fit method and passing the parameters as shown below.

We are using the algorithm from Scikit learn library to build the model as shown below,

Once the model is trained, it's ready to make predictions. We can use the predict method on the model and pass x_test as a parameter to get the output as y_pred.

Notice that the prediction output is an array of real numbers corresponding to the input array.

Model Evaluation

After training the model, the model should be tested by using the test data which is been separated while splitting the data for checking the functionality of the model.

Regression Evaluation Metrics:

These model evaluation techniques are used to find out the accuracy of models built in the Regression type of machine learning models. We have three types of evaluation methods.

- R-square_score
- RMSE – root mean squared error

1. R-squared _score -

Formula

$$R^2 = 1 - \frac{RSS}{TSS}$$

R^2 = coefficient of determination

RSS = sum of squares of residuals

TSS = total sum of squares

It is the ratio of the number of correct predictions to the total number of input samples.

Calculating the r2 score value using for all the models.

```
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))
```

```
-5.517285423636865
1.0
0.9747969952887571
-12.188104231382285
0.8349874938269883
```

```
p2 = Dtree.predict(x_test)
p3 = Rand.predict(x_test)
p4 = svr.predict(x_test)
p5 = XGB.predict(x_test)
```

```
print(metrics.r2_score(p1,y_test))
print(metrics.r2_score(p2,y_test))
print(metrics.r2_score(p3,y_test))
print(metrics.r2_score(p4,y_test))
print(metrics.r2_score(p5,y_test))
```

```
-5.399396398322181
0.6920677009517378
0.8031828166614183
-11.972215715232434
0.7922184852381723
```

- After considering both r squared values of test and train we concluded that random forest regressor is giving the better value, it is able to explain the 97% of the data in train values.
- Random forest gives the best r2-score, so we can select this model.

RMSE –Root Mean Square Error

Formula

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i - \hat{x}_i\right)^2}{N}}$$

RMSD = root-mean-square deviation
i = variable i
N = number of non-missing data points
x_i = actual observations time series
$\hat{x}_i$ = estimated time series

Randforest gives the best r-score value

```
#RMSE values
MSE = metrics.mean_squared_error(p3,y_test)
```

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

```
798.4970439382182
```

RMSE value for Random forest is very less when compared with other models, so saving the Random forest model and deploying using the following process

Save the Model

After building the model we have to save the model.

Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions or transport data over the network. wb indicates write method and rd indicates read method.

This is done by the below code

```python
import pickle

pickle.dump(Rand,open("model.pkl",'wb'))
pickle.dump(le,open("encoder.pkl",'wb'))
```

Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Main Python Script

Let us build an app.py flask file which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.

In order to develop web API with respect to our model, we basically use the Flask framework which is written in python.

Line 1-9 We are importing necessary libraries like Flask to host our model request

Line 12 Initialise the Flask application

Line 13 Loading the model using pickle

Line 16 Routes the API URL

Line 18 Rendering the template. This helps to redirect to the home page. In this home page,we give our input and ask the model to predict

In line 23 we are taking the inputs from the form

Line 28 Feature Scaling the inputs

Line 31 Predicting the values given by the user

Line 32-35 if the output is false render no chance template If the output is True render chance template

Line 36 The value of __name__ is set to __main__ when the module run as the main program otherwise it is set to the name of the module .

```
1      import numpy as np
2      import pickle
3      import joblib
4      import matplotlib
5      import matplotlib.pyplot as plt
6      import time
7      import pandas
8      import os
9      from flask import Flask, request, jsonify, render_template
0
1
2      app = Flask(__name__)
3      model = pickle.load(open('G:/AI&ML/ML projects/Traffic_volume/model.pkl', 'rb'))
4      scale = pickle.load(open('C:/Users/SmartbridgePC/Desktop/AIML/Guided projects/scale.pkl','rb'))
5
6      @app.route('/')# route to display the home page
7      def home():
8          return render_template('index.html') #rendering the home page
9
0      @app.route('/predict',methods=["POST","GET"])# route to show the predictions in a web UI
1      def predict():
2          #  reading the inputs given by the user
3          input_feature=[float(x) for x in request.form.values() ]
4          features_values=[np.array(input_feature)]
5          names = [['holiday', 'temp', 'rain', 'snow', 'weather', 'year', 'month', 'day',
6             'hours', 'minutes', 'seconds']]
7          data = pandas.DataFrame(features_values,columns=names)
8          data = scale.fit_transform(data)
9          data = pandas.DataFrame(data,columns = names)
0           # predictions using the loaded model file
1          prediction=model.predict(data)
2          print(prediction)
3          text = "Estimated Traffic Volume is :"
4          return render_template("index.html",prediction_text = text + str(prediction))
5           # showing the prediction results in a UI
6      if __name__=="__main__":
7
8          # app.run(host='0.0.0.0', port=8000,debug=True)    # running the app
9          port=int(os.environ.get('PORT',5000))
0          app.run(port=port,debug=True,use_reloader=False)
```

Run the App

Open anaconda prompt from the start menu

- Navigate to the folder where your python script is.
- Now type the "python app.py" command

Navigate to the localhost where you can view your web page, Then it will run on **local host:5000**

```
n [1]: runfile('G:/AI&ML/ML projects/Traffic_volume/app.py', wdir='G:/AI&ML/ML
rojects/Traffic_volume')
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
:\Users\SmartbridgePC\anaconda3\lib\site-packages\sklearn\base.py:324:
serWarning: Trying to unpickle estimator StandardScaler from version 0.23.2 when
sing version 1.0.1. This might lead to breaking code or invalid results. Use at
our own risk. For more info please refer to:
ttps://scikit-learn.org/stable/modules/model_persistence.html#security-
aintainability-limitations
 warnings.warn(
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Output

- Copy the HTTP link and paste it in google link tab, it will display the form page
- Enter the values as per the form and click on predict button
- It will redirect to the page based on prediction output
- The output will be displayed in the prediction text as Estimated Traffic volume is in units.

Conclusion: In conclusion, TrafficIntelligence's AI-powered traffic estimation solution represents a paradigm shift in traffic management. By harnessing the power of machine learning, computer vision, and real-time data analytics, we've created a system that provides unparalleled accuracy, efficiency, and scalability.

"With TrafficIntelligence, traffic managers can:

- Make data-driven decisions with confidence
- Optimize traffic signal timing and resource allocation
- Reduce congestion, travel times, and emissions
- Enhance traveler experience and safety

"As we continue to push the boundaries of innovation, we envision a future where traffic management is:

- Predictive: anticipating and mitigating congestion before it occurs

- Adaptive: adjusting to changing traffic conditions in real-time
- Integrated: seamlessly connecting transportation modes and infrastructure