

# Traffic Telligence : Advanced Traffic Volume Estimation with Machine Learning

## Introduction :

Traffic Telligence 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, Traffic Telligence provides accurate forecasts and insights to enhance traffic management, urban planning, and commuter experiences.

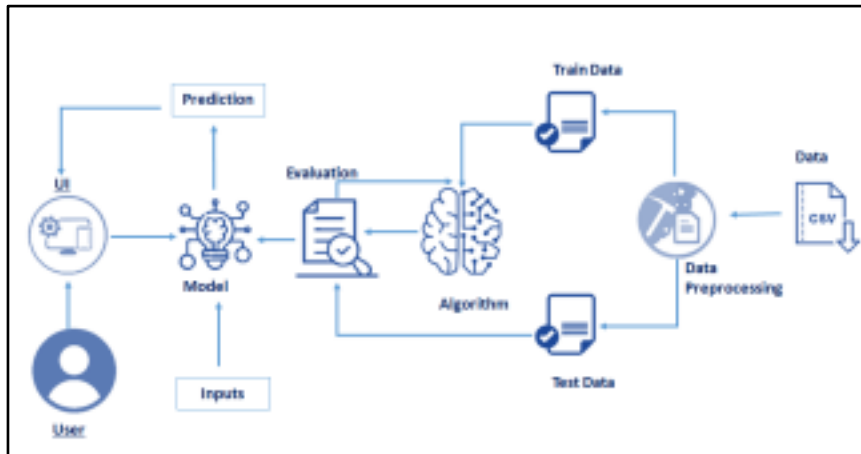
**Scenario 1: Dynamic Traffic Management** Traffic Telligence enables dynamic traffic management by providing real-time traffic volume estimations. Transportation authorities can use this information to implement adaptive traffic control systems, adjust signal timings, and optimize lane configurations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

**Scenario 2: Urban Development Planning** City planners and urban developers can leverage Traffic Telligence 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 Traffic Telligence'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 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



## Milestone 1: 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. Pkl
- Templates folder which contains index.HTML file, chance.HTML file, noChance.HTML file.
- Scale.pkl for scaling, encoder.pkl file for encoding the categorical data, imputer.pkl file for filling out the missing values



Edit with WPS Office

## **Milestone 2: Data Collection:**

ML depends heavily on data, without data, it is impossible for an “AI” model to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training **data set**. It is the actual **data set** used to train the model for performing various actions.\_

### **Download the dataset**

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository etc.

Please refer to the [link](#) given below to download the data set and to know about the dataset

## **Milestone 3: Data Pre-processing**

Data Pre-processing includes the following main tasks

- o Import the Libraries.
- o Importing the dataset.
- o Checking for Null Values.
- o Data Visualization.
- o Feature Scaling.
- Splitting Data into Train and Test.\_

### **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
```

### Activity:1.1 Import important libraries

#### 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")
```

- 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
```



Edit with WPS Office

## 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, 'Wist': 5942, 'Rain': 5665, 'Snow': 2875, 'Drizzle': 1818, 'Fog': 912, nan: 49, 'Smoke': 20, 'Squall': 4})

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





## Data Visualization

Data visualization is where a given data set is presented in a graphical format. It helps the detection of patterns, trends and correlations that might go undetected in text-based data.

Understanding your data and the relationship present within it is just as important as any algorithm used to train your machine learning model. In fact, even the most sophisticated machine learning models will perform poorly on data that wasn't visualized and understood properly.

- To visualize the dataset we need libraries called Matplotlib and Seaborn.
- The Matplotlib library is a Python 2D plotting library that allows you to generate plots, scatter plots, histograms, bar charts etc.

Let's visualize our data using Matplotlib and seaborn library.

Before diving into the code, let's look at some of the basic properties we will be using when plotting.

xlabel: Set the label for the x-axis.

ylabel: Set the label for the y-axis.

title: Set a title for the axes.

Legend: Place a legend on the axes.

1. `data.corr()` gives the correlation between the columns

Correlation is a statistical term describing the degree to which two variables move in coordination with one another. If the two variables move in the same direction, then those variables are said to have a positive correlation. If they move in opposite directions, then they have a negative correlation. -

- Correlation strength varies based on colour, lighter the colour between two variables, more the strength between the variables, darker the colour displays the weaker correlation
- We can see the correlation scale values on the left side of the above image

2. Pair Plot: Plot pairwise relationships in a dataset.

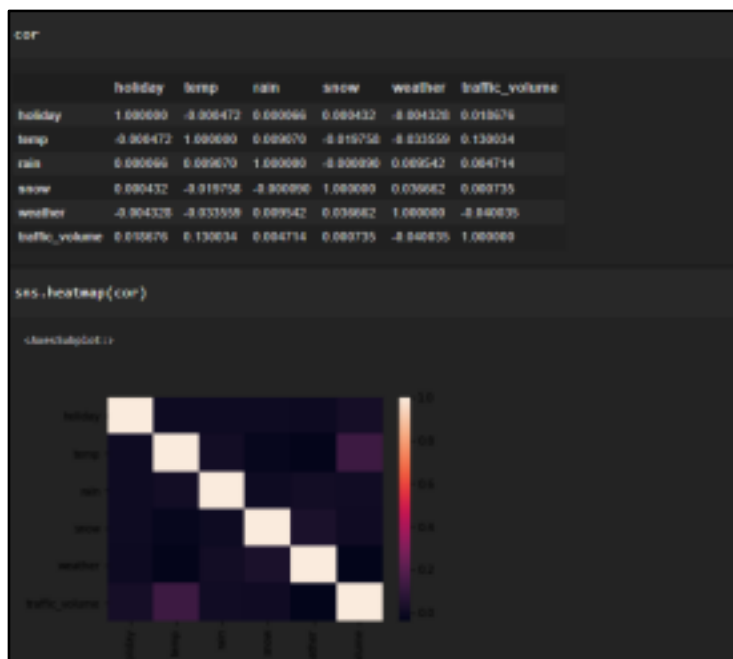
A pair plot is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters. It also helps to form some simple classification models by drawing some simple



Edit with WPS Office

lines or making a linear separation in our data-set.

- By default, this function will create a grid of Axes such that each numeric variable in data will be shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots are treated differently: a univariate distribution plot is drawn to show the marginal distribution of the data in each column.
- We implement this using the below code.



Code:- `sns.pairplot(data)`



Pair plot usually gives pairwise relationships of the columns in the dataset. From the above pair plot, we infer that



Edit with WPS Office

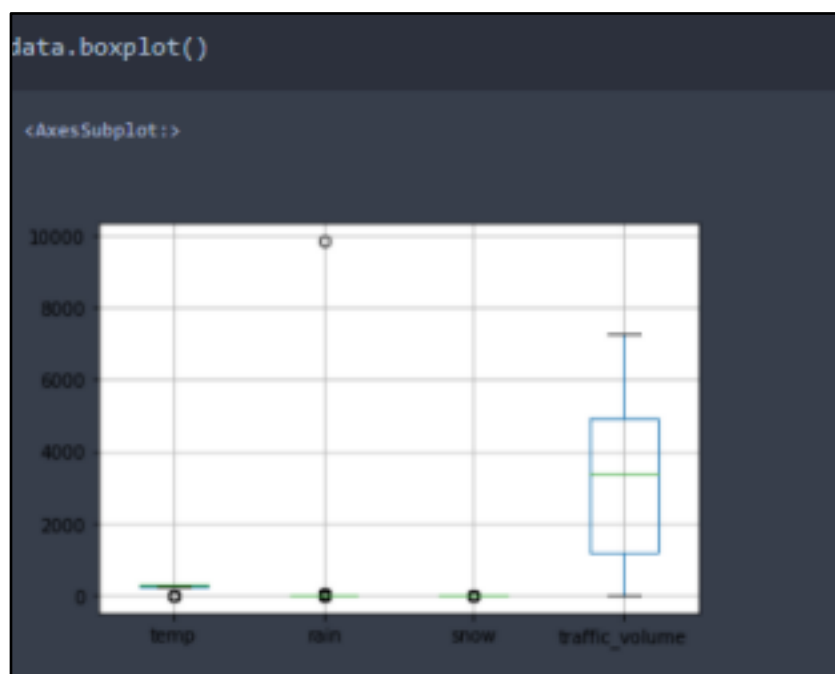
1. From the above plot we can draw inferences such as linearity and strength between the variables. how features are correlated (positive, neutral and negative)

### 3. Box Plot:

Box-plot is a type of chart often used in explanatory data analysis. Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages.

Box plots are useful as they show the average score of a data set. The median is the average value from a set of data and is shown by the line that divides the box into two parts. Half the scores are greater than or equal to this value and half are less.

jupyter has a built-in function to create a boxplot called `boxplot()`. A boxplot plot is a type of plot that shows the spread of data in all the quartiles



From the above box plot, we infer how the data points are spread and the existence of the outliers

4. Data and time columns need to be split into columns so that analysis and training of the model can be done in an easy way, so we use the split function to convert date into the year, month and day. time column into hours, minutes and seconds.



```
# splitting the date column into year,month,day
data[["day", "month", "year"]] = data["date"].str.split("-", expand = True)

# splitting the date column into year,month,day
data[["hours", "minutes", "seconds"]] = data["Time"].str.split(":", expand = True)

data.drop(columns=['date', 'Time'],axis=1,inplace=True)

data.head()
```

	holiday	temp	rain	snow	weather	traffic_volume	day	month	year	hours	minutes	seconds
0	7	288.28	0.0	0.0	1	5545	02	10	2012	09	00	00
1	7	288.36	0.0	0.0	1	4516	02	10	2012	10	00	00
2	7	288.58	0.0	0.0	1	4767	02	10	2012	11	00	00
3	7	290.13	0.0	0.0	1	5026	02	10	2012	12	00	00
4	7	291.14	0.0	0.0	1	4918	02	10	2012	13	00	00

## 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.  $y = \text{data}[\text{traffic\_volume}]$  - independent

$x = \text{data.drop}(\text{traffic\_volume}, \text{axis}=1)$

```
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.



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 have 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)
```



## **Milestone 4: Model Building**

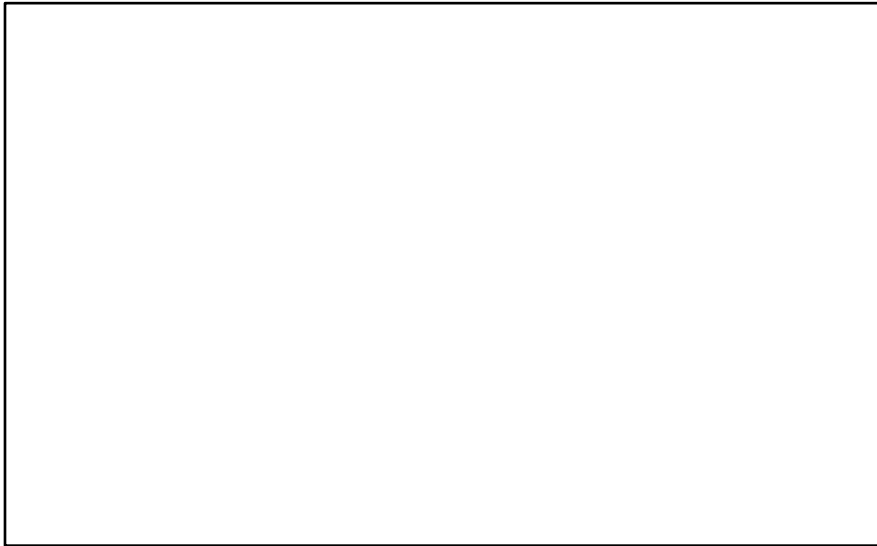
The model building includes the following main tasks

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

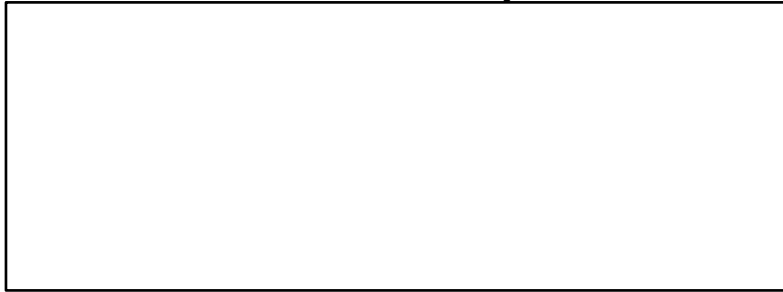
## **Training and Testing the Model**

- 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 -**





Fit the models with `x_train` and `y_train` –



`y_train` values and calculate the accuracy -

Predict the



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.



Edit with WPS Office

## Model Evaluation

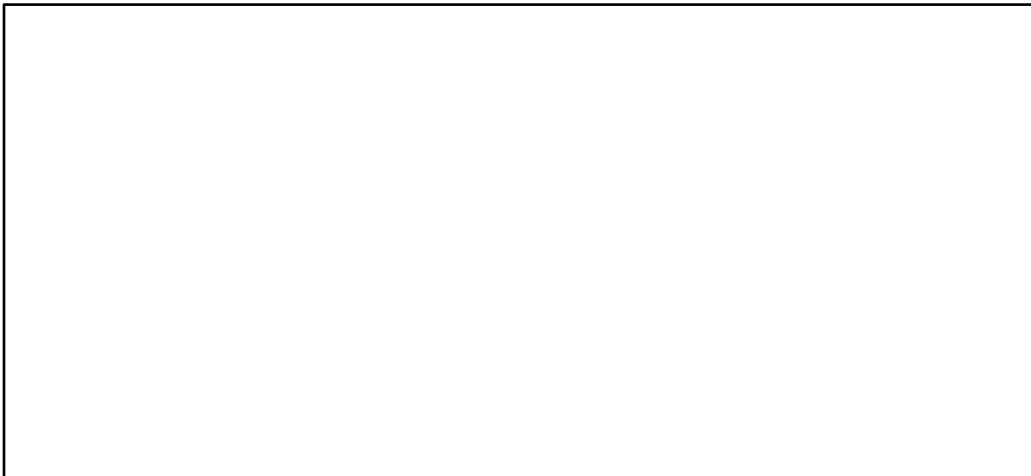
After training the model, the model should be tested by using the test data which is been separated while splitting the data for checking the functionality of the model.

### 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 –

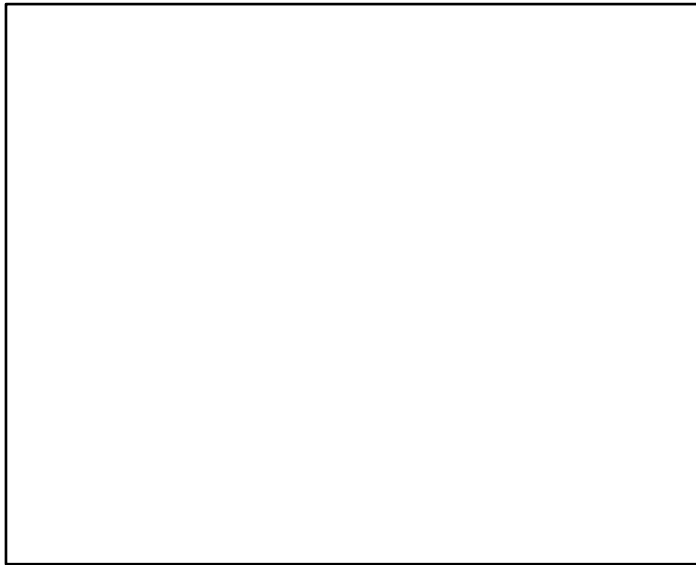


It is the ratio of the number of correct predictions to the total number of input samples.

Calculating the  $r^2$  score value using for all the models.

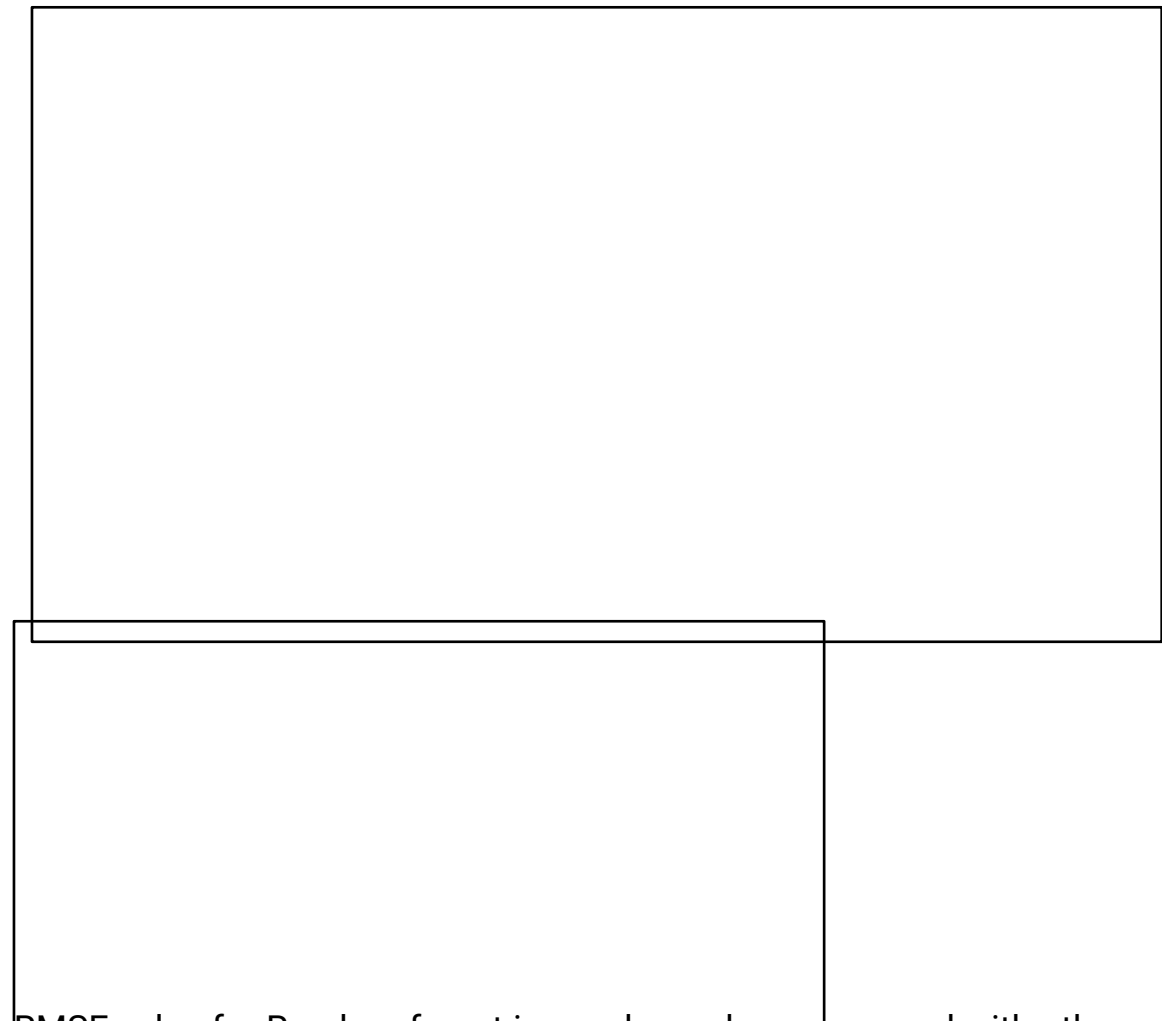






- After considering both  $r^2$  values of test and train we concluded that random forest regressor is giving the better value, it is able to explain the 97% of the data in train values.
  - Random forest gives the best  $r^2$ -score, so we can select this model.
- RMSE –Root Mean Square Error**





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





## **Milestone 6: 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 user where he has to enter the values for predictions. The entered values are given to the saved model and the prediction is showcased on the UI.

This section has the following tasks

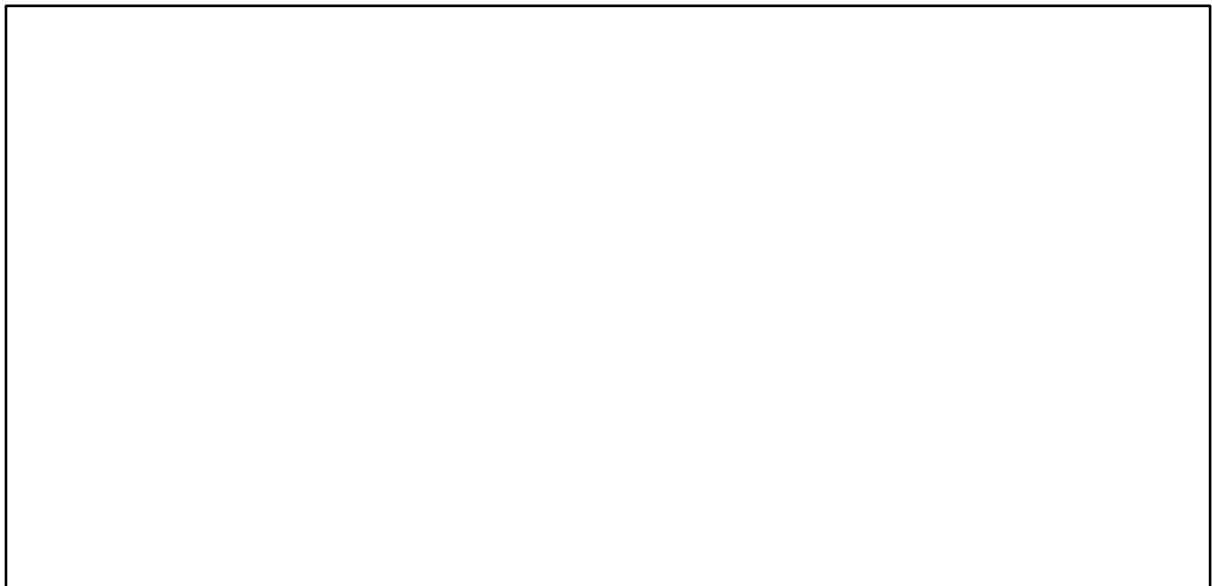
- Building HTML Pages
- Building server-side script

### **Build HTML Code**

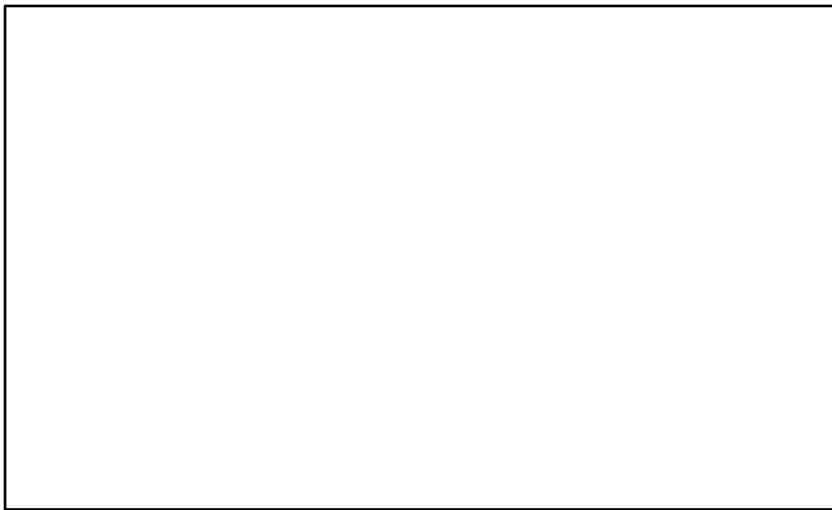
In this HTML page, we will create the front-end part of the web page. On this page, we will accept input from the user and predict the values. For more information regarding HTML, click on the [link](#).

In our project we have HTML files, they are

**1.index.html - paste the image**



2. The HTML page looks like this-

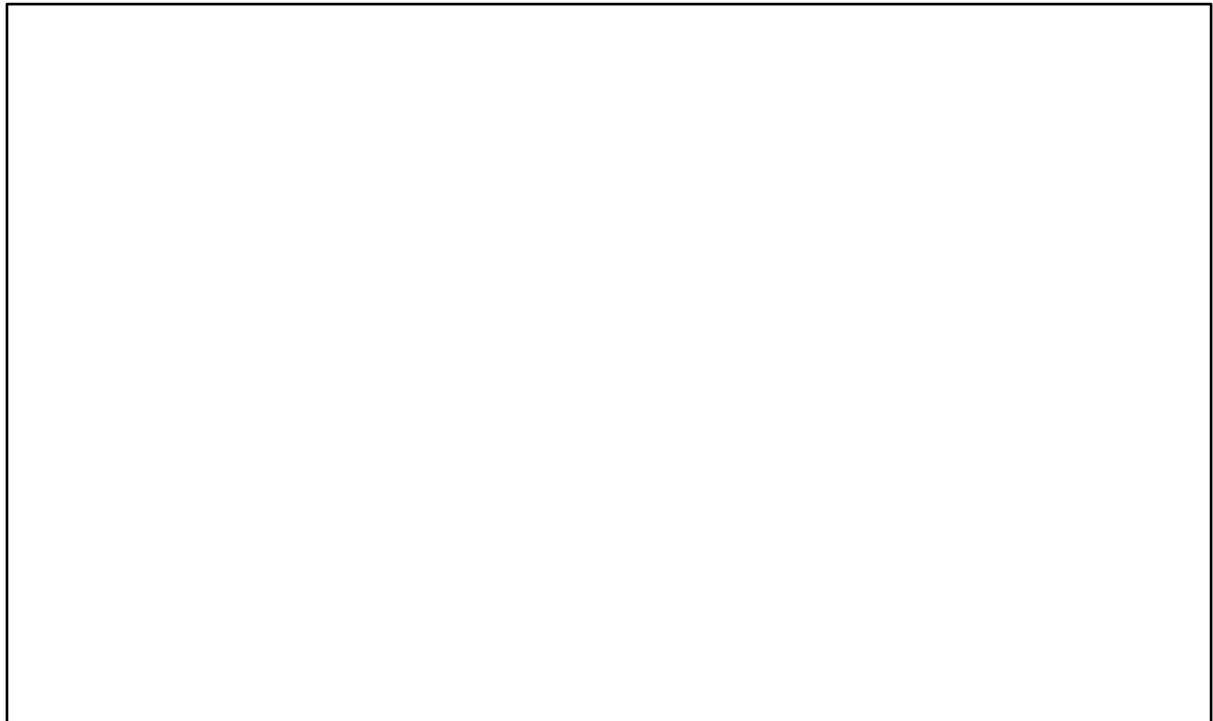


3. It will display all the input parameters and the prediction text will display the output value of the data given by the user.

Milestone 7: Performance Testing



Edit with WPS Office



### 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



Edit with WPS Office

the main program otherwise it is set to the name of the module .



### Run the App

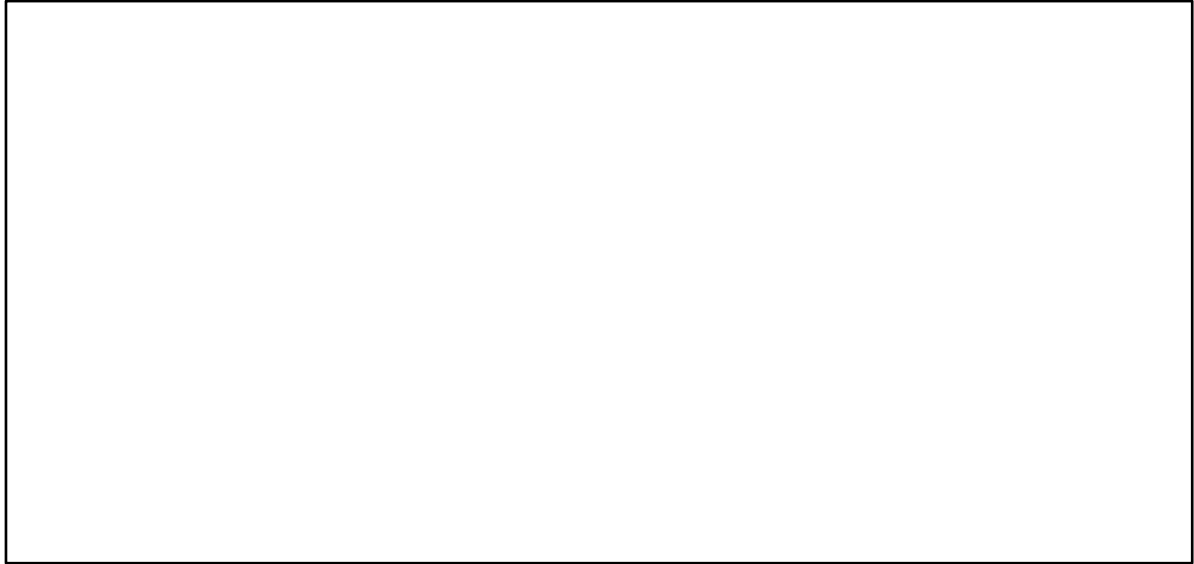
Open anaconda prompt from the start menu

- 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**



Edit with WPS Office



### Milestone 8: Project Demonstration & Documentation

#### 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.

#### Team Members:

- 1) Narayana Yaswanth Sri Naga Satya Sai
- 2) Aluru Chaithanya Kumar
- 3) S Karthik
- 4) Kuruva Harish





Edit with WPS Office