# Technical Design Document

Version 1.0

Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 17th August 2020 | 1.0 | Initial Draft | Supratik Ghosh |

Contributors

The content of this document has been authored with the combined input of the following group of key individuals.

|  |  |
| --- | --- |
| Name | Section Worked Upon |
| Supratik Ghosh | Initial Draft |
| Dudhu Seshubabu | Initial Draft |

Contents

[Technical Design Document 1](#_Toc49704708)

[1. Introduction 3](#_Toc49704709)

[2. Work-flow overall 3](#_Toc49704710)

[3. Workflow Data Ingestion and File Conversion 4](#_Toc49704711)

[4. EDA & Data pre-processing 6](#_Toc49704712)

[5. Model Selection and Model Training 13](#_Toc49704713)

[**5.1.** Flask API creation 17](#_Toc49704714)

[**5.2.** Methods Analysis 20](#_Toc49704715)

[6. Cloud deployment and Dockerization 21](#_Toc49704716)

[7. Application Monitoring(Dashboard) 24](#_Toc49704717)

[8. Hardware Requirements 26](#_Toc49704718)

[Requirements for model training 26](#_Toc49704719)

[Requirements for model testing 26](#_Toc49704720)

[9. Model Retraining approach(strategy) 29](#_Toc49704721)

[Learnings and Obstacles 29](#_Toc49704722)

1. Introduction:

The goal here is to build an end to end Machine learning solution which will predict the hourly traffic volume between Minneapolis and St Paul.

**Problem Statement**

To build a model which will be able to predict the traffic volume of particular hour

Hourly Interstate 94 Westbound traffic volume for MN DoT ATR station 301, roughly midway between Minneapolis and St Paul, MN. Hourly weather features and holidays included for impacts on traffic volume.

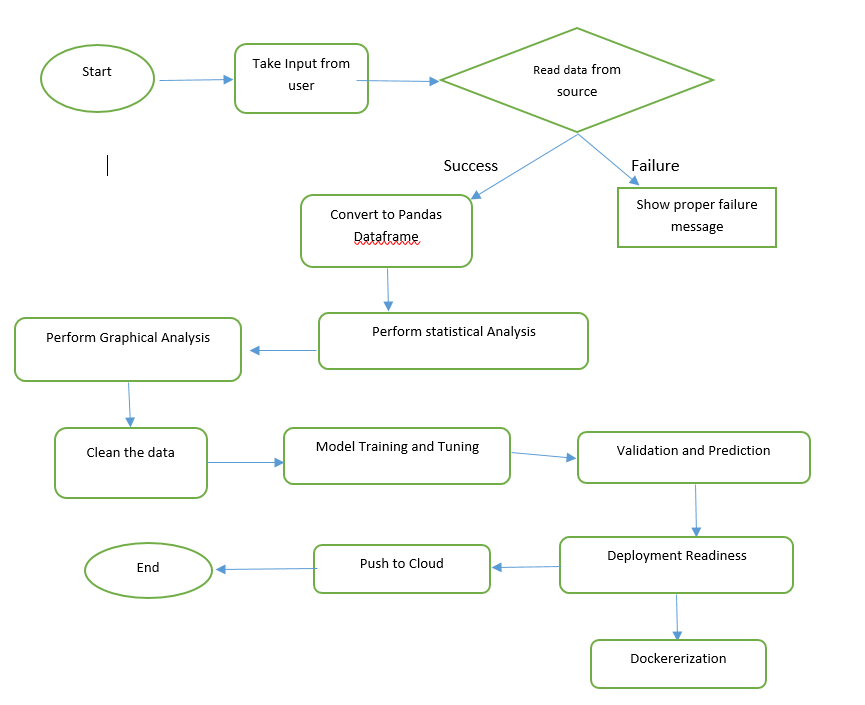
**Note**: All the code will be written in **Python** version 3.7.

High level Objectives:

* Enable Reading/loading of data from the various sources and convert them into pandas dataframe.
* Understand the Features and label columns
* Perform Statistical Analysis on the data and make the data more useful to the ML model.
* Perform Graphical Analysis on the data and make the data more useful for the ML model.
* Perform all the necessary Data cleaning operation.
* After data cleaning operation check with both statistical & graphical analyses for comparison.
* Choose 2-3 different ML models for training.
* Perform Model Tuning.
* Show the Performance Metrics and list the top 3 models for the given dataset.
* Test the performance of the Model.
* Steps of Cloud deployment & Dockerization.

1. Work-flow overall**:**

Our model architecture is given below, this is the generic work-flow we have followed here -

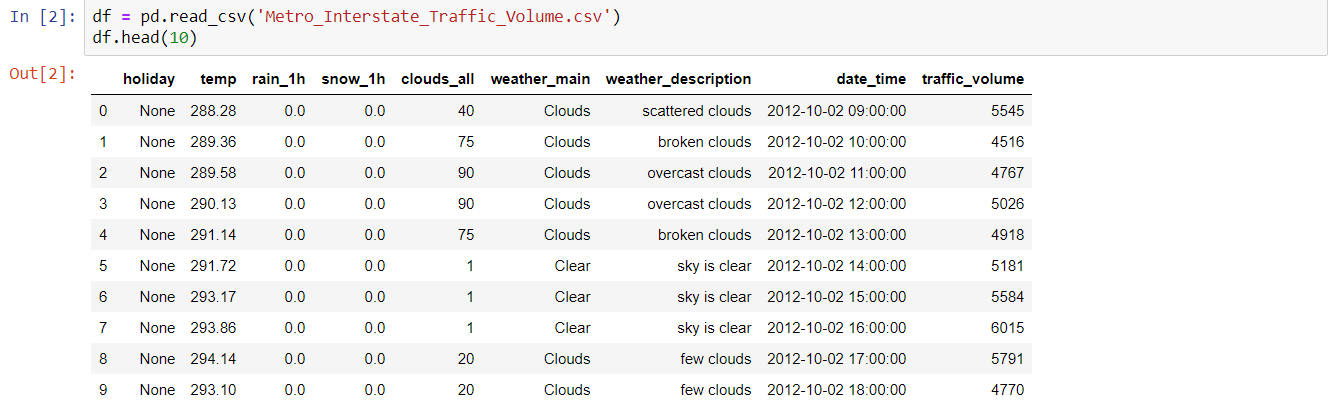


# Workflow Data Ingestion and File Conversion

Data source: CSV file

(data link: <https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume>)

Method of data read: pandas.read\_csv



The columns of the original datasets are below –

'holiday',

'temp',

'rain\_1h',

'snow\_1h',

'clouds\_all',

'weather\_main',

'weather\_description',

'date\_time',

'traffic\_volume'

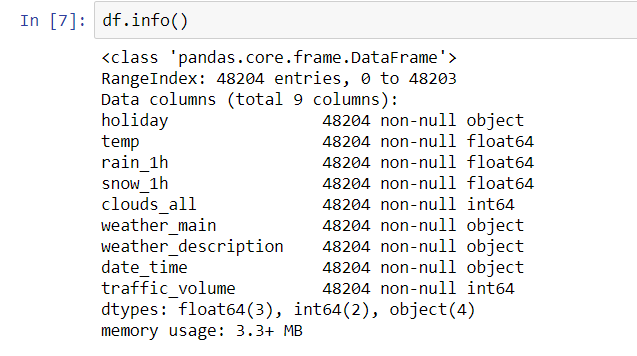
The basic overview of this dataset is depending on different parameter what is the hourly traffic volume on a day (this dataset consist of data from 2nd Oct,2012 to 30th September,2018)

Here, our label column is ‘**traffic\_volume**’.

**Data Description**

* holiday - Categorical US National holidays plus regional holiday, Minnesota State Fair
* temp -Numeric Average temp in kelvin
* rain\_1h -Numeric Amount in mm of rain that occurred in the hour
* snow\_1h -Numeric Amount in mm of snow that occurred in the hour
* clouds\_all -Numeric Percentage of cloud cover
* weather\_main -Categorical Short textual description of the current weather
* weather\_description - Categorical Longer textual description of the current weather
* date\_time - DateTime Hour of the data collected in local CST time
* traffic\_volume Numeric Hourly. Considered to be Target variable.

Initial shape of the dataset is - (48204, 9) and the dataset consists of 5 numerical columns and 4 categorical columns.



Our problem in hand is to predict the probable traffic\_volume on certain day given the value of all the other feature columns such as ('holiday', 'temp', 'rain\_1h', 'snow\_1h', 'clouds\_all', 'weather\_main', 'weather\_description', 'date\_time')

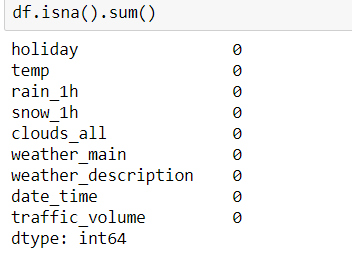
1. EDA & Data pre-processing:

We have to divide our work in hand in different parts –

1. Whether there is any ‘Null’ or ‘NA’ values present in the dataset
2. Whether there is any outlier in the dataset and perform necessary outlier handling techniques.
3. Understand the distribution of the data(traffic volume) depending on various feature columns and conclude which columns are necessary for our prediction.
4. Encode the categorical features
5. Divide the X & Y features.
6. Perform train-test split of the data.
7. Model building and hyper-parameter tuning of the created models.
8. Selecting the best model for our problem.

**Checking for Null values**:

Our dataset didn’t have any null values present.

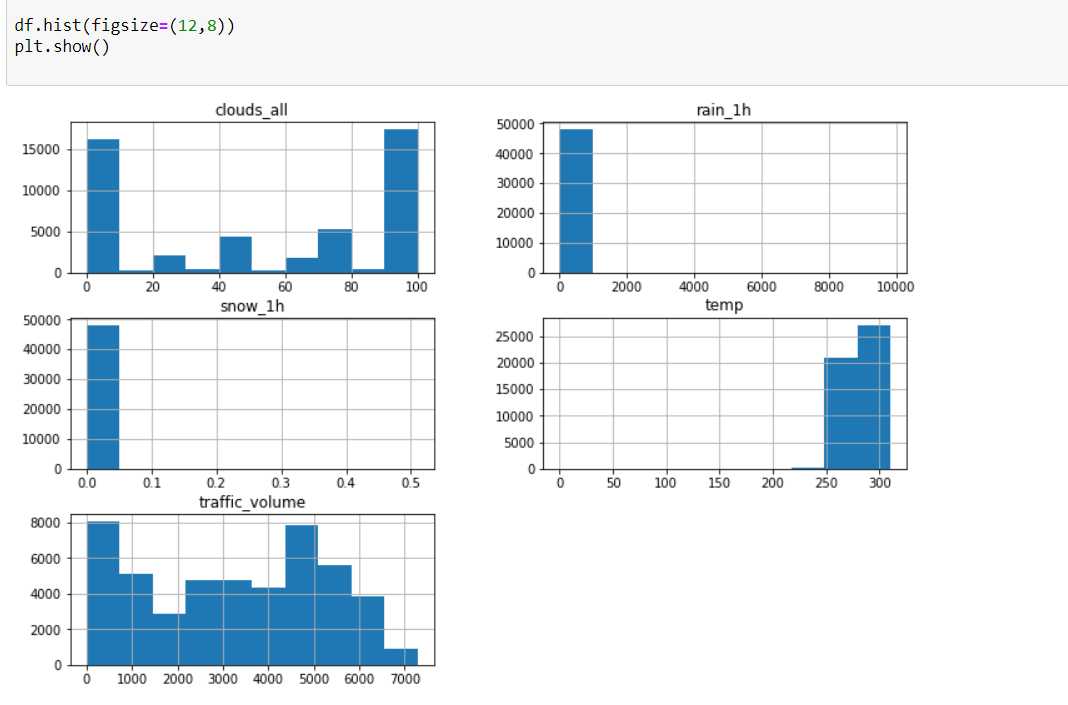


So, we don’t have to drop or perform null value encoding.

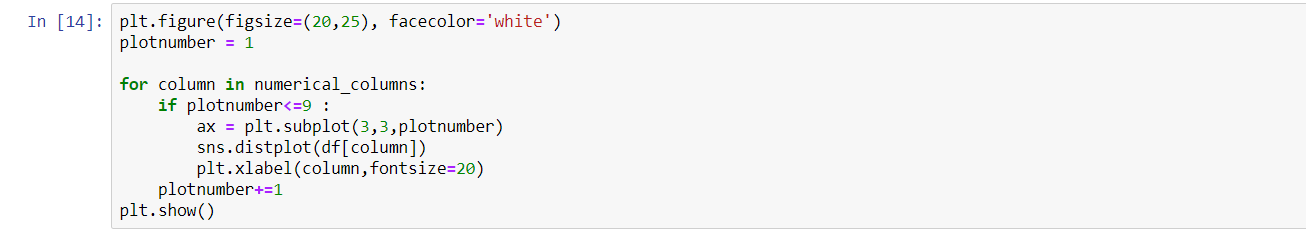
* **Outlier Detection**:

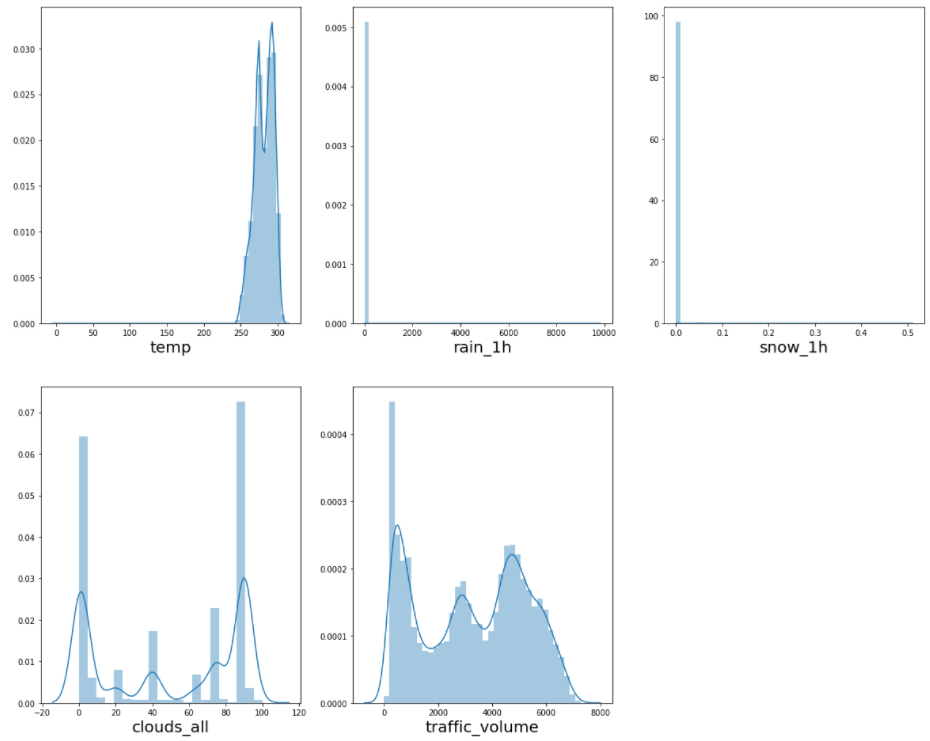
For outlier detection we used Histograms, sns.Boxplot and sns.distribution plot mainly, apart from that also used normal data checking approaches and finding IQR of different columns.

Historgrams:



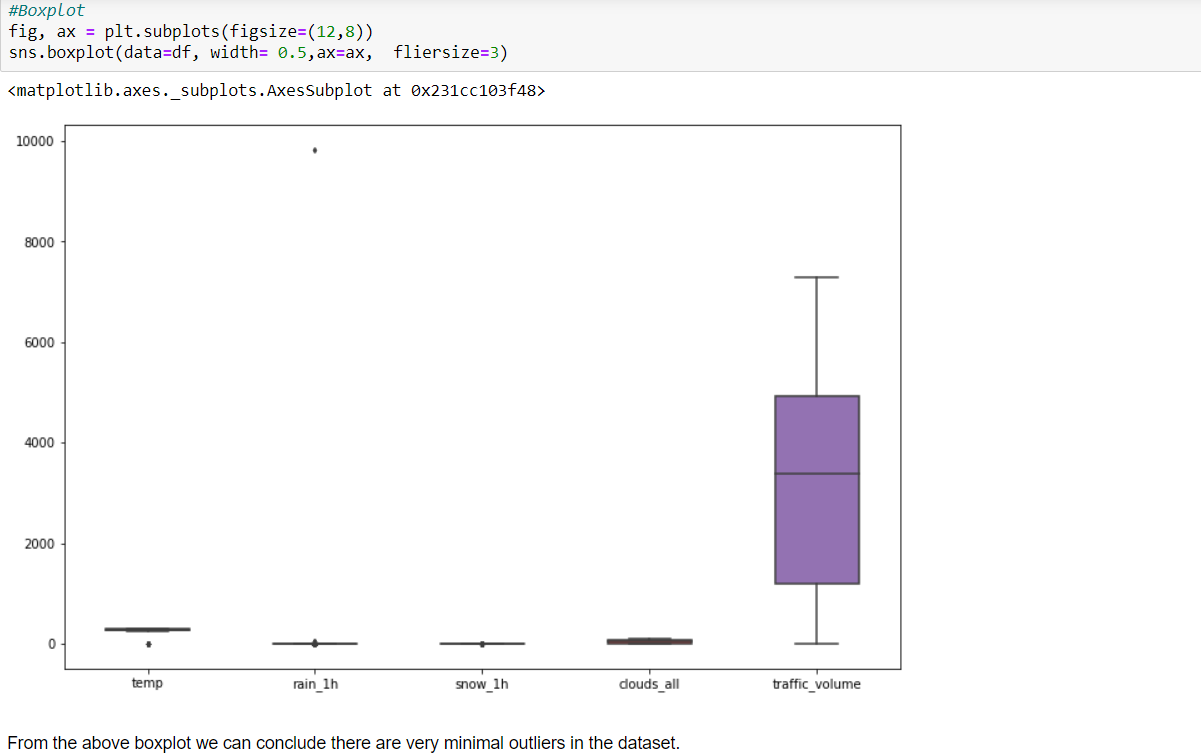
Distribution plots:





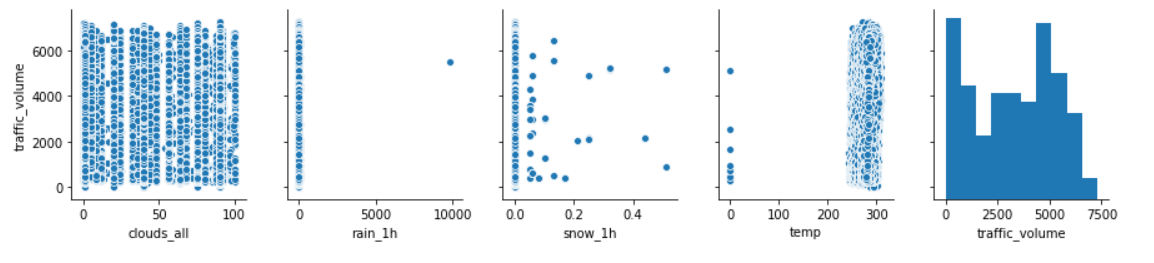
Observations:

1. The ‘temp’ column maximum of its values in between 260 to 300. Probably the temperature being shown here is in ‘Kelvin scale’ (273K = 0 C)
2. ‘rain\_1h’ – the given data shows that these particular place doesn’t have too much of rains. It is mostly ‘No-rain’ or a clear or little rainy weather is being observed in this time-period.
3. ‘snow\_1h’ is also same as rain, in the given time period there was not likely very freezzy weather, since it’s very little amount of snow per day is recorded, mostly no-snowfall.
4. ‘clouds-all’ values in this column ranges between 0 to 100, mostly lot of values are there around 0-25 range and in 70-100 range, this is basically denoting that the weather of this particular place is mostly either Clear sky or Cloudy, further we will see it in the ‘weather\_main’ column which denotes the exact weather condition of the place.
5. ‘traffic volume’ column is distributed mostly within the value range of 200 to 6000, though there are a few high & low values that this range is also available, we will perform Boxplot on this to confirm whether there is outlier in it or not.



From the boxplot we can see it is evident there is very less outliers in dataset. Only in the temp and rain\_1h column a few can be seen, which we have corrected in EDA notebook.

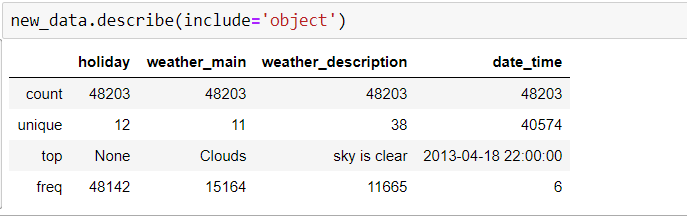
Now quickly we will look into the dispersion and distribution of data with respect to the label column(traffic\_volume)



* **Encoding of the categorical column**:

There are 3 categorical columns in the dataset –

'holiday', 'weather\_main','weather\_description'



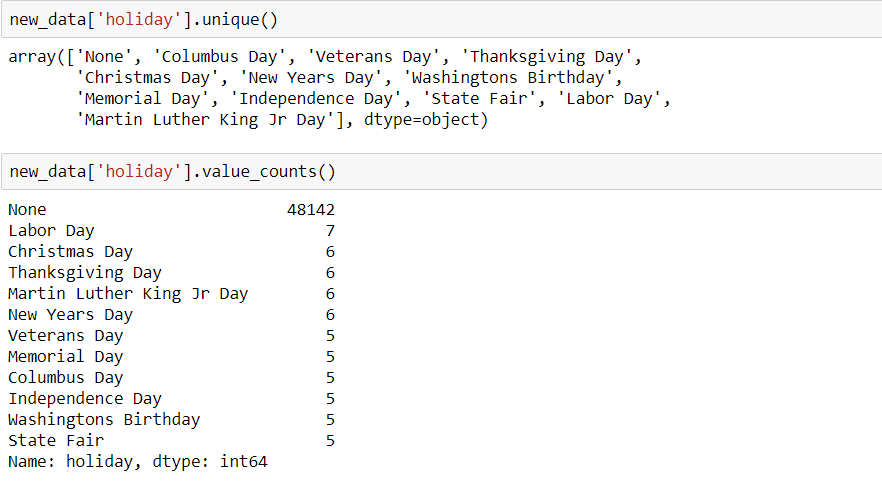
For the categorical columns we will perform 3 tasks in order that is –

1. Check the unique values, b) perform value\_counts() c) barplot and countplot.

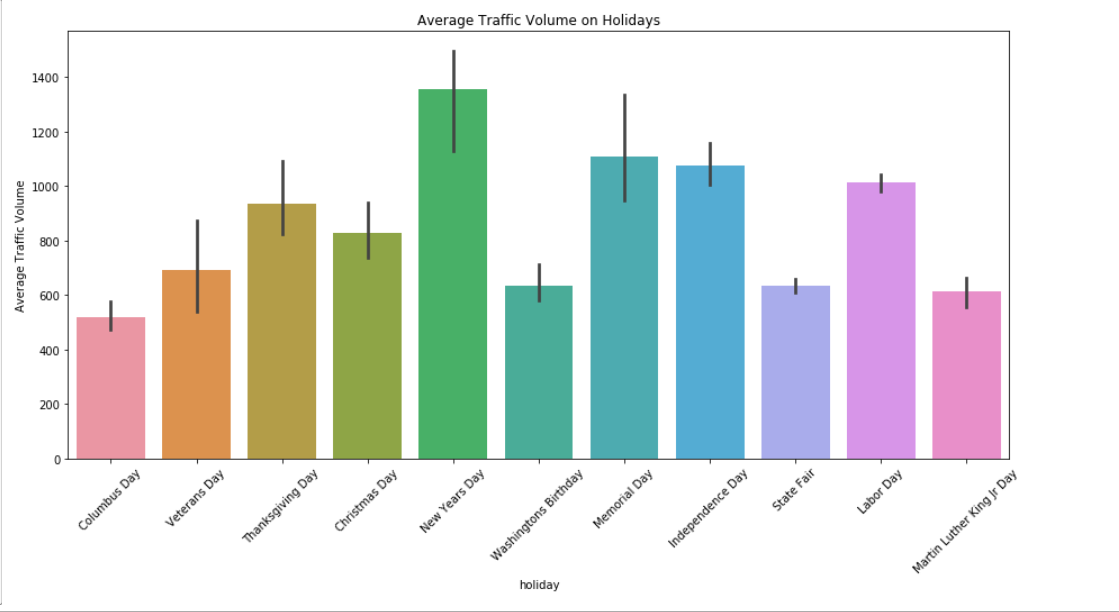
**Holiday**:

In a year consists of 11 holidays, on holidays traffic volume seems to be less than non-holidays. Among that Newyear has more traffic volume compare to other days whereas on colombus day,Washington, Martin LutherKingJr days has very traffic volume.

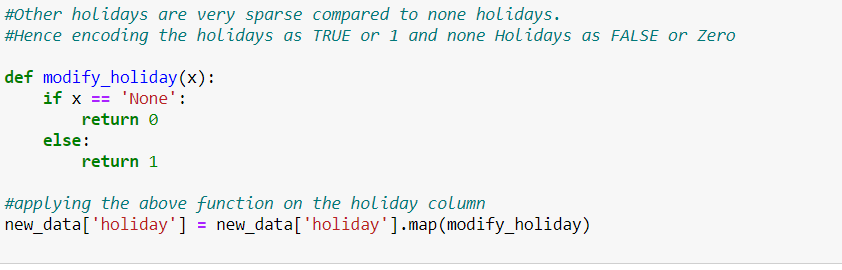
Though Number of holiday is less compare to non-holidays, in order to avoid cold start problem.Transforming the holidays into one group and Non holidays into other group.



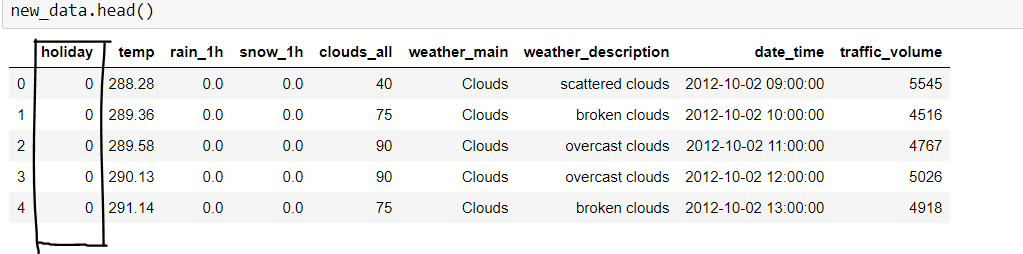
Below plot average traffic volume on each holiday.



Since in our dataset, we can observe Non-holidays are way too much, so we will encode this ‘holiday’ column into two part as holiday & non-holiday, and holidays we should mark it as ‘1’ and non-holidays as ‘0’. Below is the code snippet for that -

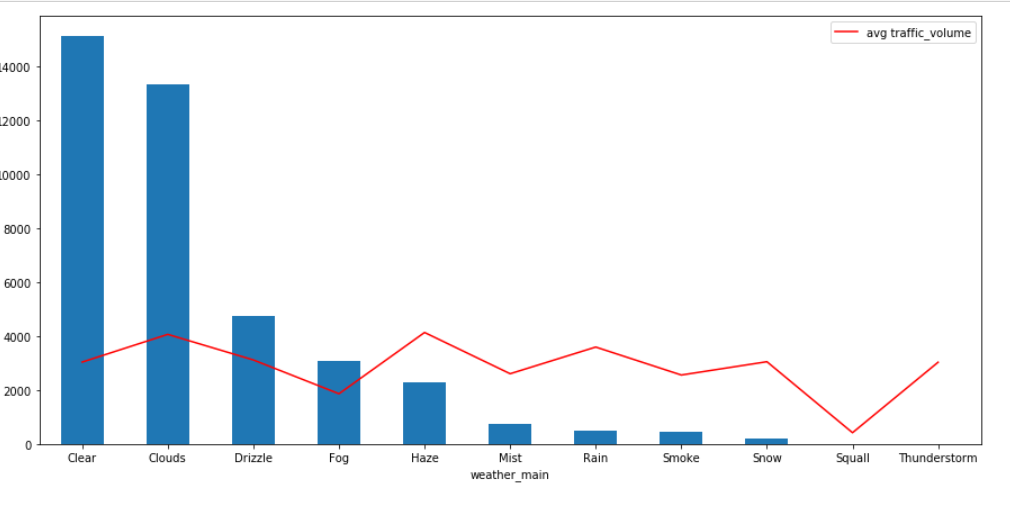


After encoding the holiday column looks like this –



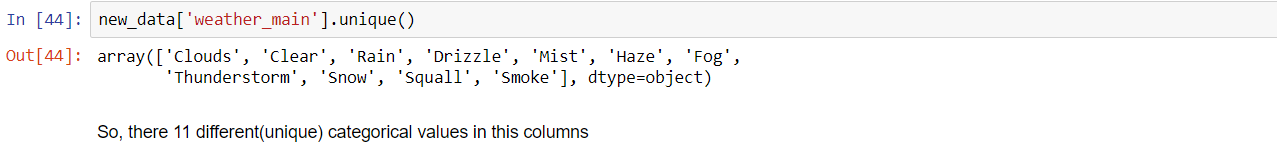
**Weather Main:**

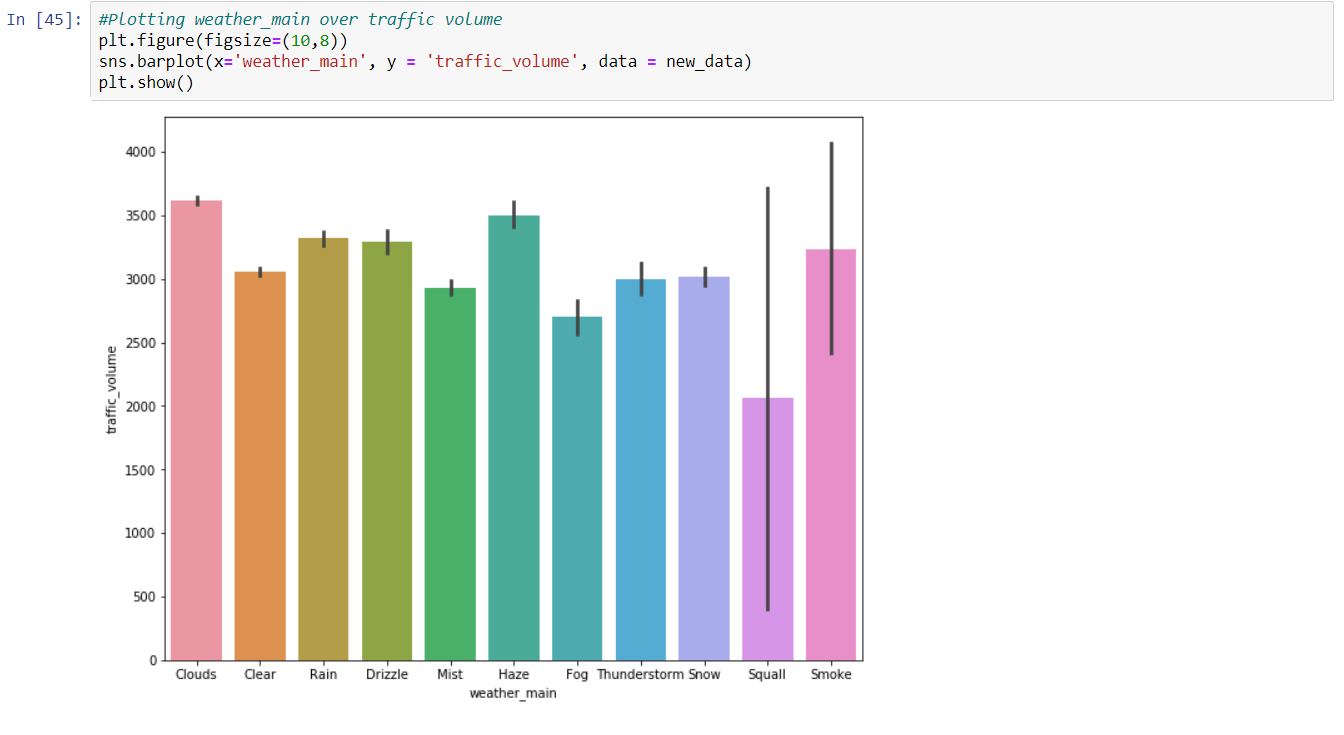
Weather conditions classifies into 11 categories



Average Traffic volume is influenced by different weather condition.

The unique values available in weather\_main column





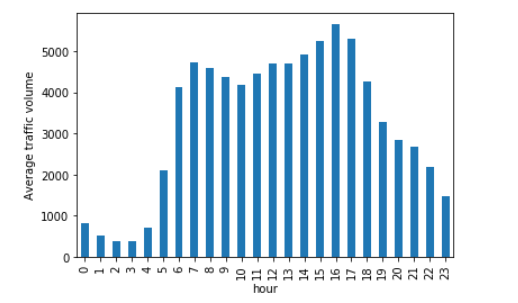
For this column we have to perform ‘One-hot encoding’.

**Weather Description:**

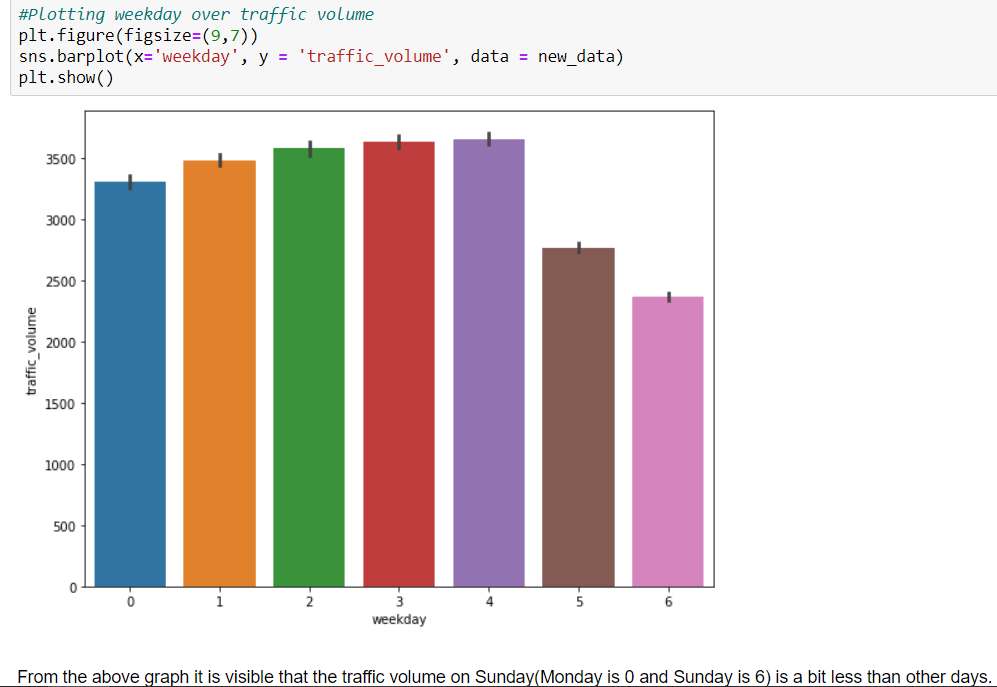
Description about weather conditions in long text format, which categorize into groups in ‘weather\_main’ feature. All weather description is almost same as weather main. Hence, we are dropping this column.

Time based features was created (from the ‘date\_time’ column), traffic volume changes with hour to hour. Volume differs from weekday to week day.

Hours:



During 7AM traffic volume is more than early hours and around 17 PM traffic volume reaches to the peak.



0 refers to Monday & 6 Sunday On weekends traffic volume comparatively less than other weekdays.

1. Model Selection and Model Training:

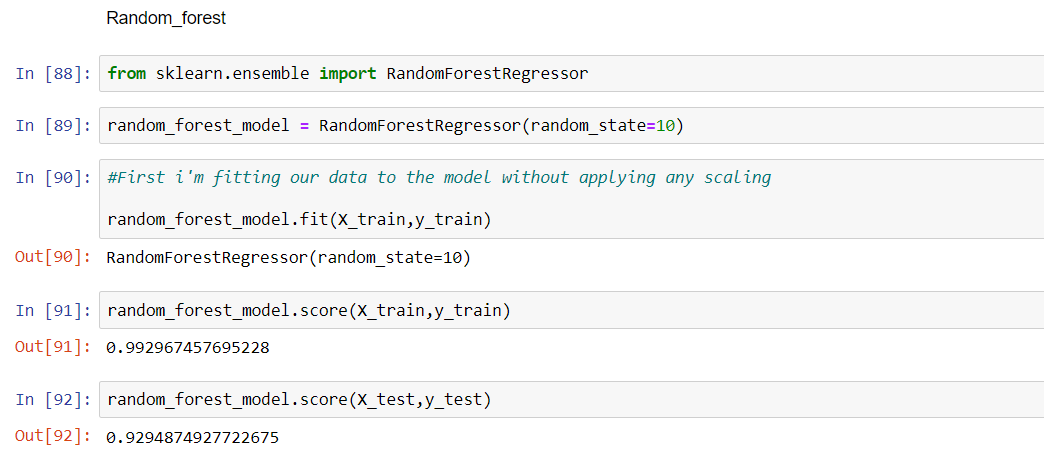
For this project we have chosen Random-Forest, Xgboost and Timeseries based ARIMA , SARIMA models as our initial models, then based on performance we have selected our final model.

Now for model training we have split our data in two parts, train-set & test-set, since this is a time based data, we would first create our train set on the older data then test our model performance on the test data which would be the newer or latest data to understand the performance of our model. Here the data is from 2nd October,2012 to 30th Sept,2018. We are taking our train set data till 31st December 2016, rest will our test data.

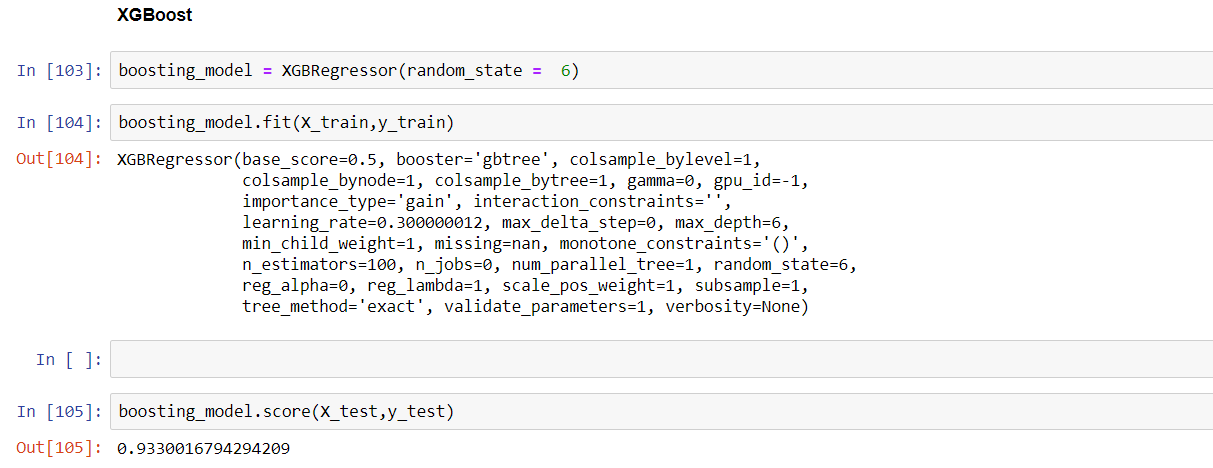


Now we will look one by one model training and their performance –

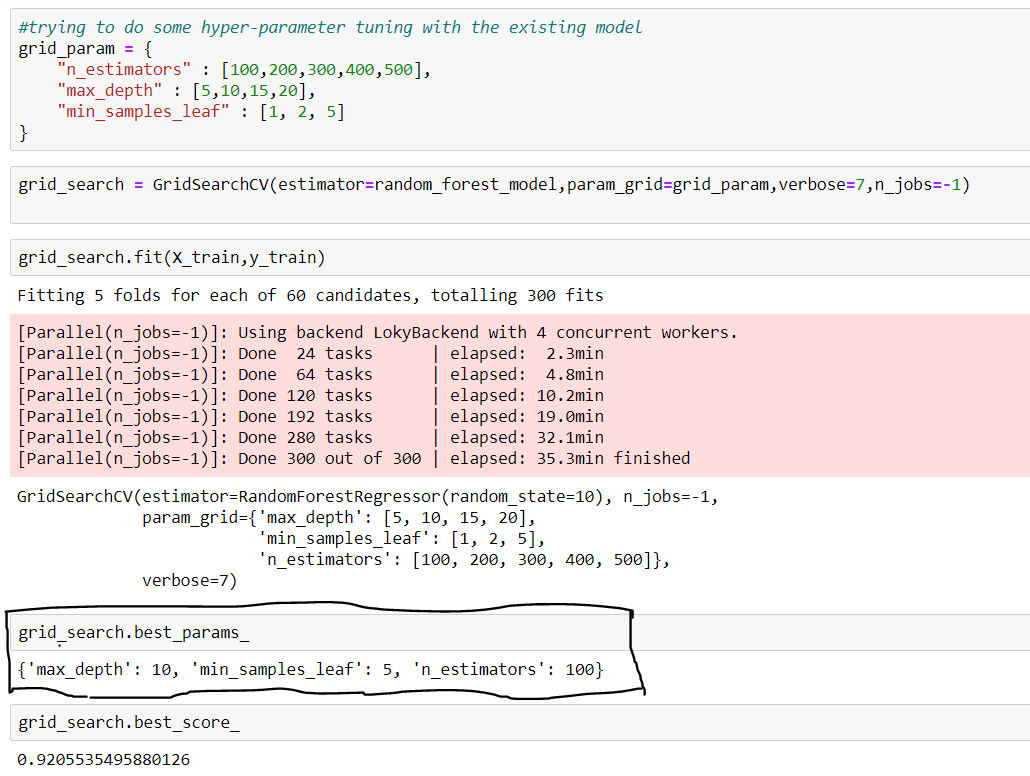
**Random-Forest**:



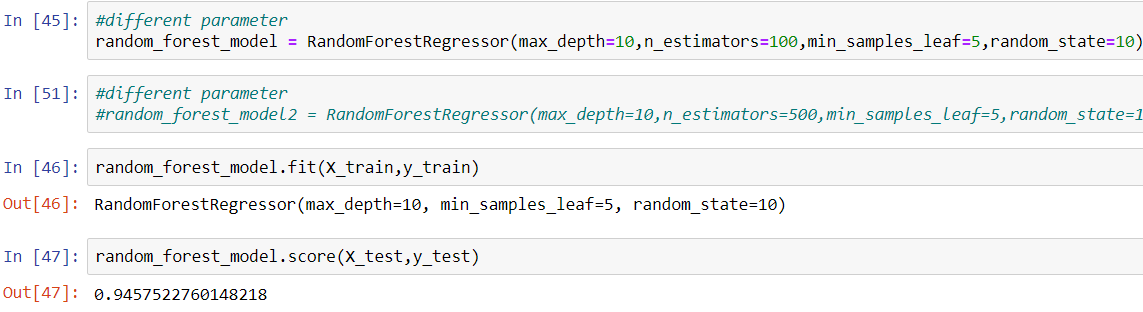
**XGBoost**:



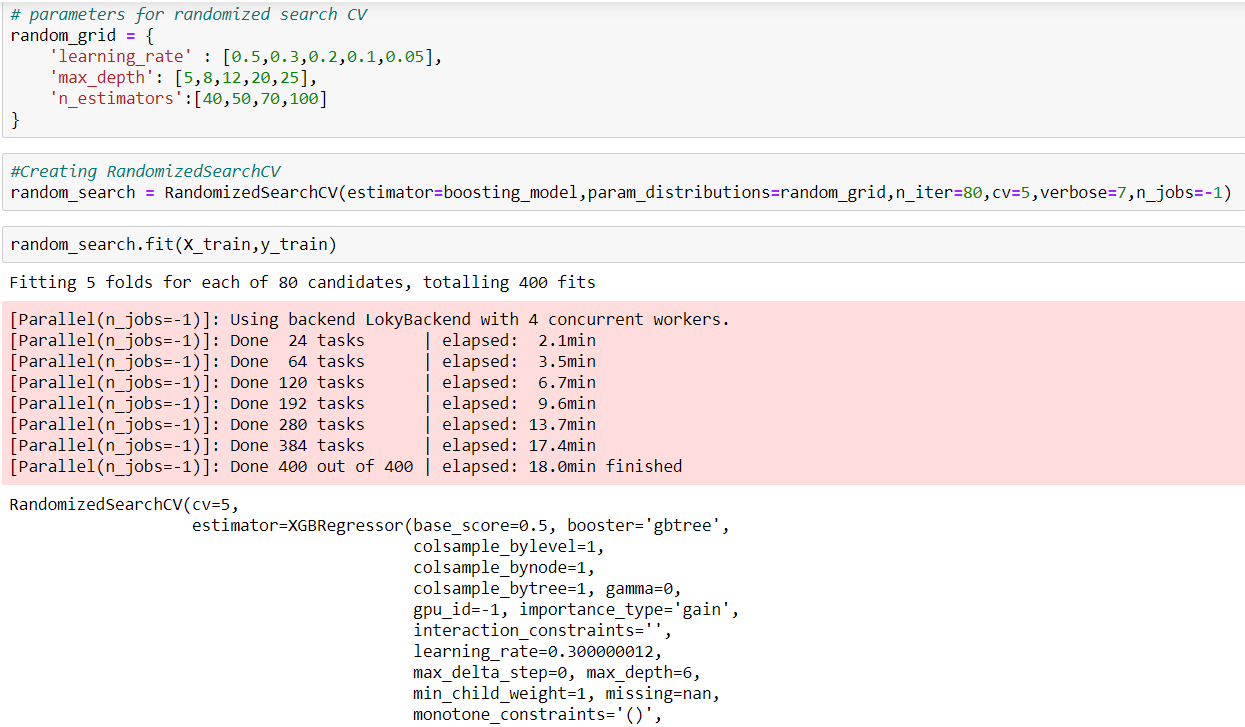
Hyper-parameter tuning:

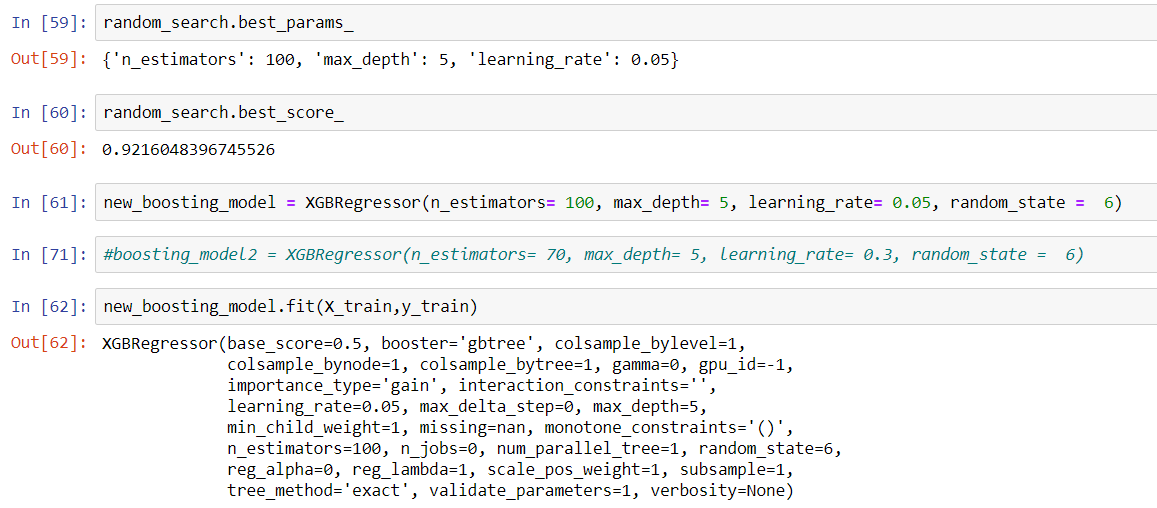


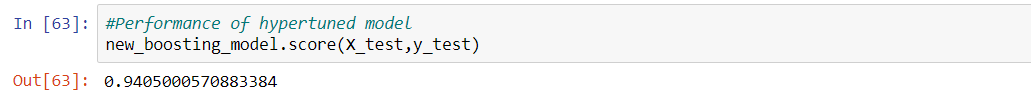
On the basis of the best parameters we will create another random-forest model, let’s see it below –



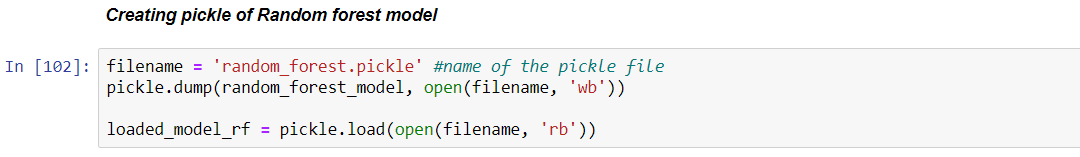
We see improved performance of our model. The same we will do for our Xgboost model, and then compare the best performance & select our final model which we will use for prediction.







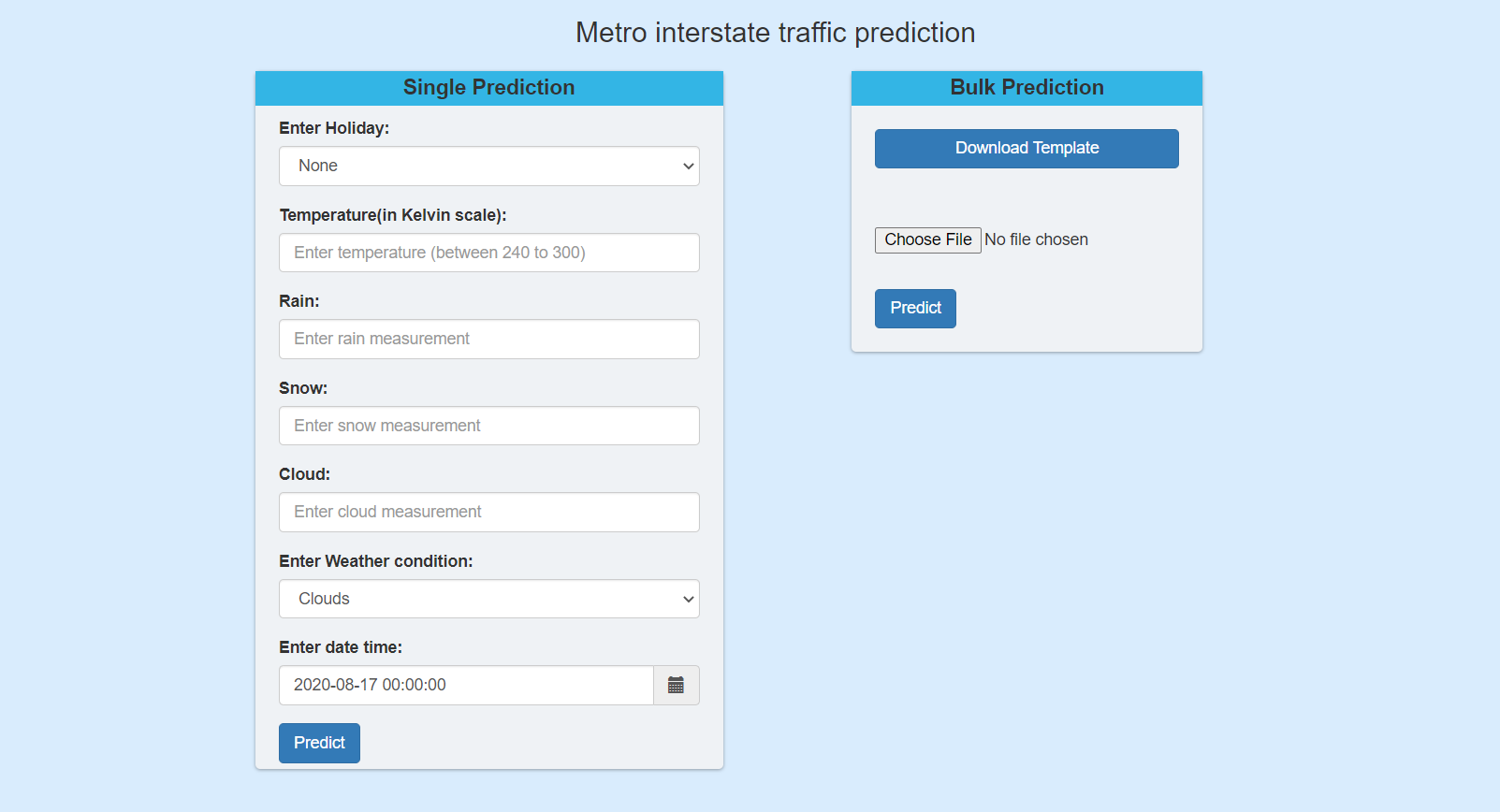
By comparing the performance, we can see random-forest is performing better, hence we will use random-forest algorithm for prediction and we will create a pickle file of it.



Now we will combine all this into our project, and create it pycharm. We will build flask app for creating APIs

* 1. Flask API creation:

Our end goal was to create a web UI where user can either opt for individual parameter based prediction or bulk prediction using a CSV file. Our UI looked like below –



In order to build this web UI, we have to create 4 different API’s and 2 html pages.

Our API’s are –

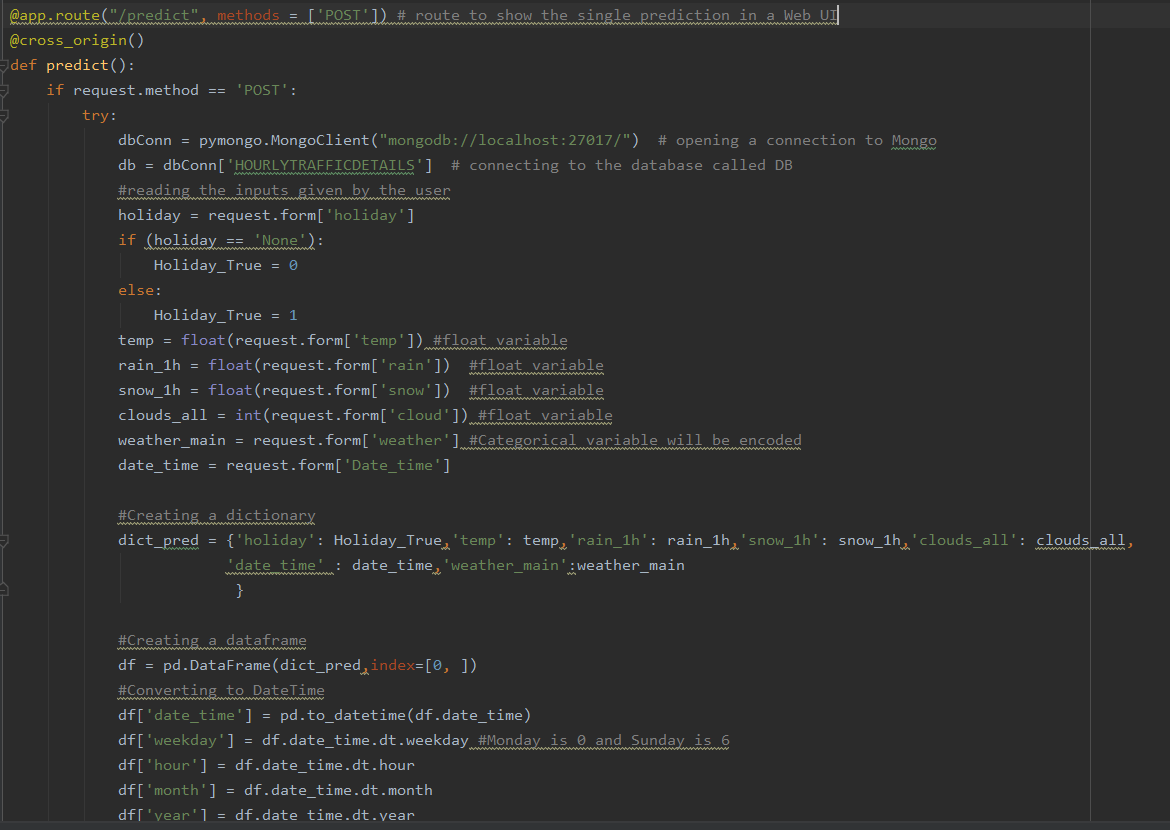
1. Homepage(/) -> route to display home page
2. /download -> to download a template CSV file for bulk prediction
3. /predict -> route to show the single prediction in a Web UI
4. /predict\_file -> route to show bulk prediction

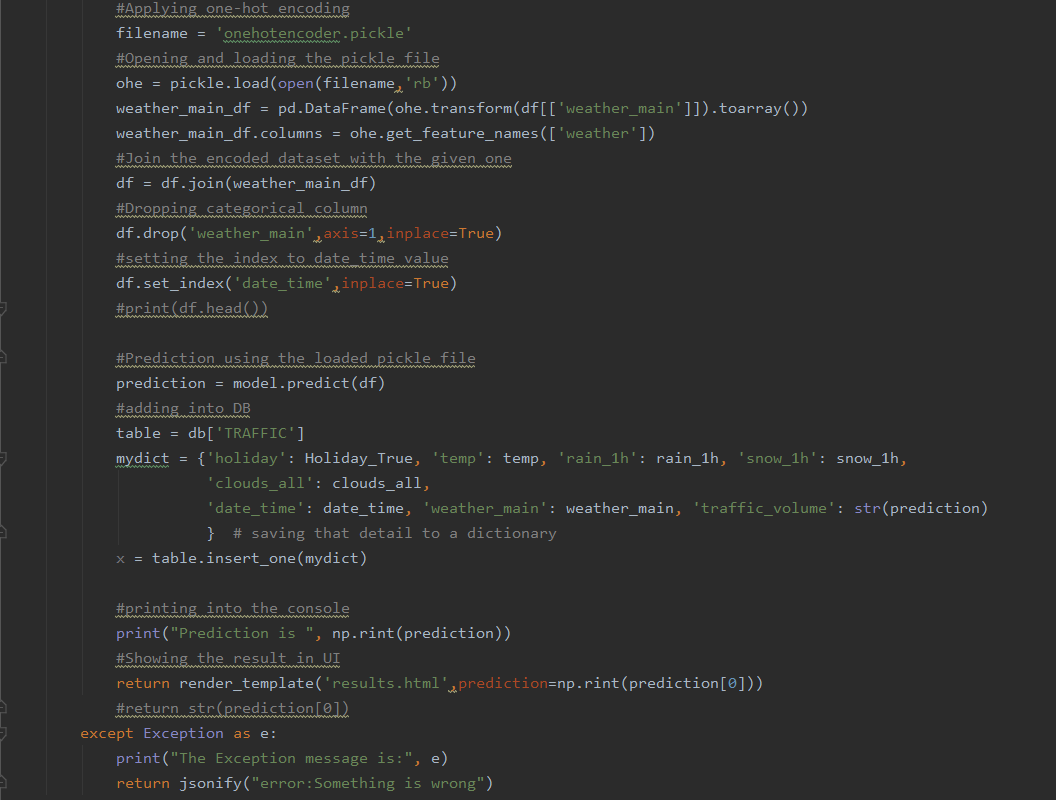
We will understand the work flow of the two main API/ routes that is ‘/predict’ and ‘/predict\_file’

‘/predict’ -> used for single prediction

Steps involved:

1. Reads the inputs from the user
2. Converts into a dataframe
3. Extracts weekday/ Dayofweek, hour, month, year from the ‘date\_time’ column
4. Performs one-hot encoding for the ‘weather\_main’ column & drop the categorical column
5. perform prediction using the random-forest model pickle file
6. return back the result to the web UI

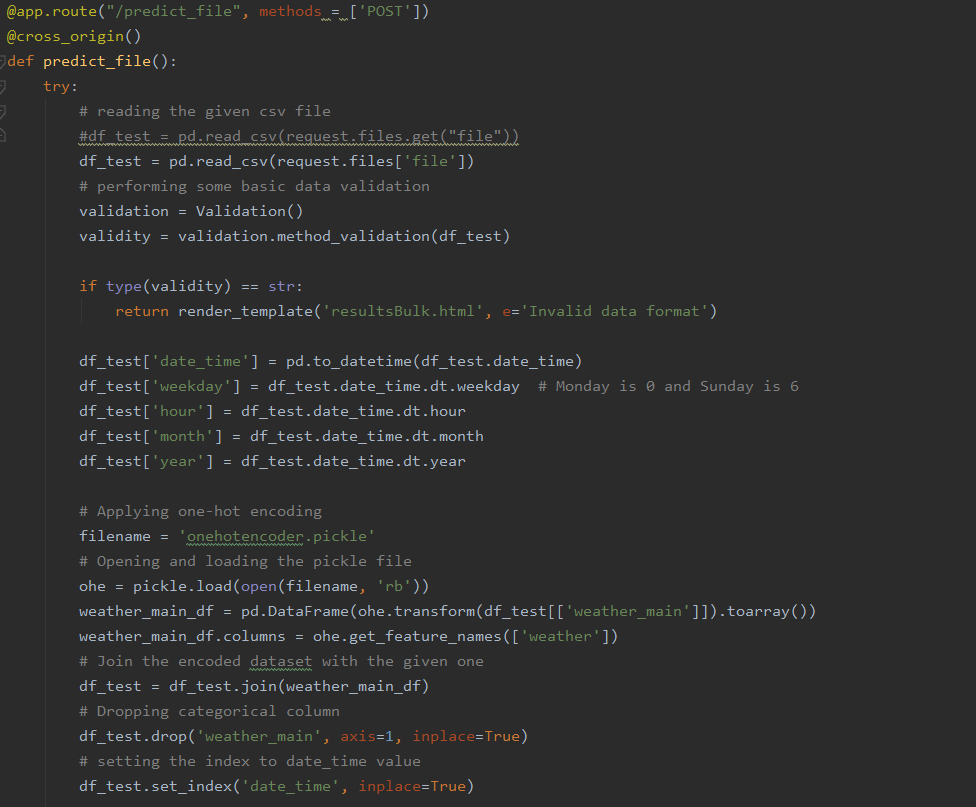


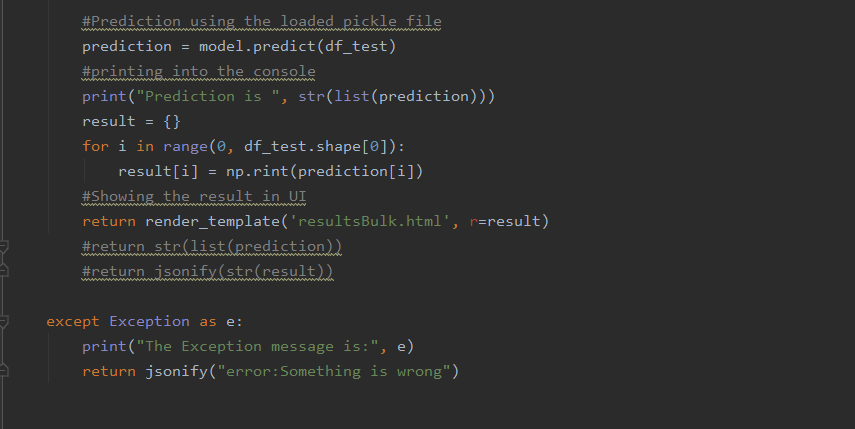


‘/predict\_file’ -> used for bulk prediction

Steps involved –

1. read the uploaded csv file
2. perform validation and check if the required X columns or feature columns are present
3. if validation passes then go to step 4 else If fails then display Invalid data format.
4. Apply ‘one-hot coding’ on ‘weather\_main’ column
5. Perform prediction using random-forest pickle file
6. Return back the result to the web UI



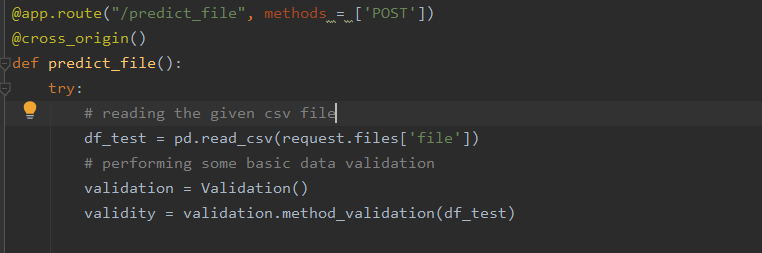


Initially we tested the performance of predictor app in our local.

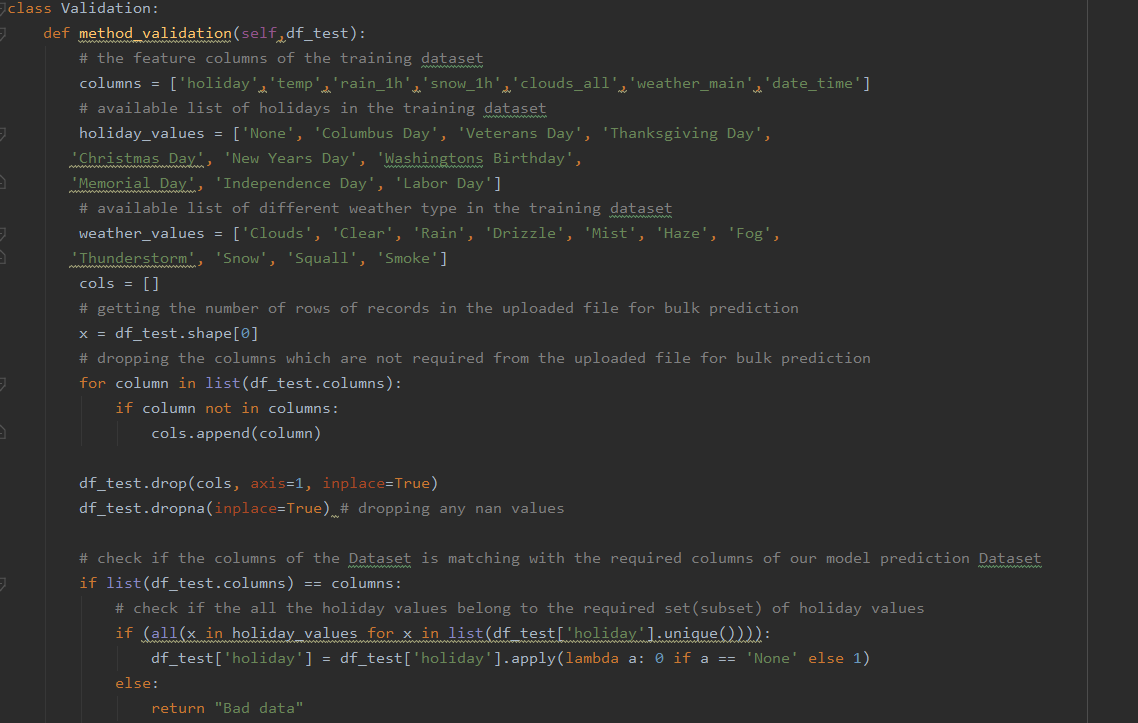
# Methods Analysis

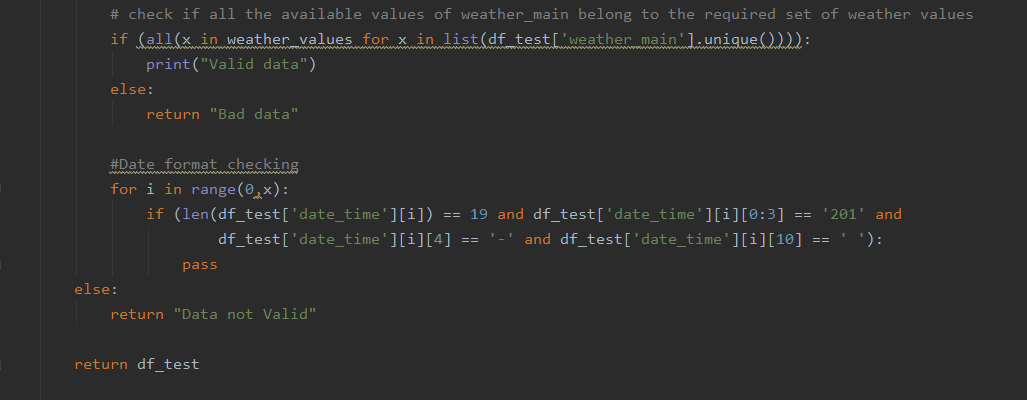
Our application has one method that is **method\_validation** which is present in the validation class. This method is called from the /predict\_file route to validate the uploaded file for bulk prediction.

Snapshot of the calling function –



After reading the csv file, it is passed to the ‘method\_validation()’ method, where different validations are being performed. Now we will see the validation method





The flow of the method is first it will drop all the unnecessary columns from the uploaded CSV file. Then after dropping the columns if the columns of the resultant dataset matches with feature columns of train dataset then we will go to second round of validation or else return ‘Data not valid’.

In the second round of validation we will validate mainly the ‘holiday’, ‘weather’ and ‘date\_time’ columns.

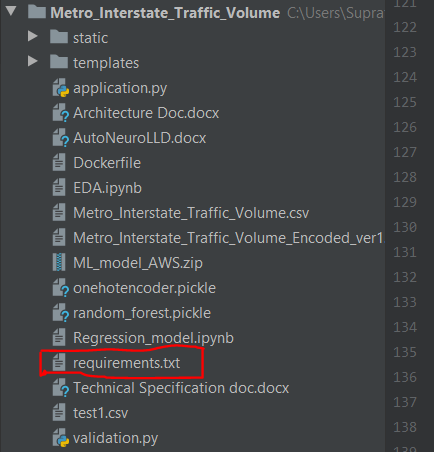
If all the validations are passed, then the dataset is returned back to calling function for bulk prediction.

1. Cloud deployment and Dockerization:

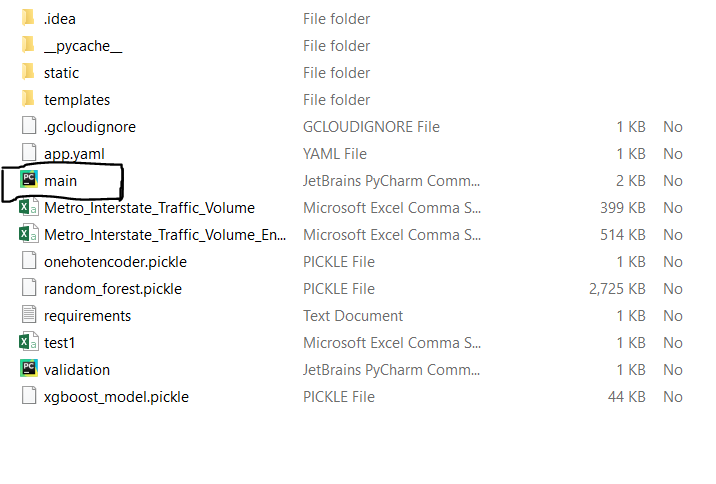
I have deployed our project in GCP and the steps involved for deployment are –

* Go to <https://cloud.google.com/> and create an account if already haven’t created one. Then go to the console of your account.
* Go to *IAM and admin* and click *manage resources*
* Click CREATE PROJECT to create a new project for deployment.
* Once the project gets created, select App Engine and select Dashboard.
* Go to <https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe> to download the google cloud SDK to your machine.
* Click *Start Tutorial* on the screen and select Python app and click start.
* Create a file ‘app.yaml’ and put ‘runtime: python37’ in that file.
* Create a ‘requirements.txt’ file by opening the command prompt/anaconda prompt, navigate to the project folder and enter the command ‘pip freeze > requirements.txt’.

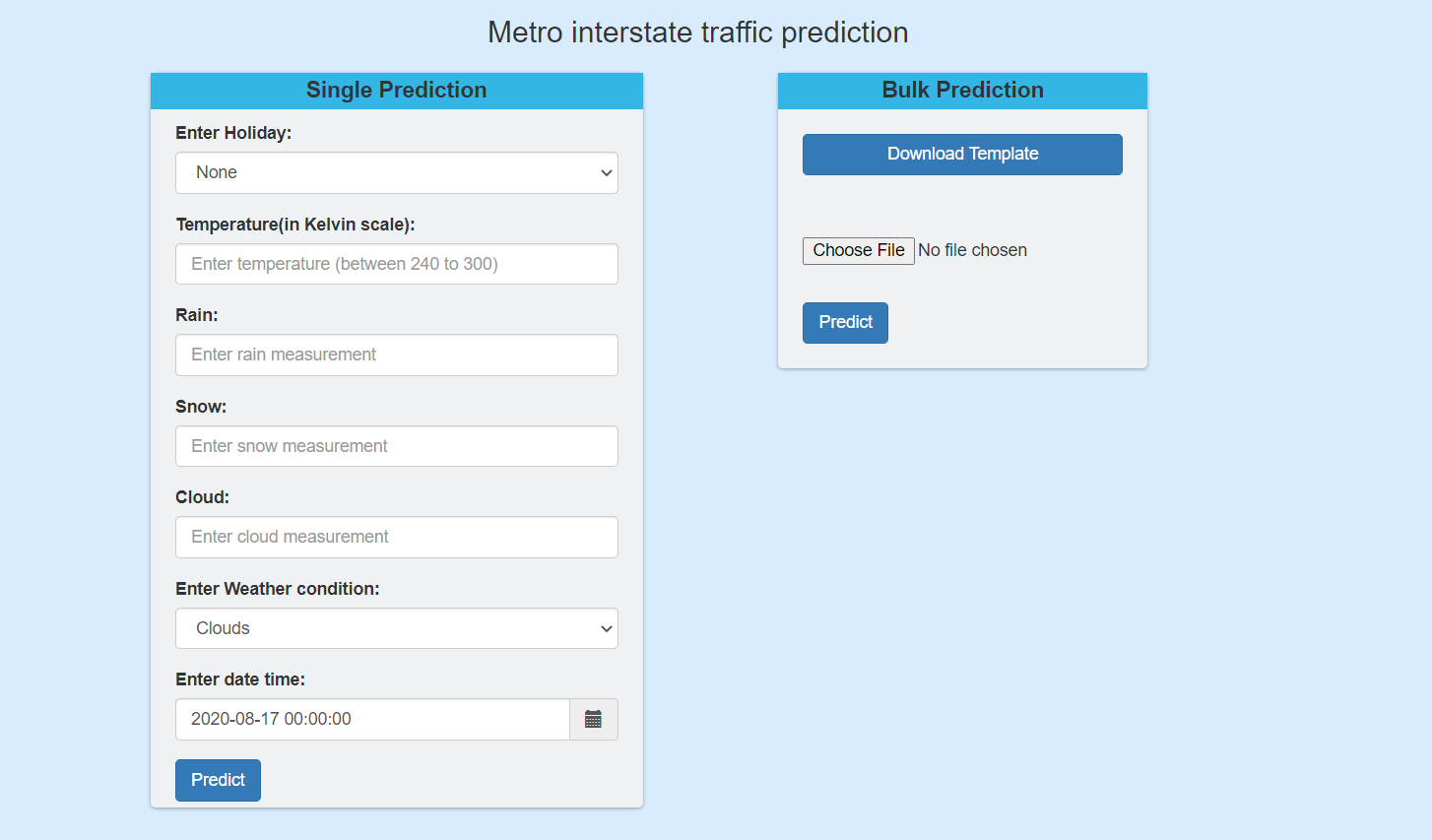
It is recommended to use separate environments for different projects.



* Your python application file should be called ‘main.py’. It is a GCP specific requirement.



* Open command prompt window, navigate to the project folder and enter the command gcloud init to initialise the gcloud context.
* It asks you to select from the list of available projects.
* Once the project name is selected, enter the command gcloud app deploy app.yaml --project <project name>.
* After executing the above command, GCP will ask you to enter the region for your application. Choose the appropriate one.
* GCP will ask for the services to be deployed. Enter ‘y’ to deploy the services.
* And then it will give you the link for your app, and the deployed app looks like:



Link of the app: <https://metrointerstatetrafficpredict.df.r.appspot.com/>

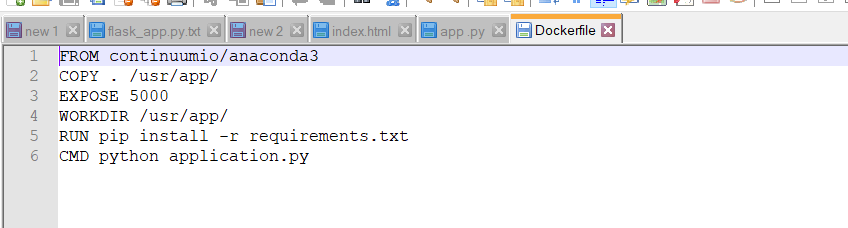
**Dockerization**:

Steps to follow to dockerize your application –

1. Write the docker file (it has to be created in the same location as your working project folder)

in order create the docker file follow the below steps-

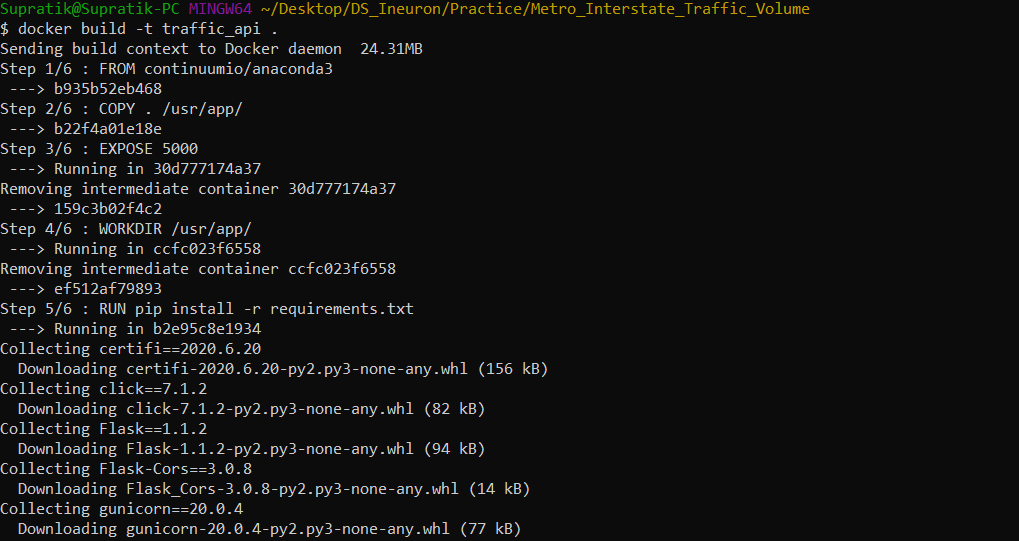
a) Open Docker Quick Terminal -> Go to the project directory

b) write the command touch Dockerfile (this will create your docker file, then open it using Notepad++) 

2) Building the docker image

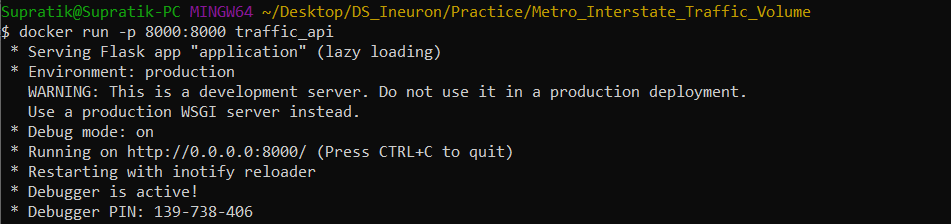
Go to Docker Quick Terminal -> type pwd -> then change the CD to your project directory(cd path)

1. Building the docker image using the command docker build -t traffic\_api .



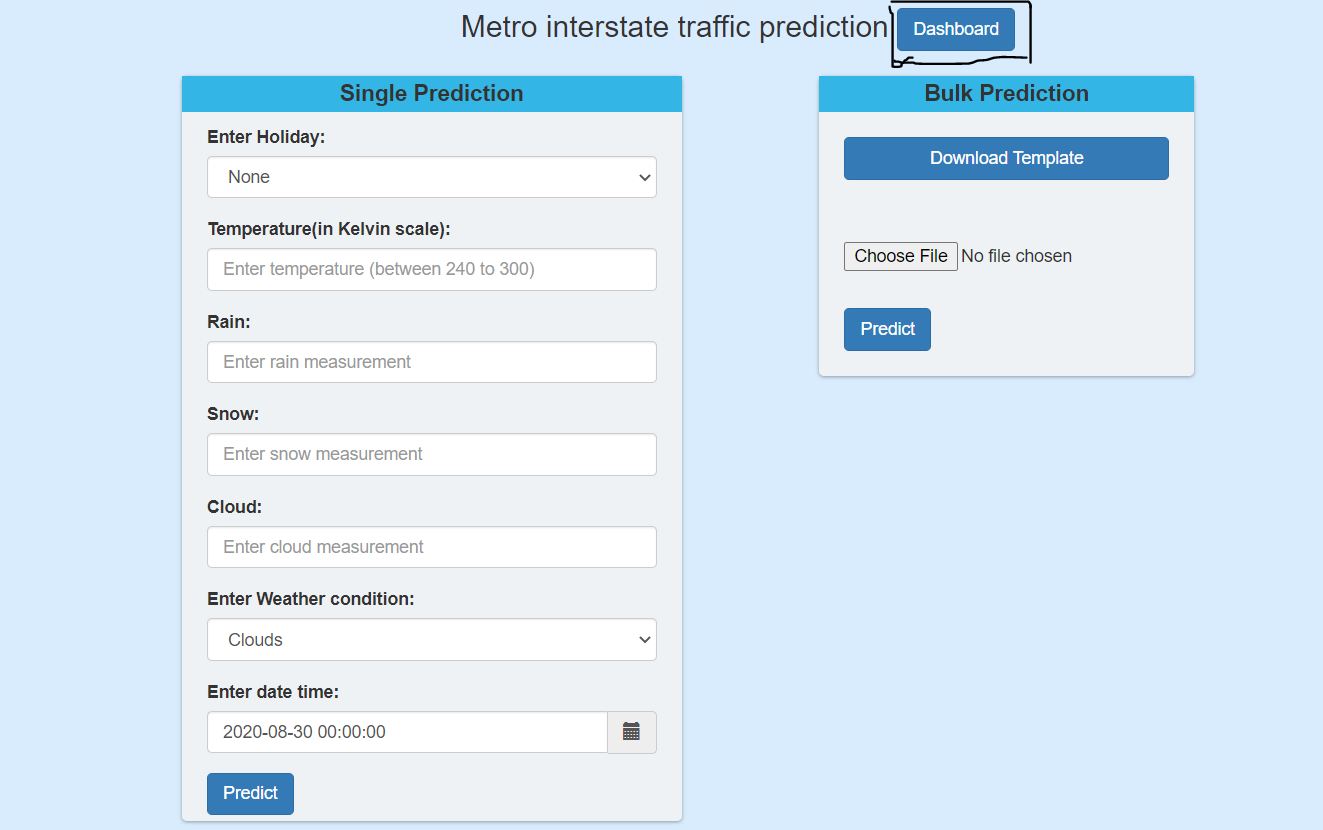
1. Running your app.py via docker –

docker run -p 8000:8000 traffic\_api



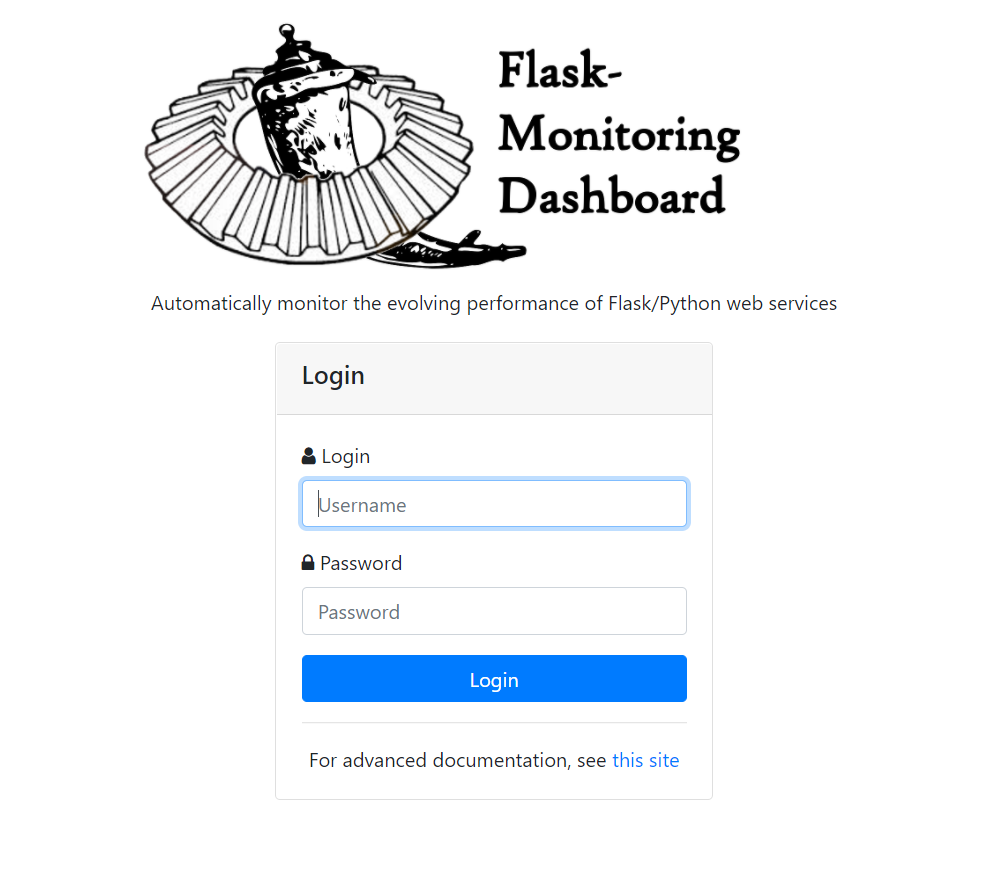
# Application Monitoring(Dashboard)

To monitor our application, we have prepared a dashboard also, which can be accessed by clicking the ‘Dashboard’ hyperlink on the application home page (see the below screenshot)

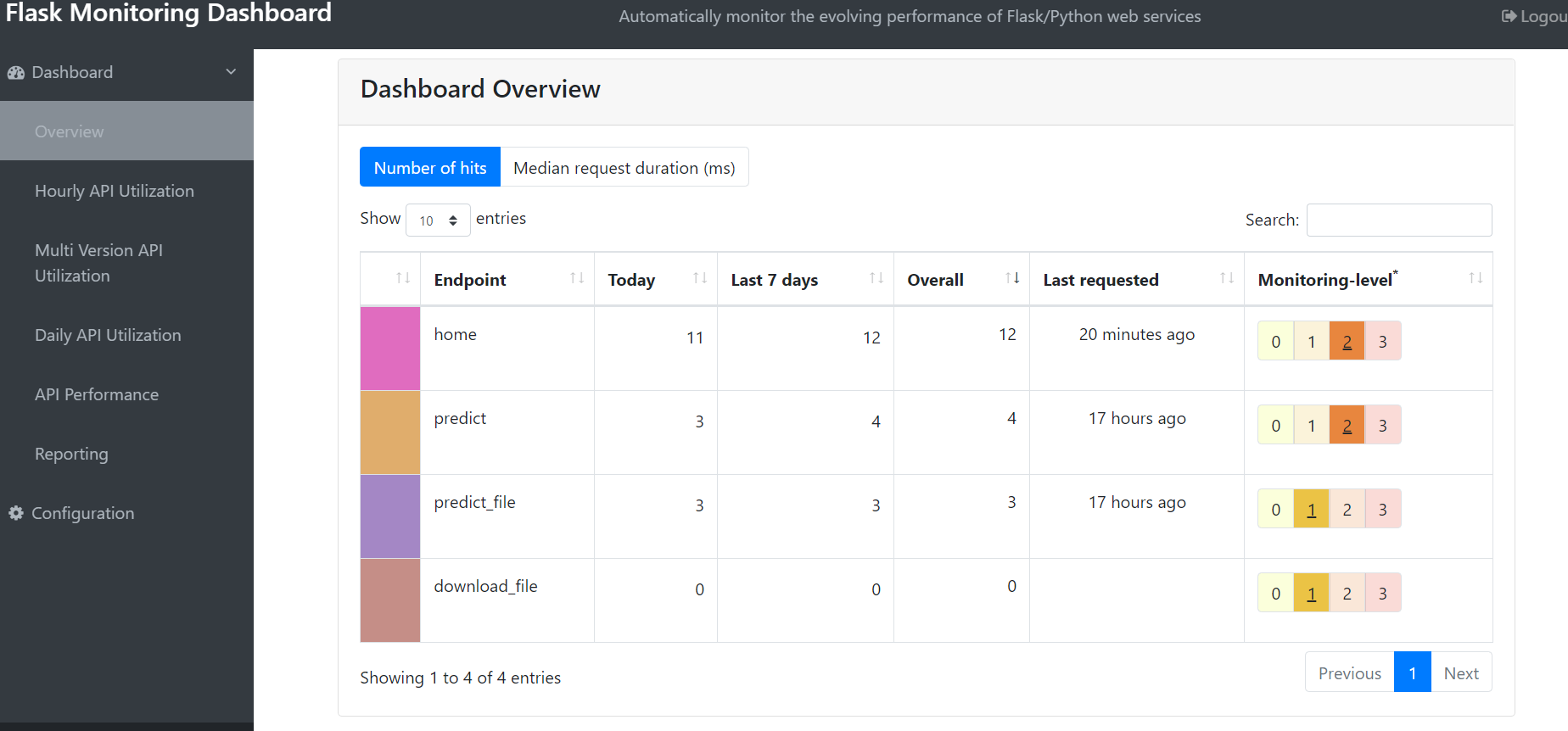


(the button highlighted in black rectangular box)

By clicking on the Dashboard button it will take you to the next page, where you will be prompted to enter username & password (please remember both the username & password is admin)

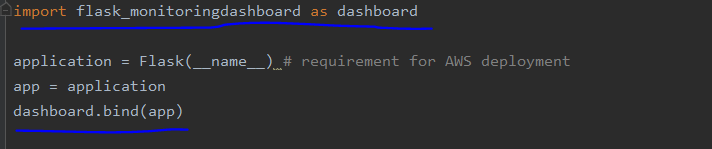


Once you login with the username & password, you will find the dashboard view of the application of how many hits,miss etc (check below screenshot for the dashboard overview)

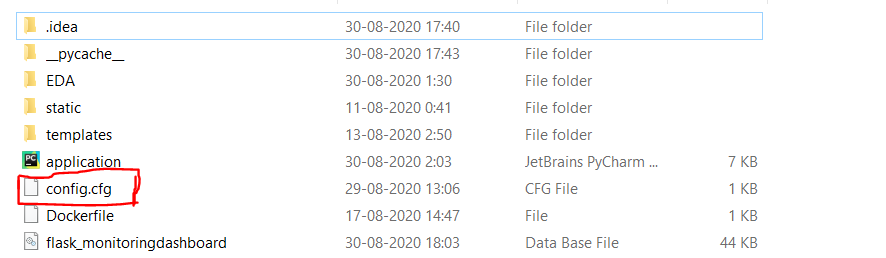


In order to build this dashboard, the requirements are listed below –

1. We need to install flask\_monitoringdashboard package for this dashboard (pip install flask\_monitoringdashboard)
2. Once you’ve installed you need to add 2 lines of code in you app.py file



1. Apart from this we need a config file which will contain the below details and this config file should be in the same project directory.





1. Hardware Requirements

Requirements for model training

The minimum configuration should be:

8 GB RAM

2 GB of Hard Disk Space

Intel Core i5 Processor

Requirements for model testing

The minimum configuration should be:

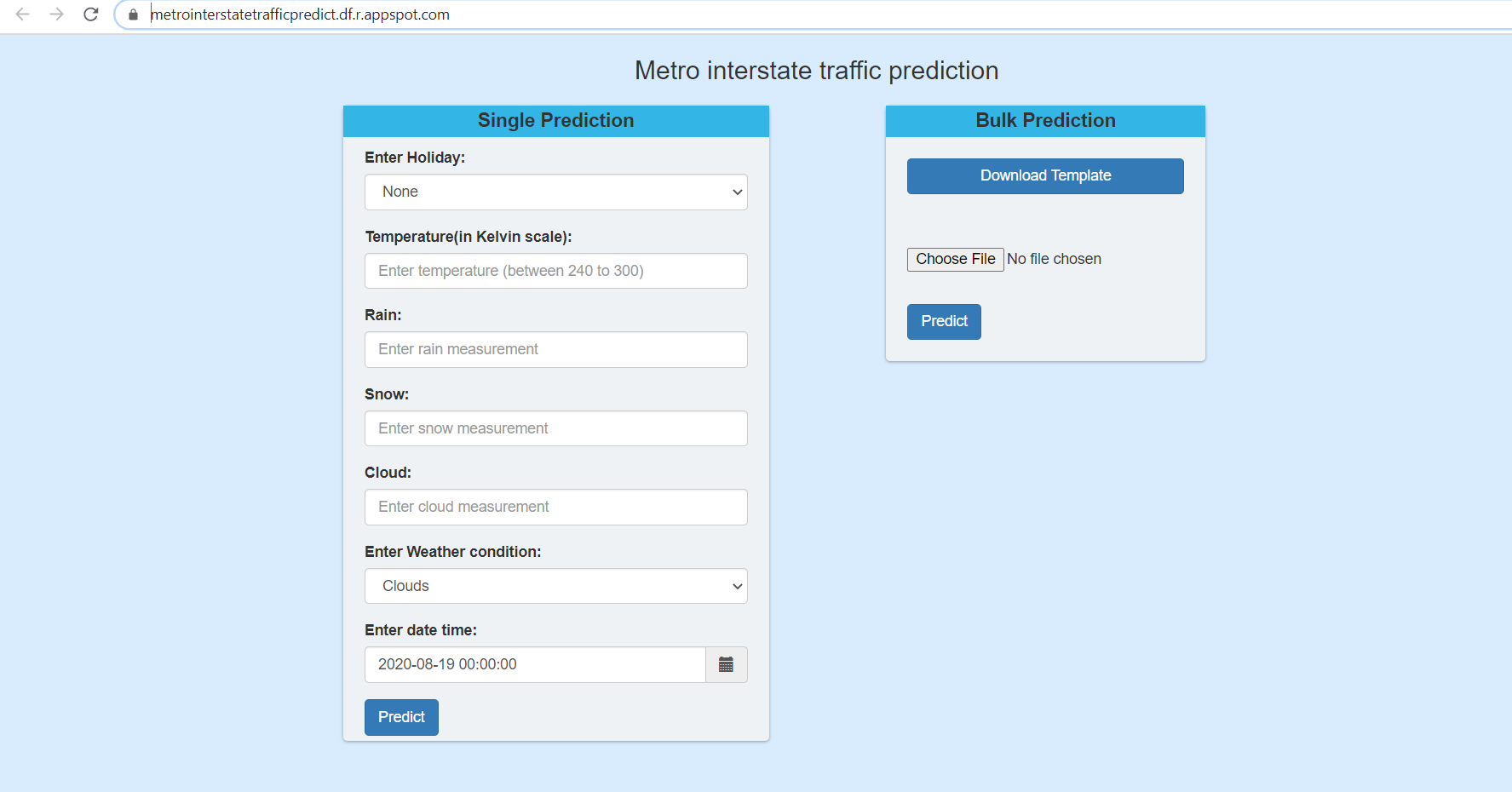
4 GB RAM

2 GB of Hard Disk Space

Intel Core i5 Processor

**Demonstration of the app**:

Open the app URL(<https://metrointerstatetrafficpredict.df.r.appspot.com/>) in any browser.



For single prediction –

Enter the values of the variables

[Possible example of values for the variables are given below –

‘Holiday’ -> ('None'/'Columbus Day'/'Veterans Day'/ 'Thanksgiving Day'/

'Christmas Day'/ 'New Years Day'/ 'Washingtons Birthday'/

'Memorial Day'/'Independence Day'/ 'State Fair'/ 'Labor Day'/

'Martin Luther King Jr Day')

‘Temperature -> any Decimal value between 230 to 310.

‘Rain’ -> any decimal (floating point number) between 0 to 100.

‘Snow’ -> any decimal value between 0 to 20.

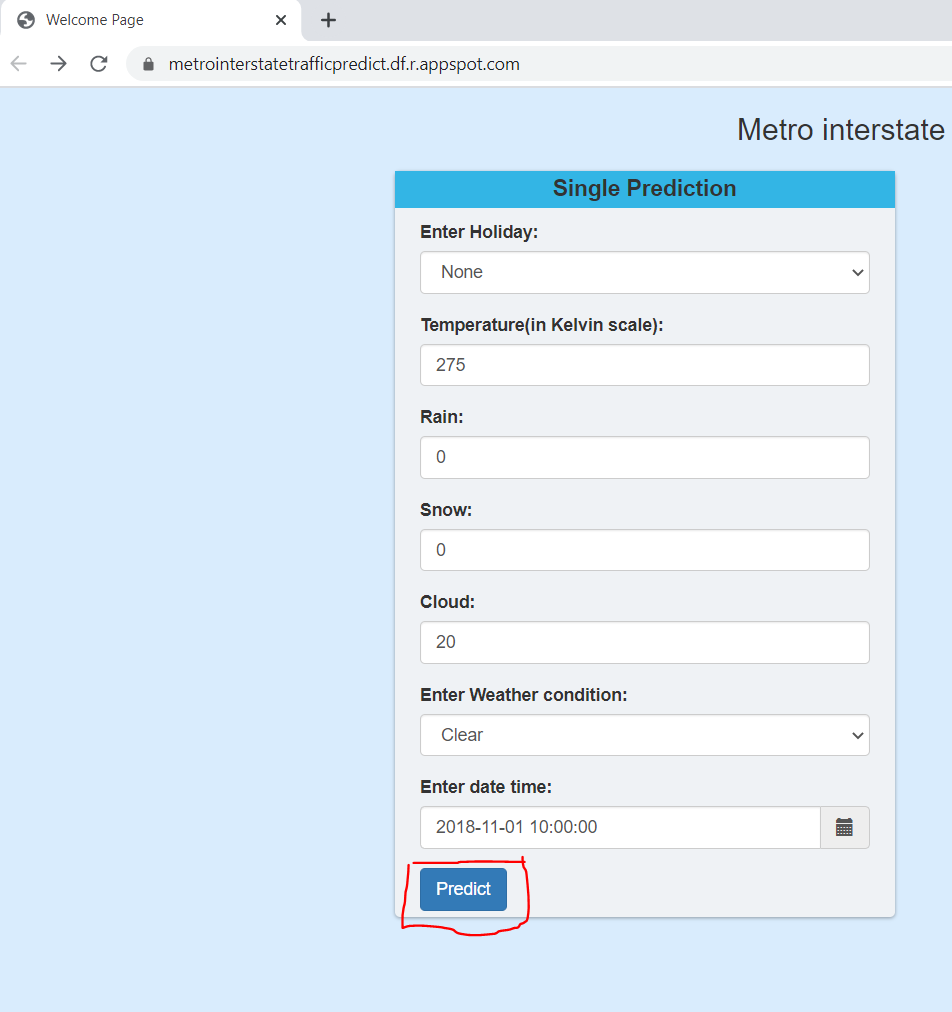
Cloud -> any decimal value between 0 to 100.

‘Weather condition’ -> selecting any value from the available drop down list. Possible values available are –

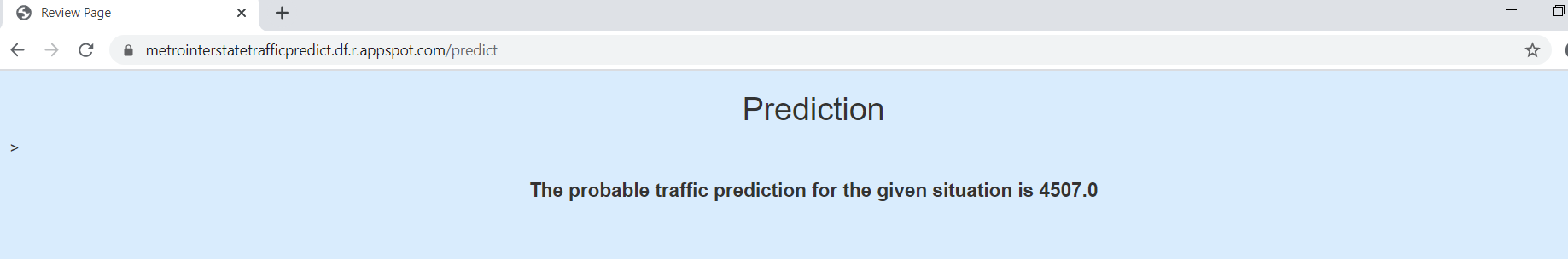
('Clouds'/ 'Clear'/ 'Rain'/ 'Drizzle'/ 'Mist'/ 'Haze'/ 'Fog'/ 'Thunderstorm'/ 'Snow'/ 'Squall'/ 'Smoke')

‘date\_time -> enter any data\_time value from the datatime picker.

]

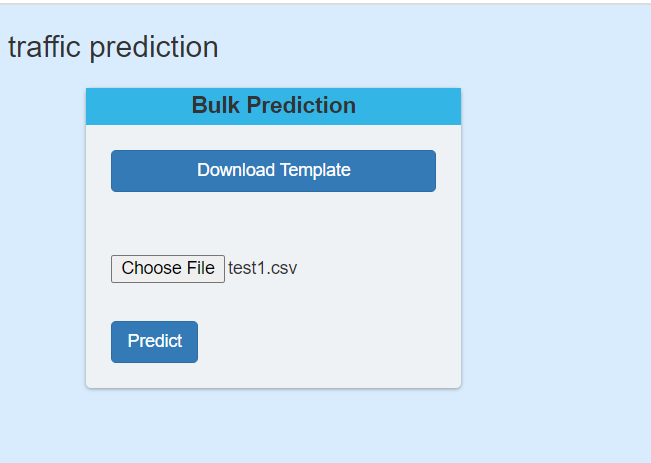


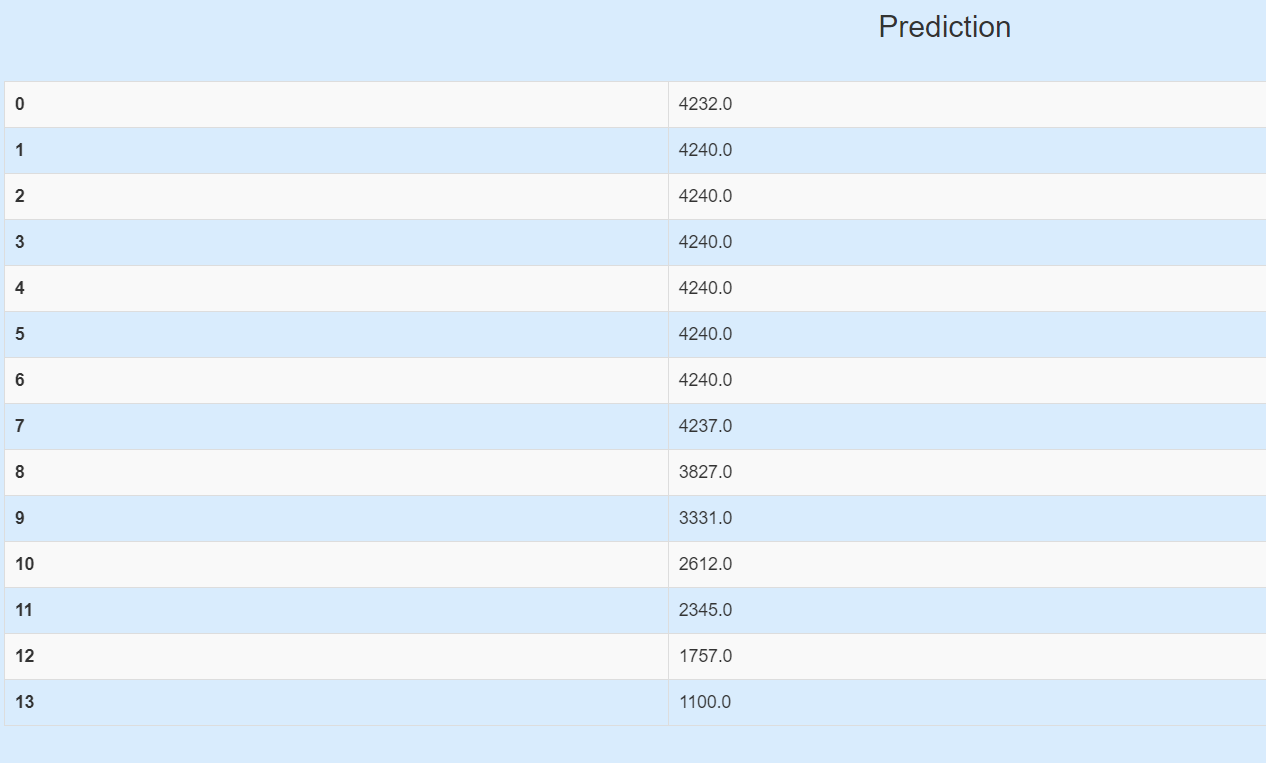
Press the ‘predict’ button, and wait for few seconds till the results page is uploaded.



For bulk prediction –

Upload any .csv file through the ‘choose file’ button. If it matches with ‘X’ columns of the trained model, then predictions will be displayed in a tabular manner like below –





1. Model Retraining approach(strategy):

1. For Model retraining we need to save the predictions which are predicted by the model. (for this purpose we are saving the predictions using mongo db)

2. Predictions that are saved in the model deployment area will be monitored.

3.  We need to collect new data at a frequency of around 1 month/3 month from different data sources. A scheduler can be created for the same.

4. We will clean our predictions label then and along with this, the new data collected we will check for error in our model or significant reduction in accuracy.

5. If we find significant change in accuracy or error in prediction, we will use this new data and the predictions from the existing model for re-training and validation of our model.

6. With this combination of new data and the data from saved predictions we have to perform Exploratory Data Analysis and then again retrain our model. (in this scenario we will again follow the model building architecture)

# Learnings and Obstacles

* The first problem we faced was to understand whether it was a Time-series problem or a Regression problem. Then based on our EDA, we understood it was a Regression problem.
* Understand how to group different values of one column in order to do feature engineering.
* Faced challenges while performing Hyper-parameter tuning on our models, in order to increase model accuracy.
* Then while building our application, bulk prediction became an issue since how to make the user understand the correct format of the columns required for prediction. Henceforth added the template button.
* Creation of application dashboard was a complete new challenge, learned a new package available in python flask called flask\_monitoringdashboard. Really very interesting package with just 2 lines of code you can create your own dashboard to track all the flask API’s in your application.
* Faced problems during AWS cloud deployment, so finally deployed our application to GCP.
* Worked on docker for the first time, faced lot of issues initially but finally was able to dockerize our application.