A PROJECT REPORT ON

TRIP TIME PREDICTION

Submitted to

International Institute of Information Technology, Bangalore

Submitted by

Adarsh Das	MT2018501
Anjali Joshi	MT2018504

Under the guidance of Prof. G Srinivasaraghavan

INDEX

Problem Statement

In this project we are asked to predict the trip duration of bmtc buses from source to destination .

Dataset Used

Raw Dataset : It consists of six columns of busid, latitude , longitude, direction, speed and time-stamp.

- Busid: It consists of bus id of around 6000 buses .
- Latitude: It is the latitude of the location at which bus was present at a given timestamp.
- Longitude: It is the longitude of the location at which bus was present at a given timestamp.
- Direction: it is the direction in which bus is moving.
- Speed: speed of bus at that timestamp.
- Timestamp: It consists of time and date of a bus at a particular start latitude and longitude.

NOTE: Raw dataset is 14GB in size and doesn't consist of column names . Hence these names are provided according to the understanding of data. Columns like direction and speed consist of almost all null values and hence wasn’t considered for prediction.

Train Dataset: It consists of 7 columns of s_lat, s_long, s_date, s_clock, e_lat, e_long, e_clock, e_date and time_diff.

- s_lat: It consists of latitude of the start location for a bus.
- s_long: It consists of longitude of the start location for a bus.
- s_date: It is the date at which a bus started its journey.
- s_clock: It gives the time at which journey of bus was started.
- e_lat: It consists of latitude of the end location for a bus.
- e_long: It consists of longitude of the end location for a bus.
- e_date: It is the date at which a bus ended its journey.
- e_clock: It gives the time at which journey of bus was ended.
- time_diff: It is the time difference between the start and end location of bus for a particular set of start and end latitude and longitude combination.

NOTE: Above dataset's each row belongs to a bus journey for a particular day (not taken for two different days) that is s_date and e_date are same valued columns. The train dataset consist of only 400 buses.

Test Dataset: There are 100 locations for each Id in the test set including starting and ending point of the journey for which we need to predict the duration of journey. Each location is specified by latitude and longitude separated by ':'.

- Busid: It consists of bus id of around 108 buses .
- Timestamp: It consists of time and date of a bus at a particular start latitude and longitude.
- LATLONG1: It refers to starting point.
- LATLONG100: It refers to ending point of the journey for each Id in the test set.
- LATLONG2 - LATLONG99: These are locations in the path of journey.

Approach

As we have to predict trip time for buses starting from a particular latitude and longitude.

So we are going to use regression model for this purpose.

We will first convert the raw dataset provided to us into required train dataset removing irrelevant columns.

Then we will do preprocessing and model training.

Analysis Of Dataset

- **Chunking:** As file of 14GB size can't be opened in the text editor (too large) , as well as computation will take very long time to go through it. So we divided it into small chunks of size 610MB and around 22 chunks were created(where each chunk consist of 10000000 rows).As column headers are not present so appended the 'index', 'bus_id','lat','long','p1','p2','timestamp' to top of each csv.

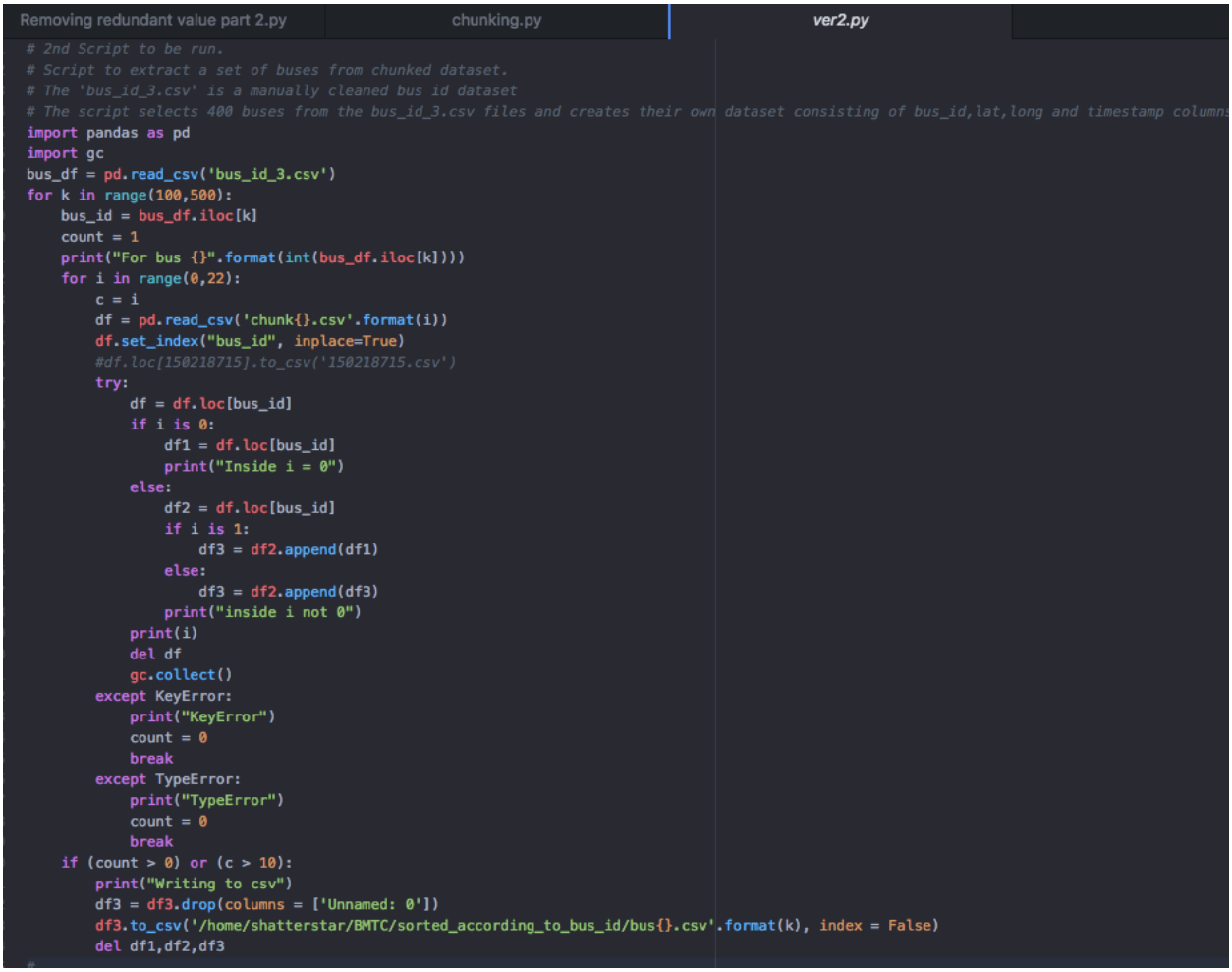
  NOTE: Script present in handling chunking.py)

Removing redundant value part 2.py | chunking.py

```
# 1st Script to be run to separate w1.csv file.
# This code does the following things
# Reads chunks of '10000000' rows from the w1 dataset given to us.
#We manage to create 22 chunks #each of size '610 MB' from the 14GB data given to us
# Append the 'index','bus_id','lat','long','p1','p2','timestamp' to top of each csv
# Create a set of unique bus ID's in all the chunks
# Store it in 'bus_id.csv' file. But it stores the set as a string and not a pandas series.
#Manual cleaning needs to be done

import pandas as pd
import fileinput
import pandas as pd
import csv
B = {}
for i,chunk in enumerate(pd.read_csv('w1.csv',chunksize = 10000000)):
    chunk.to_csv('chunk{}.csv'.format(i))
    for line in fileinput.input(files=['chunk{}.csv'.format(i)],inplace=True):
        if fileinput.isfirstline():
            print('index,bus_id,lat,long,p1,p2,timestamp')
        print(line)
    df = pd.read_csv('chunk{}.csv'.format(i))
    A = set(df['bus_id'].unique())
    B = set(B).union(set(A))
    print(len(B))
    df2 = df.drop(columns=['index','p1','p2'],axis=1)
    df2.to_csv('chunk{}.csv'.format(i))
with open('bus_id.csv','w') as file:
    file.write((str(B)))

```

- **Same bus data in same csv**: As buses are arranged in random order in terms of there bus id. So, we made a unique bus _id.csv by running script through all csv files. then using these unique bus id, filtered the dataset belonging to particular bus and converted it into that busid csv files for around 400 buses consisting of bus_id,lat,long and timestamp columns (each bus_id.csv consist of data for a particular busid only). Then dropped the busid column as now the csv files are independent of bus number.

  NOTE: Script present in handling ver2.py)



- **Rearranging Dataset: As the raw dataset's each row consist of just the latitude and longitude of location, bus was at a particular timestamp and we need to find the trip time for a bus from a particular source to a destination. So created single row consisting of both start and end latitude and longitude by combining two different rows for a particular busid separated by random interval . Final dataset columns are ['s_lat', 's_long',' s_time', 'e_lat','e_long','e_time']. NOTE: Script present in handling final.py)**

Removing redundant value part 2.py | *final.py*

```python
# 3rd Script to be run.
# We now have a dataset of 400 buses.
# We will create a new dataset. Each row of the dataset will correspond to a single bus id start and stop location and timestamp.
# Final dataset columns are ['s_lat','s_long','s_time','e_lat','e_long','e_time'].

import pandas as pd
import random
import gc
df_final = pd.DataFrame(columns = ['s_lat','s_long','s_time','e_lat','e_long','e_time'])
for i in range(100,500):
    try:
        k = 0
        df = pd.read_csv('bus{}.csv'.format(i))
        print('for bus{}'.format(i))
        print(i)
        df = df.sort_values(by = 'timestamp')
        df_buffer = pd.DataFrame(index = range(0,100000),columns = ['s_lat','s_long','s_time','e_lat','e_long','e_time'])
        try:
            for l in range(0,len(df['timestamp'].unique()),random.randint(7,10)):
            # Start columns
                df_buffer['s_lat'].iloc[k] = df['lat'].iloc[l]
                df_buffer['s_long'].iloc[k] = df['long'].iloc[l]
                df_buffer['s_time'].iloc[k] = df['timestamp'].iloc[l]
            # End columns
                df_buffer['e_lat'].iloc[k] = df['lat'].iloc[l+5]
                df_buffer['e_long'].iloc[k] = df['long'].iloc[l+5]
                df_buffer['e_time'].iloc[k] = df['timestamp'].iloc[l+5]
                k = k+1
                if k > 5000:
                    break
            df_buffer = df_buffer.dropna()
            df_final = df_final.append(df_buffer)
            del df
            gc.collect()
        except IndexError:
            print('IndexError')
    except FileNotFoundError:
        i = i+1
        print('Inside exception')

df_final.to_csv('final_bus_data.csv', index = False)
```

- **Date and Time extraction:** As time stamp consists of date and time separated by space . So we separated them into two columns, one consisting of date and other time. Date column is label encoded to get which day of week it belongs to and also to know the unique numbers of date present(it helped to know that the dataset is for 6 days). Time column is converted to seconds value. So we got s_date, s_clock, e_date and e_clock columns(which consist of start date, start time in seconds, end date and end time in seconds respectively).

NOTE: Script present in handling date and time.py)

Removing redundant value part 2.py | *handling date and time.py*

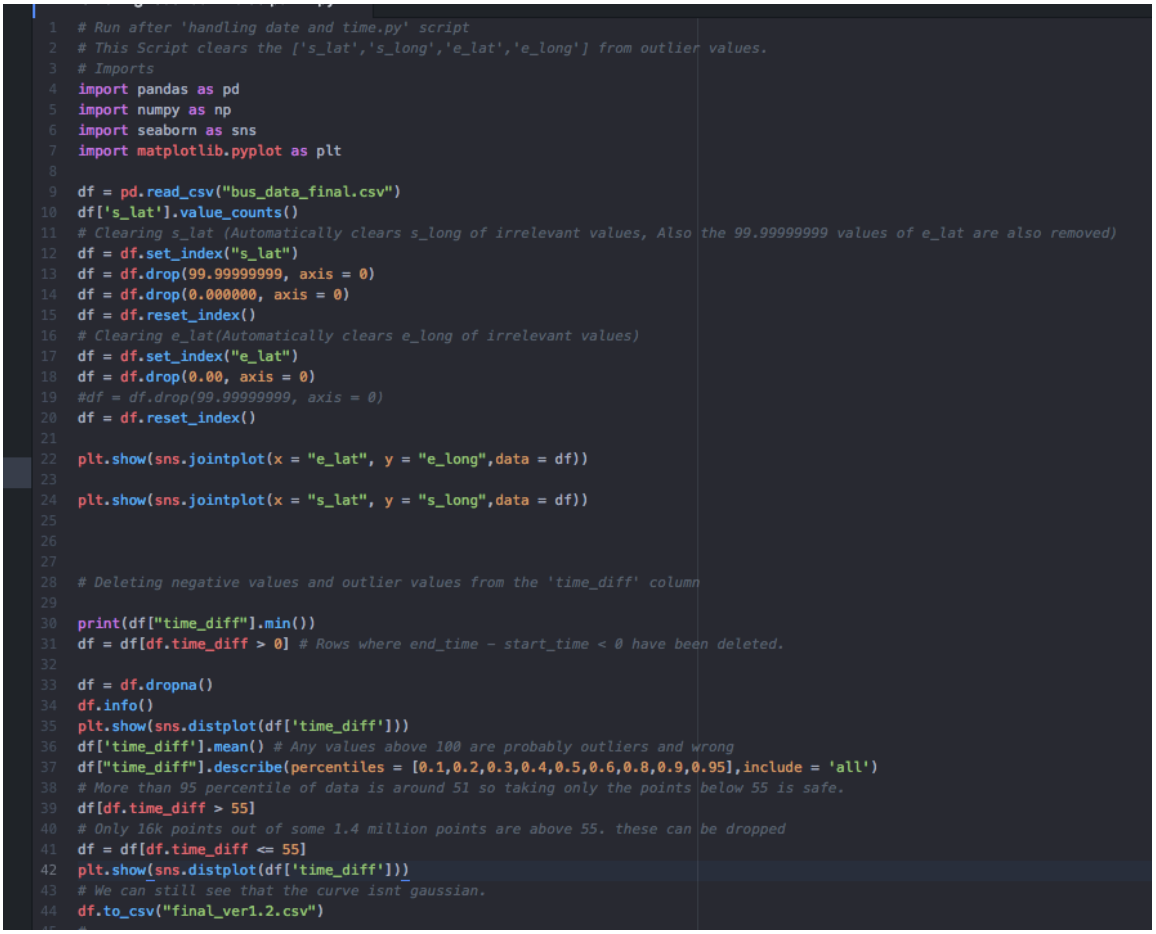
```python
# This runs after final.py script
# This script is run to handle the timestamp data columns ['s_time','e_time']
# We label encode the date parameter and convert the time parameter into seconds. The seconds value is counted from 00:00:00 time period.

import pandas as pd

df = pd.read_csv('final_bus_data.csv')

# Deal with dates
df['s_date'] = pd.to_datetime(df['s_time']).dt.date
df['e_date'] = pd.to_datetime(df['e_time']).dt.date

# Deal with time.
df['s_time'] = df['s_time'].apply(pd.Timestamp)
df['e_time'] = df['e_time'].apply(pd.Timestamp)

# Create s_clock and e_clock parameters which are in the string form HH:MM:SS
df['s_clock'] = df['s_time'].dt.strftime('%H:%M:%S')
df['e_clock'] = df['e_time'].dt.strftime('%H:%M:%S')

# Split it into hours:minutes:seconds columns
df[['H1','M1','S1']] = df['s_clock'].str.split(':', expand = True)
df[['H2','M2','S2']] = df['e_clock'].str.split(':', expand = True)


df.head()

# Convert the data from string to numeric
df['H1'] = pd.to_numeric(df['H1'], errors='coerce')
df['M1'] = pd.to_numeric(df['M1'], errors='coerce')
df['S1'] = pd.to_numeric(df['S1'], errors='coerce')
df['H2'] = pd.to_numeric(df['H2'], errors='coerce')
df['M2'] = pd.to_numeric(df['M2'], errors='coerce')
df['S2'] = pd.to_numeric(df['S2'], errors='coerce')

# Convert Hours and Minutes into seconds
df['H1'] = 3600*df['H1']
df['M1'] = 60*df['M1']
df['M2'] = 60*df['M2']
df['H2'] = 3600*df['H2']

df.head(100)

# Convert s_clock and e_clock to seconds
df['s_clock'] = df['H1'] + df['M1'] + df['S1']
df['e_clock'] = df['H2'] + df['M2'] + df['S2']

df.head(100)

# Drop un necessary columns
df = df.drop(columns = ["s_time","e_time","H1","H2","M1","M2","S1","S2"])
df.head()

# Use Label Encoding on the dates
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df['s_date'] = le.fit_transform(df['s_date'])
df['e_date'] = le.fit_transform(df['e_date'])

# Define The 'time_diff' column
df["time_diff"] = df["e_clock"] - df["s_clock"]

# Write to a new csv
df.to_csv("bus_data_final.csv",index = False)

#
```

- **Removing Outliers :** As s_lat and e_lat consist of irrelevant values like 0.000000 and 99.99999999 which will influence the conclusion that can be derived from those columns. So dropped these values from columns of s_lat and e_lat which in turned dropped the irrelevant values from s_long and e_long columns.

  NOTE: Script present in handling date and Removing redundant value part2.py)

- **Removing negative values and Outliers :** In the time_diff column there were some negative values which were dropped. In time_diff column more than 95 percentile of data is around 51sec so taking only the points below 55 sec is safe(Only 16k points out of some 1.4 million points are above 55, so these can be dropped).

  NOTE: Script present in handling date and Removing redundant value part2.py)



```
# Run after 'handling date and time.py' script
# This Script clears the ['s_lat','s_long','e_lat','e_long'] from outlier values.
# Imports
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("bus_data_final.csv")
df['s_lat'].value_counts()
# Clearing s_lat (Automatically clears s_long of irrelevant values, Also the 99.99999999 values of e_lat are also removed)
df = df.set_index("s_lat")
df = df.drop(99.99999999, axis = 0)
df = df.drop(0.000000, axis = 0)
df = df.reset_index()
# Clearing e_lat(Automatically clears e_long of irrelevant values)
df = df.set_index("e_lat")
df = df.drop(0.00, axis = 0)
#df = df.drop(99.99999999, axis = 0)
df = df.reset_index()

plt.show(sns.jointplot(x = "e_lat", y = "e_long",data = df))

plt.show(sns.jointplot(x = "s_lat", y = "s_long",data = df))



# Deleting negative values and outlier values from the 'time_diff' column

print(df["time_diff"].min())
df = df[df.time_diff > 0] # Rows where end_time - start_time < 0 have been deleted.

df = df.dropna()
df.info()
plt.show(sns.distplot(df['time_diff']))
df['time_diff'].mean() # Any values above 100 are probably outliers and wrong
df["time_diff"].describe(percentiles = [0.1,0.2,0.3,0.4,0.5,0.6,0.8,0.9,0.95],include = 'all')
# More than 95 percentile of data is around 51 so taking only the points below 55 is safe.
df[df.time_diff > 55]
# Only 16k points out of some 1.4 million points are above 55. these can be dropped
df = df[df.time_diff <= 55]
plt.show(sns.distplot(df['time_diff']))
# We can still see that the curve isnt gaussian.
df.to_csv("final_ver1.2.csv")
```

- **Correlation**: Correlation is the statistical measure that describes association between different random variable. Bellow table gives the correlation between each and every column.

	s_lat	s_long	e_lat	e_long	s_date	s_clock	e_date	e_clock	time_diff
s_lat	1	-8.25837e-05	0.998065	0.00189049	-0.0103538	0.00268842	-0.0103538	0.00267385	-0.0519341
s_long	-8.25837e-05	1	0.00195519	0.996106	0.0116307	0.0151896	0.0116307	0.0151887	-0.00259055
e_lat	0.998065	0.00195519	1	-0.000161252	-0.0103396	0.00279452	-0.0103396	0.00277998	-0.051828
e_long	0.00189049	0.996106	-0.000161252	1	0.0118728	0.0150059	0.0118728	0.0150052	-0.0021921
s_date	-0.0103538	0.0116307	-0.0103396	0.0118728	1	-0.0115896	1	-0.0115858	0.0130104
s_clock	0.00268842	0.0151896	0.00279452	0.0150059	-0.0115896	1	-0.0115896	1	0.0430654
e_date	-0.0103538	0.0116307	-0.0103396	0.0118728	1	-0.0115896	1	-0.0115858	0.0130104
e_clock	0.00267385	0.0151887	0.00277998	0.0150052	-0.0115858	1	-0.0115858	1	0.0433448
time_diff	-0.0519341	-0.00259055	-0.051828	-0.0021921	0.0130104	0.0430654	0.0130104	0.0433448	1

It is showing very low correlation between (time_diff , e_long) and in between (time_diff, s_long). Correlation is not the right criteria whether variable have relationship or not because it tells only about the linear relationship but gives very low correlation values for non-linear relationships.

- **Drop date columns:** As the new dataponts are made for same date so we dropped the s_date, and e_date columns for further simplification.

Visualization

- The most convenient way to take a quick look at time_diff column in seaborn is the distplot() function. By default, this will draw a histogram and fit a kernel density estimate (KDE).

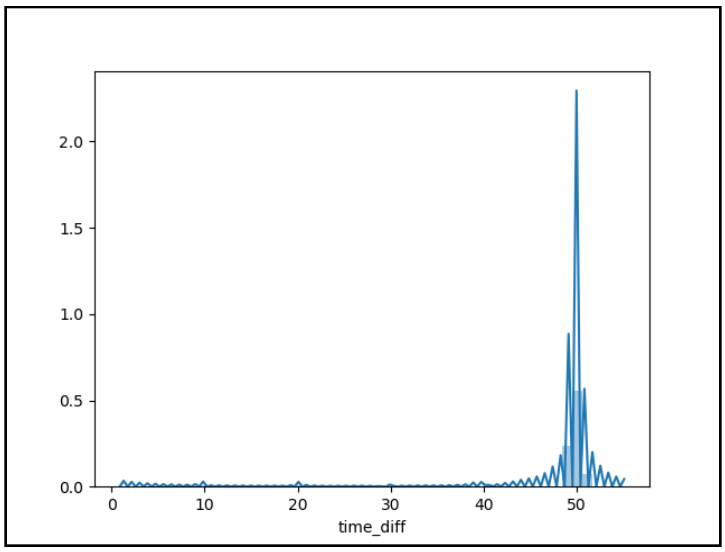


Fig1: time_diff distribution plot

As you can see that majority of datapoint lie between 0 to 55. and only 16k out 1.4 million are above that . So these data points were dropped . And again for a quick look on the univariate distribution we used distplot() for time_diff column shown bellow.

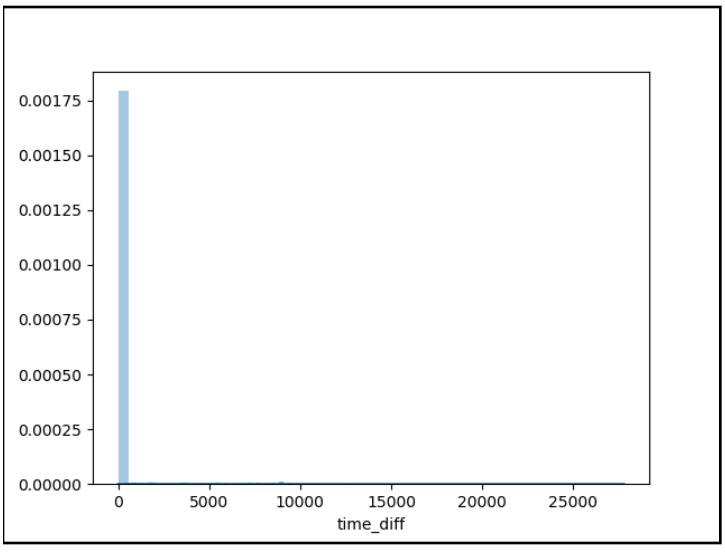


Fig2: time_dff distribution plot after removing time_dff value above 55

- We need to see relationship between (s_lat, s_long) and between (e_lat, e_long) . So pairs plot allows us to see both distribution of single variables and relationships between two variables . This is plotted after removing outliers 0.0000 and 99.999999 in order to see if all latitude and longitude combinations belong to same area or not.

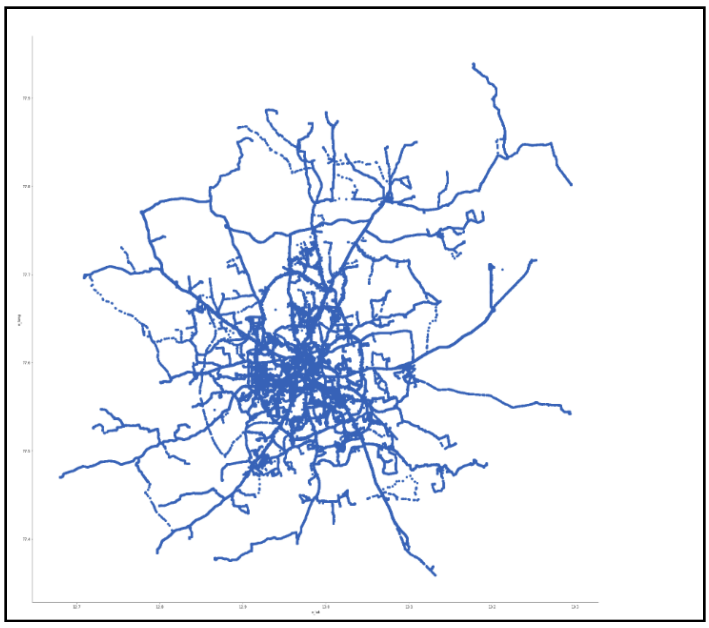


Fig3: e_lat vs e_long

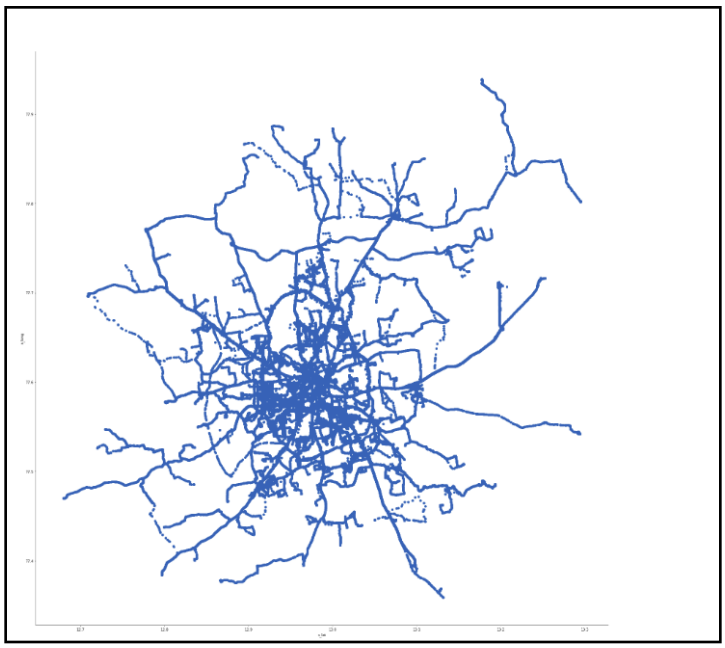


Fig4: s_lat vs s_long

- In order to see the distributions of all column we are using .hist(). A histogram is a representation of the distribution of data. This function groups the values of all given Series in the DataFrame into bins and draws all bins . This is useful when the DataFrame's Series are in a similar scale.

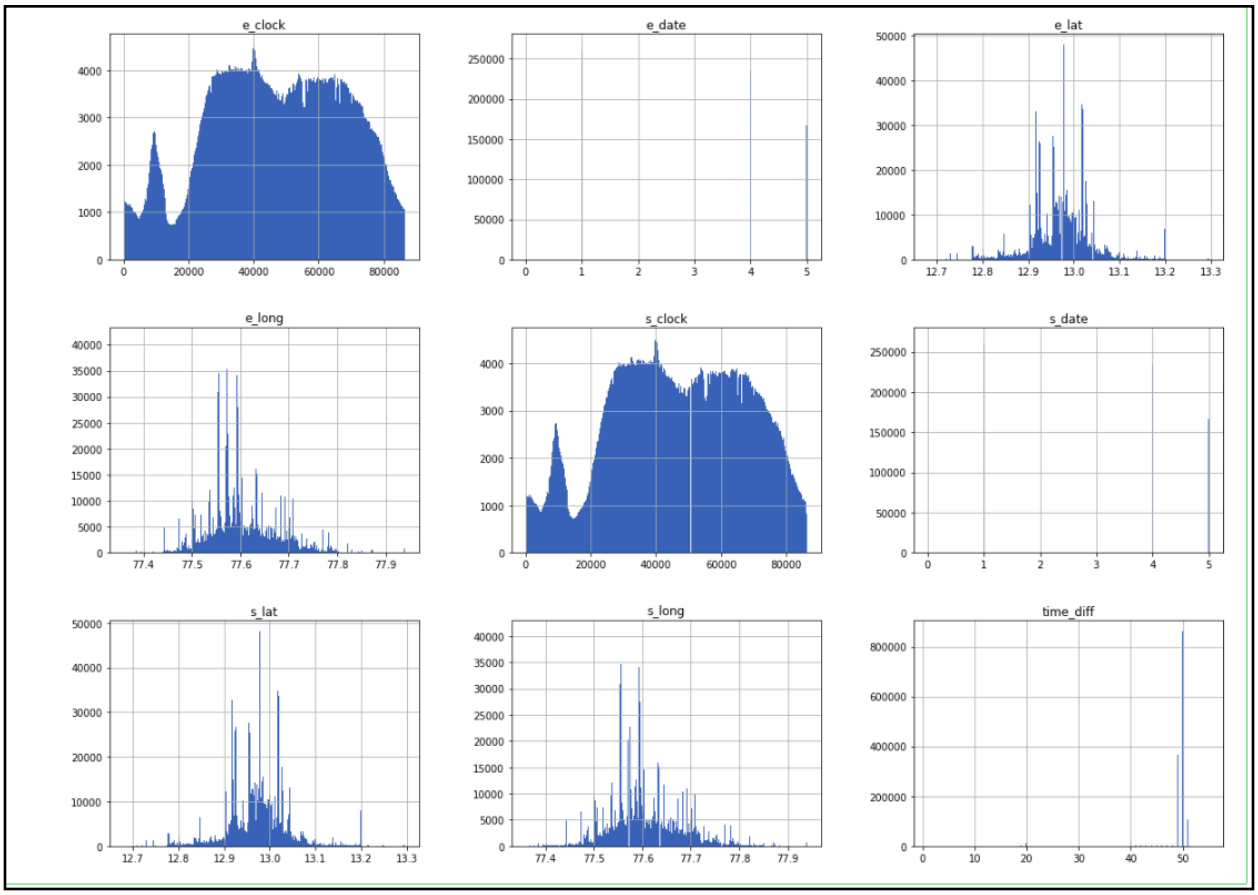


Fig5: histogram plot for every feature before normalization

After seeing this plot , decided to normalize latitude and longitude values between o to 1 because longitude being in range of 70 and latitude in range of 13 will make model to be affected more by longitude than latitude . After normalization we get the following plot.

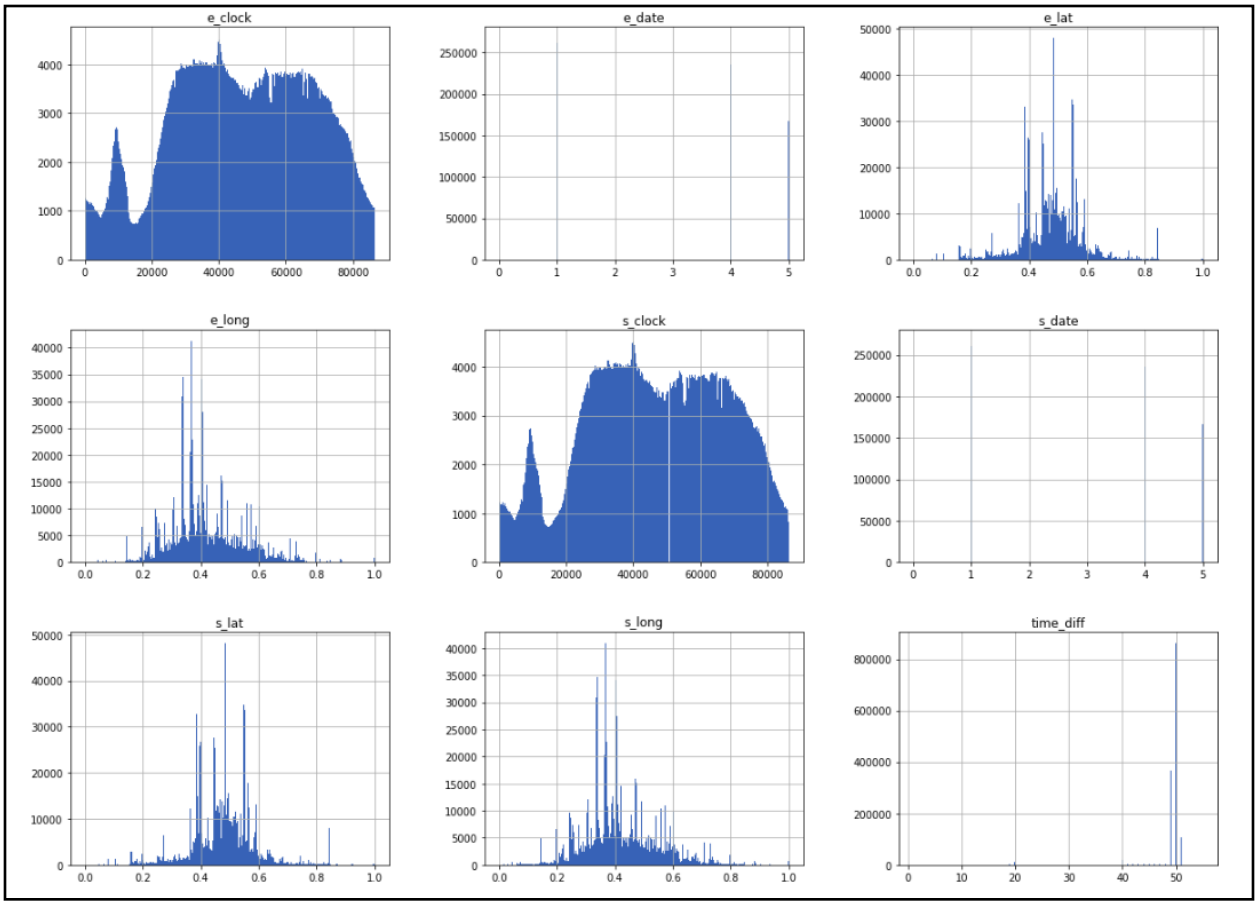


Fig6: histogram plot for every feature after normalization

Feature Engineering

- Normalization: There are features with high magnitude which will weigh high in

Euclidean distance calculation than the feature with low magnitude. Ex: longitude values will

weigh higher than latitude value as we can see in the fig5 and fig6. So we normalized there value between o and 1 using MinMaxScaler. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. in our case between zero and one. The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

Model Building

- **Building test set**: Test set provided for this project look like this:

	Id	TimeStamp	LATLONG1	LATLONG2	LATLONG3	
0	150220445	2016-07-22 06:32:30	12.845321:77.66703000000003	12.845289999999999:77.666107	12.845347:77.665565	12.8453
1	150811705	2016-07-31 05:22:33	12.965988000000001:77.535179	12.965983999999999:77.535217	12.965983999999999:77.535217	12.96598
2	150218073	2016-07-22 05:26:14	12.957028:77.559196	12.956954:77.559349	12.956980999999999:77.559494	12.9570
3	150218420	2016-07-23 02:41:32	13.020817999999998:77.502739	13.020817999999998:77.502739	13.020817999999998:77.502739	13.02081
4	150219092	2016-07-22 18:58:04	12.982555:77.603226	12.982555:77.603226	12.982555:77.603226	

Then latitude and longitude of each column are separated to find new columns of lat and long as shown bellow:

	s_clock	lat1	long1	lat2	long2	lat3	long3	
0	23550	12.845321	77.66703000000003	12.845289999999999	77.666107	12.845347	77.665565	
1	19353	12.965988000000001	77.535179	12.965983999999999	77.535217	12.965983999999999	77.535217	12.965983
2	19574	12.957028	77.559196	12.956954	77.559349	12.956980999999999	77.559494	12.957011
3	9692	13.020817999999998	77.502739	13.020817999999998	77.502739	13.020817999999998	77.502739	13.020817
4	68284	12.982555	77.603226	12.982555	77.603226	12.982555	77.603226	

Then the s_clock column is converted into time in seconds as shown bellow:

LATLONG97	LATLONG98	LATLONG99	LATLONG100	s_clock
12.91203:77.625778	12.912778999999999:77.625366	12.913666000000001:77.624832	12.914212:77.624512	23550
.965983999999999:77.535217	12.965983999999999:77.535217	12.965983999999999:77.535217	12.965983999999999:77.535217	19353
2.925502:77.54960600000003	12.925761999999999:77.54908	12.926178:77.548477	12.926508:77.548103	19574
.020817999999998:77.502739	13.020817999999998:77.502739	13.020817999999998:77.502739	13.020817999999998:77.502739	9692
12.983925:77.597755	12.983922999999999:77.597389	12.983922:77.597397	12.983919:77.597389	68284

Test is normalized between 0 and 1 using the MinMaxScalar which consists of minimum and maximum value of the respective columns of the train set. Then for the first step i.e. lat1 and long1 we take s_clock as the time difference , for second step i.e. lat2 and long2 time difference is calculated using model we generated using test set and then similarly for all latitude and longitude combination till lat100 and long100. After computing value for the 100th step calculated time difference is the required e_clock (end of trip time) for that particular row. After this we calculate time difference between s_clock

and e_clock which gives the trip duration for bus starting at lat1 and long1 and ending at lat100 and long100 for a particular datapoint. Code for this is specified bellow:

```
# For Random forest Regressor
for i in range(1,100):
    # Initial lat long time
    df_test['s_lat'] = df_test['lat{}'.format(i)]
    df_test['s_long'] = df_test['long{}'.format(i)]
    df_test['e_lat'] = df_test['lat{}'.format(i+1)]
    df_test['e_long'] = df_test['long{}'.format(i+1)]

    x_test = pd.DataFrame(df_test[['s_lat','s_long','e_lat','e_long','s_clock']],columns = ['s

    x_test_3 = min_max_scaler.fit_transform(x_test.drop(columns = 's_clock'))
    x_test_3 = pd.DataFrame(x_test_3,columns = ['s_lat','s_long','e_lat','e_long'])
    if i is 1:
        x_test_3['s_clock'] = x_test['s_clock']
    else:
        x_test_3['s_clock'] = y_pred_real_rfr
        print('Updating')
    y_pred_real_rfr = r_f_r.predict(x_test_3)

    print(i)
print(y_pred_real_rfr)
```

- **Model Training:**

As holdout method gives high variance for unseen data. So, we have used K-Fold cross validation technique. In this entire dataset is divided into 3 folds out of which 9 folds are used as train set and remaining one is used as test set and each time different sunset is selected.

We started our model building by feeding our data to Linear Regression model. It is a fairly simple model and does not perform so well. So we moved towards Decision Tree as it can implicitly perform variable screening or feature selection and it can handle both numerical and categorical data. But it is overfitting the dataset. So we moved towards random forest as compared to Decision Tree Random Forest algorithm randomly selects observation and features to build several decision tree and then averages the result and Random Forest prevents overfitting most of the time. But again it wasn't performing as desired and the algorithm is also slow for real time predictions. So we moved towards Xgboost as boosted trees are derived by optimizing an objective function (which make use of custom loss function easier), basically it can solve most all objective function that we can write gradient out. But it was not performing well and it was also overfitting the data. Xgboost model takes longer time because of the fact that trees are built sequentially. Even after using all these models performance wasn’t equal to what we desire for trip prediction because as we are prediction trip duration nearly 100 times for calculating e_clock for each row of test data set, which is making error to propagate that many times and hence performance is so weak.

Test Metrics

- In case of regression models, test metrics used are MAE, RMSE, R square and Adjusted R square. Only MAE and RMSE is used to compare between different models. Here we are using RMSE. In case of RMSE since the errors are squared before they are averaged, the RMSE gives high weight to large errors. So in your trip duration calculation case it is used because large errors are particularly undesirable.

- For cross validation, we used neg_mean_absolute_error (MAE). Bellow is a table that is showing MAE scores for different regression models. Even if linear regression model gives lowest score among all but error is huge so undesirable.

	Model	Cross Validation Scores
0	Linear Regression	-2.402674
2	Decision Tree	-2.990612
1	XGBoost Regressor	-124.684868
3	Random Forest	-309.434547

Future Scope

As we can see that our previous applied models have a very high RMSE . So these models can not be used where error is propagating after each prediction for a single datapoint. So we think that better models like neural network can help to get better RMSE as it updates its parameters after each iteration.