**FINAL PROJECT FOR COURSE MACHINE LEARNING**

***Establishment of neural network model and analysis of Radar Traffic Data***

Students:

Linxue Lai, Weicheng He

Professor:

Christophe Cerisara

Dec. 14th ,2020

**Contents**

[1 Introduction 3](#_Toc58869444)

[2 Data analysis and processing 3](#_Toc58869445)

[2.1 Data description 3](#_Toc58869446)

[2.2 Exploratory data analysis 4](#_Toc58869447)

[3 Models 7](#_Toc58869448)

[3.1 LSTM 7](#_Toc58869449)

[3.2 GRU 10](#_Toc58869450)

[3.3 Comparison 13](#_Toc58869451)

[4 Conclusion 13](#_Toc58869452)

[5 References 14](#_Toc58869453)

[Annexes 1: map visualization 15](#_Toc58869454)

[Annexes 2: main.py 16](#_Toc58869455)

## 1 Introduction

This report is about our final project for the course Machine Learning [1]. We worked in a team of two members. Our task is to build a deep learning model that predicts the traffic volume. The data that should be analyzed is called “Radar Traffic Data”, which can be download in Kaggle [2]. The traffic data is collected from radar sensors deployed by the city of Austin.

This report is developed by the following parts: data analysis and processing, model construction and parameter setting, experimental process and conclusions.

The following is our work plan:

|  |  |
| --- | --- |
| Duration | task |
| 16/11/2020 - 23/11/2020 | * Start the project: create a github project for our co-work, create a shared Google doc and a shared kaggle notebook for sharing ideas or articles referring to our project. * Conduct preliminary data analysis and read related papers |
| 24/11/2020 - 01/12/2020 | * Chose the appropriate time series analysis model: LSTM, GRU |
| 02/12/2020 - 08/12/2020 | (Weicheng HE) Construction of model LSTM and improve model.  (Linxue LAI) Construction of model GRU and improve model. |
| 09/12/2020 - 13/12/2020 | Improve codes, summarize experimental results, write report |

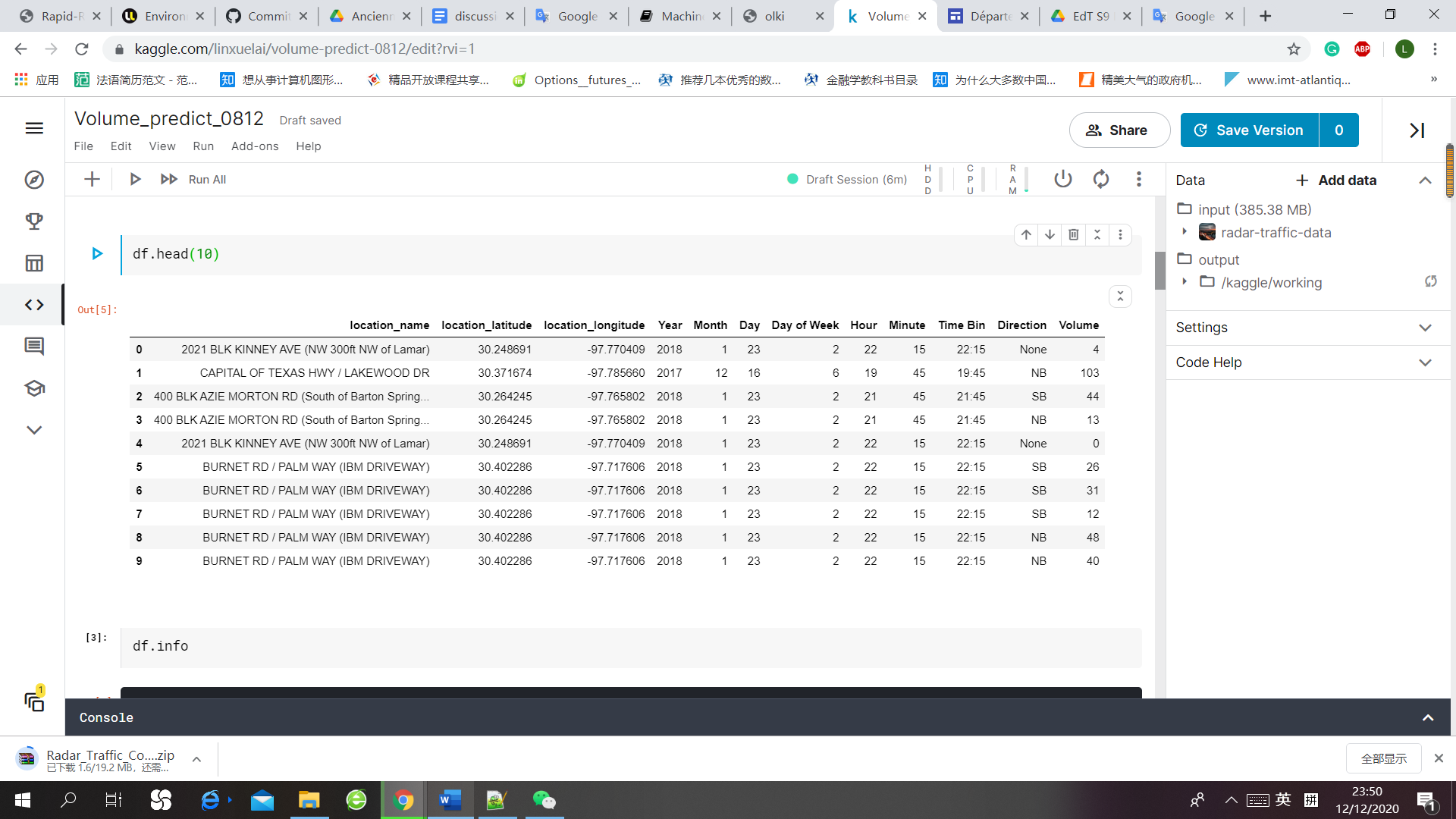
## 2 Data analysis and processing

In order to facilitate data visualization and data analysis, we use the shareable notebook of the kaggle platform for data analysis and processing.

### 2.1 Data description

Traffic data collected from the several Wavetronix radar sensors deployed by the City of Austin. Dataset is augmented with geo coordinates from sensor location dataset.

The following is an overview of some of the data:



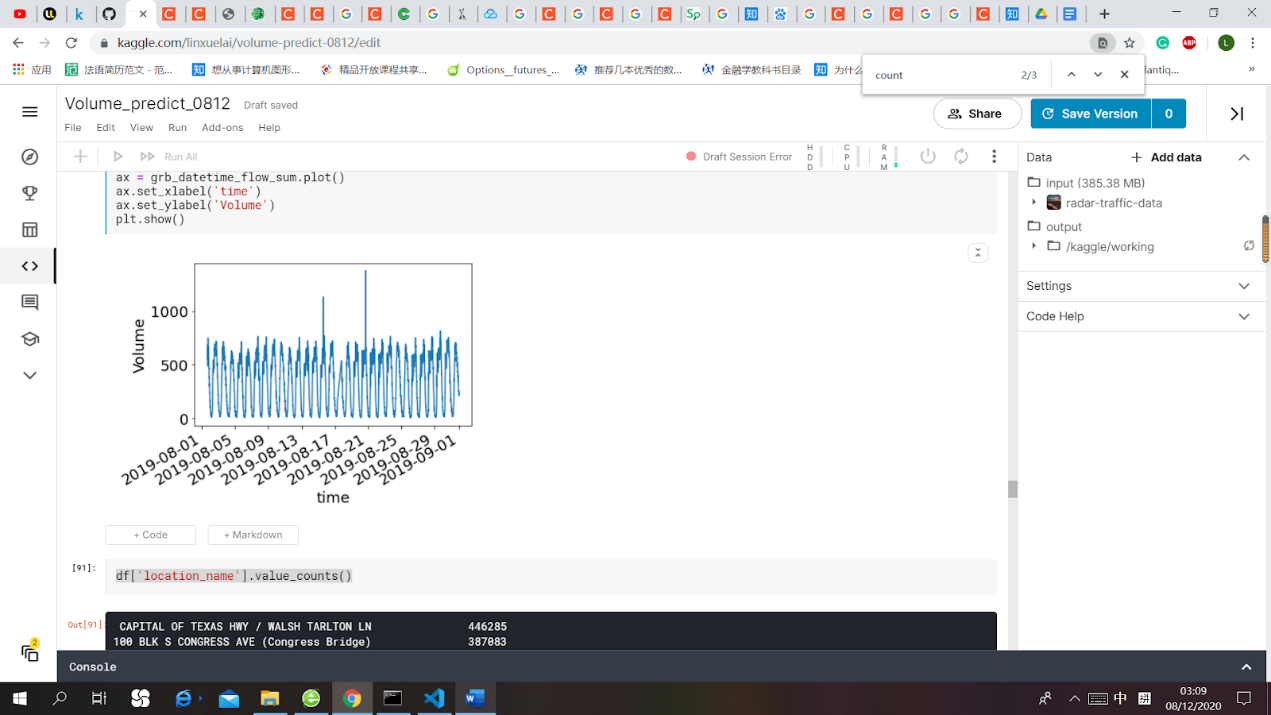
*Figure 1: Overview of dataset*

### 2.2 Exploratory data analysis

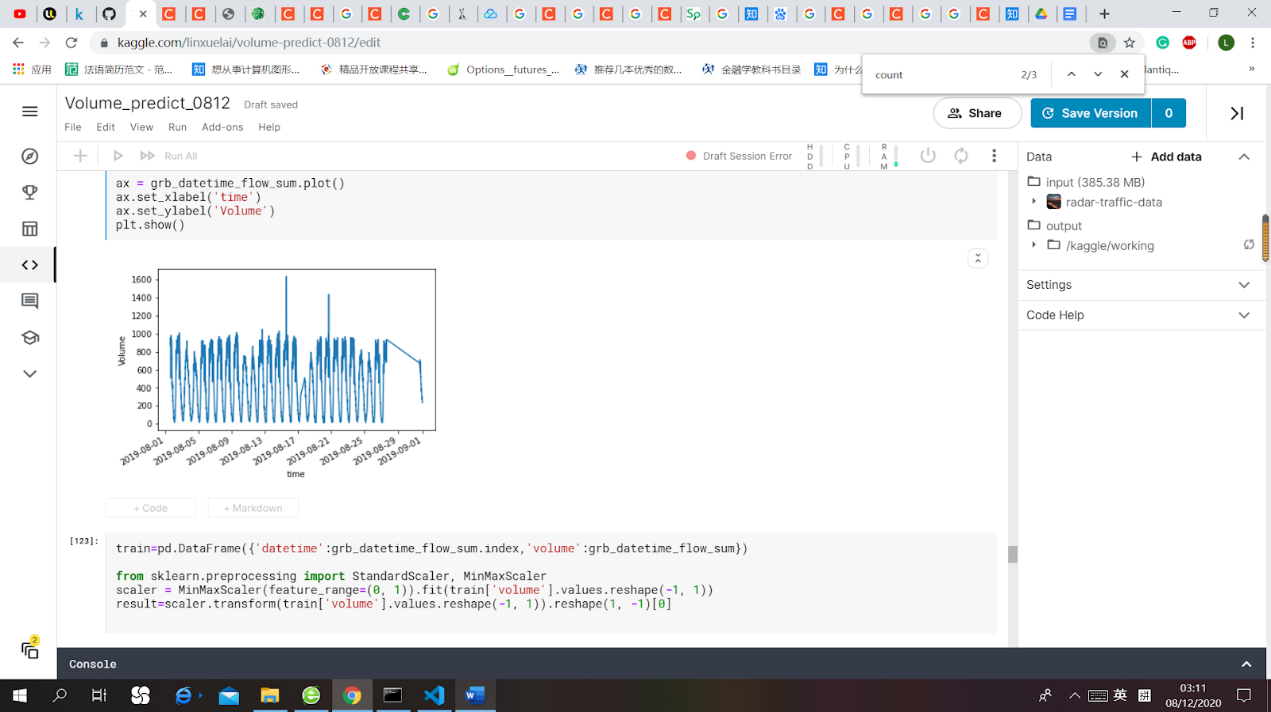
As traffic flow data has obvious periodic change characteristics, we consider using the commonly used LSTM model and GRU model for time series analysis.

There is a question to consider: Is it necessary to consider both time and space factors?

In order to simplify the problem and make it easier to understand, we first consider the changes of traffic volume at a single location over time. By looking at the data of a certain location in a certain period of time, we found that the data has an obvious cycle nature of daily changes. For example:

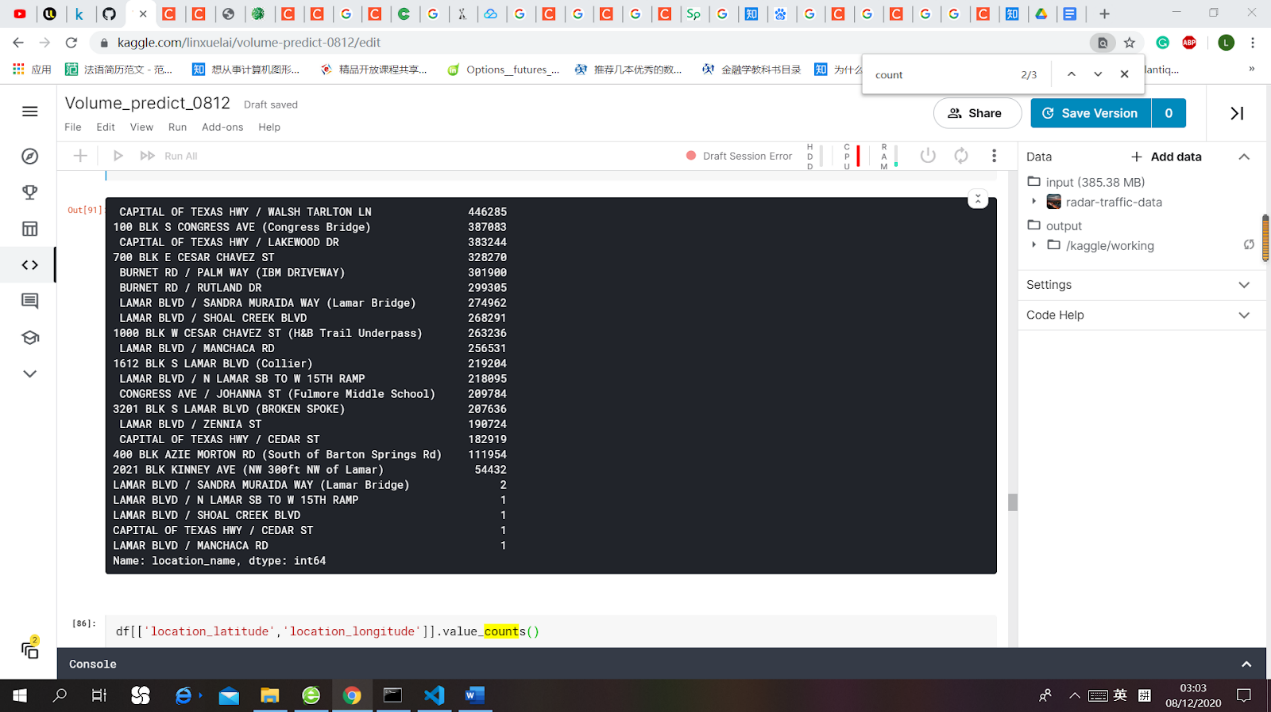
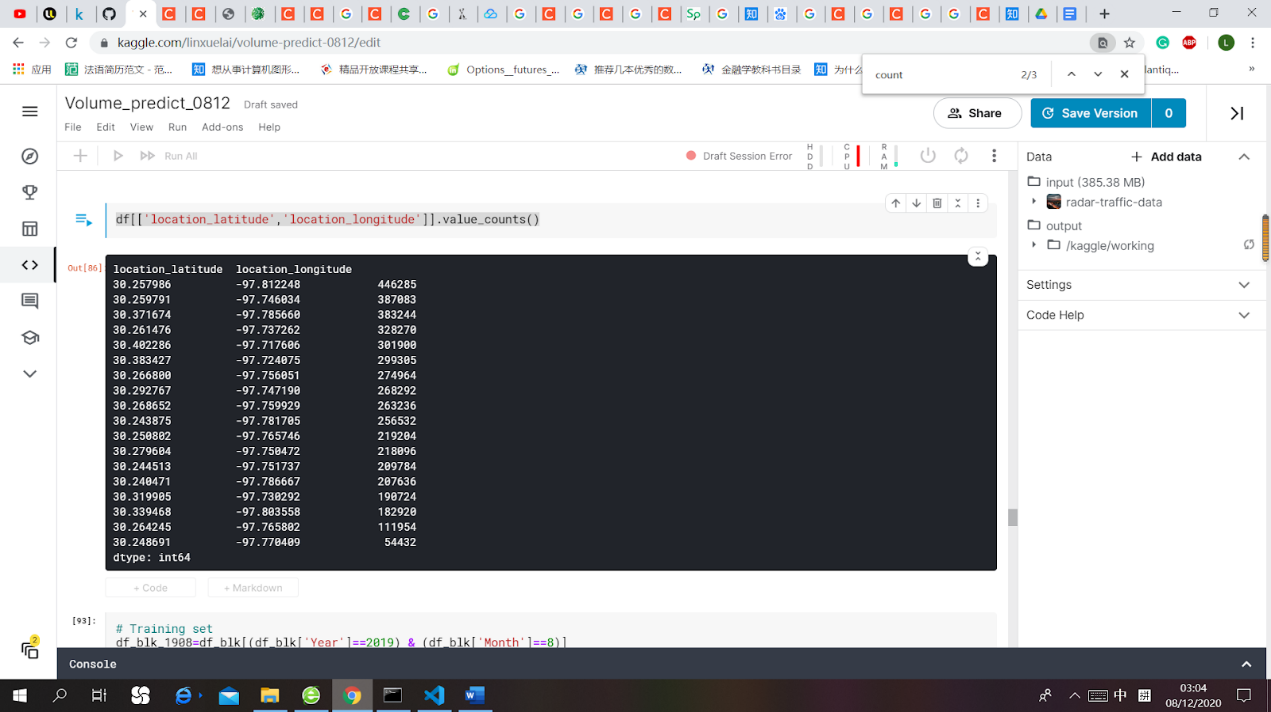


*Figure 2: Location 1 - 3201 BLK S LAMAR BLVD (BROKEN SPOKE)*



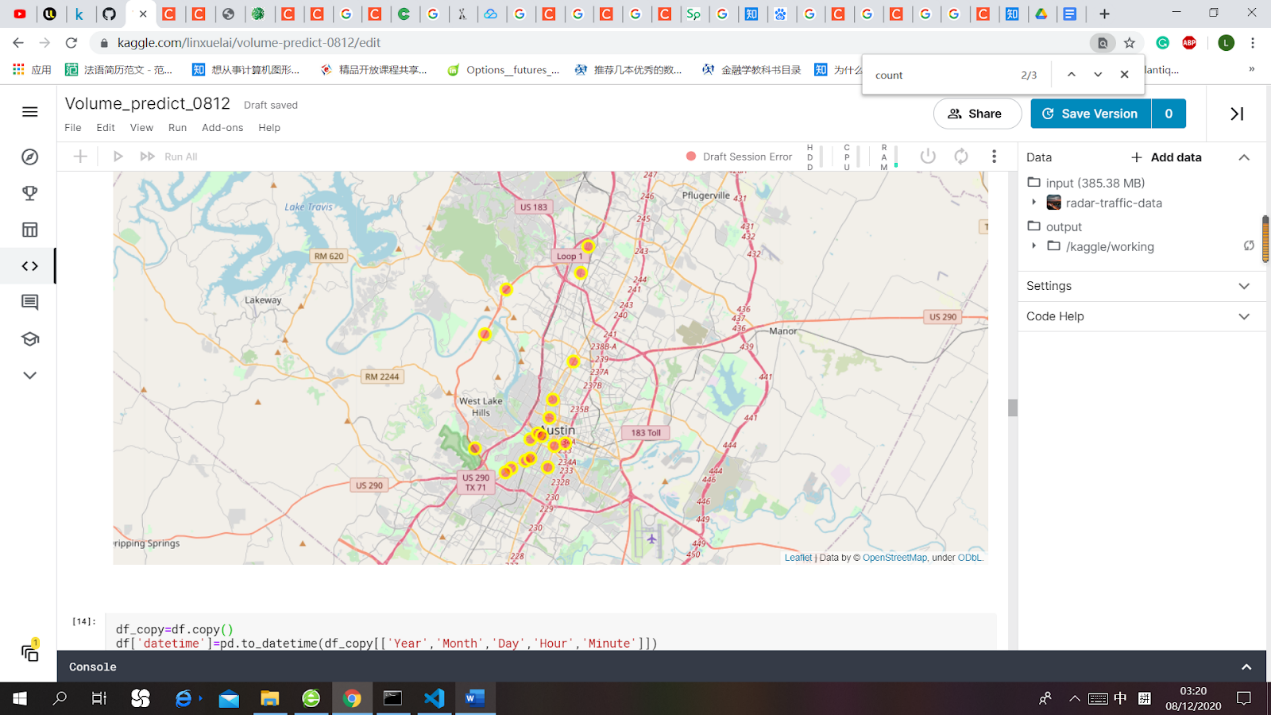
*Figure 3: Location 2 - 100 BLK S CONGRESS AVE (Congress Bridge)*

The number of locations: 23 different locations by location\_name; according to actual latitude and longitude: 18 different locations.

*Figure 4: Location names*

The following is a map-based visualization (see the codes in annexes 1):



*Figure 5: location visualization*

We use the process\_data method in the code data/data.py for data processing:

First, take the data of *location = '100 BLK S CONGRESS AVE (Congress Bridge)'* from March to June 2018 as the training set. The July data will be used as the test set.

Second, we use the training set data to implement a standardized object *scaler*, and then use the *scaler* to standardize the training set.

Since the time series prediction task needs to use historical data to predict future data, we use the time lag variable *lags* to divide the data, and finally obtain a data set of (*samples, lags*).

The divided data set still has timing characteristics in the arrangement order. Although *keras* can choose to shuffle the data during training, the execution order is to sample the data first and then shuffle, and the sampling process is still in order. of. Therefore, we use the method *np.random.shuffle* to shuffle the data and disrupt the order of the data.

# Training set

    df\_street\_1836=df\_street[(df\_street['Year']==2018)&(df\_street['Month']<7)].copy()

    # sum of volume by time bin

    grb\_datetime\_flow\_sum=df\_street\_1836.groupby('datetime')['Volume'].sum()

    train=pd.DataFrame({'datetime':grb\_datetime\_flow\_sum.index,'volume':grb\_datetime\_flow\_sum})

    # Normalization of training set

    from sklearn.preprocessing import StandardScaler, MinMaxScaler

    scaler = MinMaxScaler(feature\_range=(0, 1)).fit(train['volume'].values.reshape(-1, 1))

    result=scaler.transform(train['volume'].values.reshape(-1, 1)).reshape(1, -1)[0]

    train=[]

    for i in range(lags, len(result)):

        train.append(result[i - lags: i + 1])

    train = np.array(train)

    np.random.shuffle(train)

    X\_train = train[:, :-1]

    y\_train = train[:, -1]

    X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

The processing of the test set is similar to the above process:

#Test set

    df\_street\_1807=df\_street[(df\_street['Year']==2018) & (df\_street['Month']==7)]

    grb\_datetime\_flow\_sum=df\_street\_1807.groupby('datetime')['Volume'].sum()

    validation=pd.DataFrame({'datetime':grb\_datetime\_flow\_sum.index,'volume':grb\_datetime\_flow\_sum})

    # Normalization of test set

    from sklearn.preprocessing import StandardScaler, MinMaxScaler

    scaler = MinMaxScaler(feature\_range=(0, 1)).fit(validation['volume'].values.reshape(-1, 1))

    result=scaler.transform(validation['volume'].values.reshape(-1, 1)).reshape(1, -1)[0]

    test=[]

    for i in range(lags, len(result)):

        test.append(result[i - lags: i + 1])

    test = np.array(test)

    np.random.shuffle(test)

    X\_test = test[:, :-1]

    y\_test = test[:, -1]

    X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

    y\_test = scaler.inverse\_transform(y\_test.reshape(-1, 1)).reshape(1, -1)[0]

## 3 Models

According to data analysis results, on a certain location, traffic flow has a significantly periodic nature. So, we decided to simplify the problem by only considering time information and traffic flow.

Recurrent neural networks are a generalization of feedforward neural networks, which have been devised for handling temporal and predictive problems. We built two types of recurrent neural networks: an LSTM model and a GRU model. Each model was trained and tested traffic flow data at a certain location. At the end, the performances of two models were compared.

### 3.1 LSTM

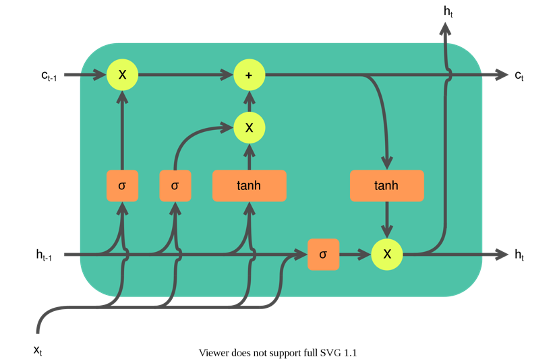
3.1.1 Introduction to LSTM

Long short-term memory (LSTM) is a particular kind of RNN, which have been explicitly designed to avoid the long-term dependency issue, like vanishing-gradient problem encountered in normal RNNs. Since it is widely used to process and predict events with time series, LSTM easily became the first model that we considered for this case.

3.1.2 Structure of LSTM model

The capability of learning long-term dependencies is due to the structure of the LSTM unit, which is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

The advantage of an LSTM cell compared to a common recurrent unit is its cell memory unit. The cell vector has the ability to encapsulate the notion of forgetting part of its previously stored memory, as well as to add part of the new information.

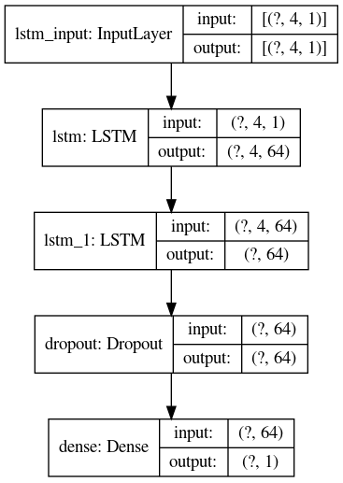


*Figure 6: LSTM network schematic diagram*

In this case, we choose to implement a two hidden layer LSTM model. The intuition is that the deep LSTM network is able to learn the temporal dependencies of the aggregate traffic flow: the LSTM unit of each layer extract a fixed number of features which are passed to the next layer. The depth of the network is to increase the accuracy of the prediction. In this case, we consider that two-layer LSTM is adequate.

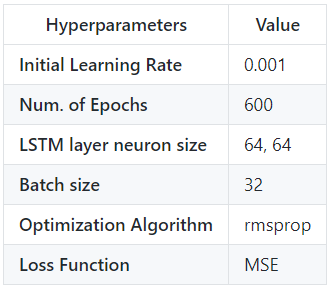
Then the LSTM layer is accompanied by a Dropout layer, which help to prevent overfitting by ignoring randomly selected neurons during training, and hence reduces the sensitivity to the specific weights of individual neurons. 20% is set as a good compromise between retaining model accuracy and preventing overfitting.

In the data process step, we calculated the aggregate traffic flow measurements for every 15 minutes. Then a time lag of one hour was set for dataset preparation. In other words, we used a input which is aggregate traffic flow measurements during one hour and an output that represents traffic flow volume after one hour to train the model.



*Figure 7: 2 hidden-layer LSTM model structure*

The implementation of this LSTM model is done in Python, using Keras and Tensorflow, as backend. The chosen hyper-parameters are reported in Table 1. They were tuned in order to get a good tradeoff between the prediction accuracy and the training time.



*Table 1: Training Hyperparameters*

3.1.3 Results of LSTM model

We evaluate the prediction error by serval assessment indicator as shown in the Table 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metrics** | **MAE** | **MSE** | **RMSE** | **MAPE** | **R2** | **Explained variance score** |
| LSTM | 27.58 | 1289.14 | 35.90 | 12.66% | 0.9636 | 0.9654 |

*Table 2: LSTM Model assessment*

### 3.2 GRU

3.2.1 Introduction to GRU

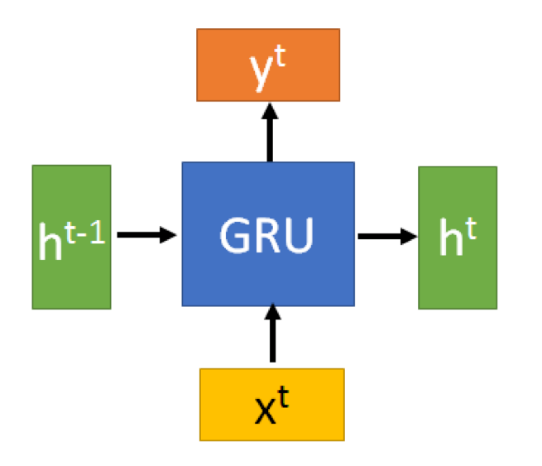
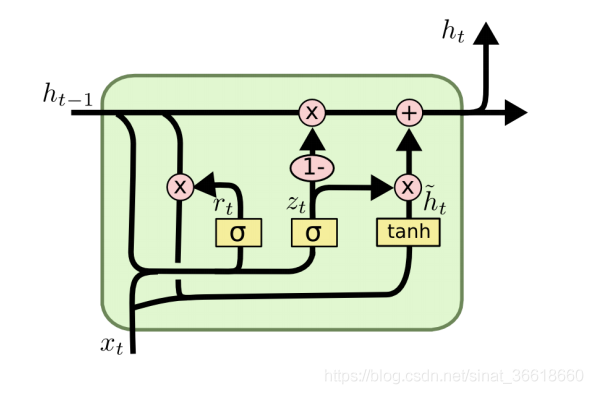
GRU (Gate Recurrent Unit) is a type of Recurrent Neural Network (RNN). Like LSTM (Long-Short Term Memory), it is also proposed to solve problems such as long-term memory and gradients in back propagation.

In many cases, GRU and LSTM are almost the same in actual performance, so why do we use the newcomer GRU (proposed in 2014).

*“We choose to use Gated Recurrent Unit (GRU) (Cho et al., 2014) in our experiment since it performs similarly to LSTM (Hochreiter & Schmidhuber, 1997) but is comutationallly cheaper.” (R-NET: MACHINE READING COMPREHENSION WITH SELF-MATCHING NETWORKS (2017))*

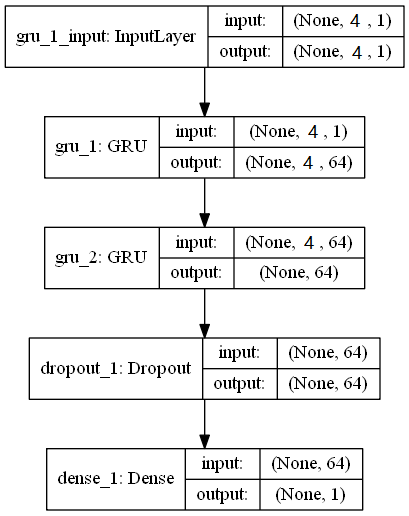
3.2.2 Structure of GRU model

The input and output structure of GRU is the same as that of ordinary RNN. There is a current input xt and a hidden state ht-1 passed down from the previous node. This hidden state contains information about the previous node. Combining xt and ht-1, GRU will get the output yt of the current hidden node and the hidden state ht passed to the next node.

*Figure 8: (a) GRU input and output structure; (b) GRU network schematic diagram*

We use a 2-hidden layer GRU structure, as shown below:



*Figure 9: 2-hidden layer GRU structure*

The model implementation code is as follows:

In the code *main.py*:

lags=4

units=[lags,64,64,1]

In the code model/model.py :

def get\_gru(units):

    """GRU(Gated Recurrent Unit)

    Build GRU Model.

    # Arguments

        units: List(int), number of input, output and hidden units.

    # Returns

        model: Model, nn model.

    """

    model = Sequential()

    model.add(GRU(units[1], input\_shape=(units[0], 1), return\_sequences=True))

    model.add(GRU(units[2]))

    model.add(Dropout(0.2))

    model.add(Dense(units[3], activation='sigmoid'))

    return model

The experiment procedure (find code main.py in annexes 2):

One of the locations is selected, and a location in a certain period of time is selected for GRU training, and only time sequence is considered.

The data is at 15min intervals.

We run python main.py to do our whole experiment, the code will do firstly initialization of some important parameters in *Class predict\_volume*:

    def \_\_init\_\_(self,raw\_data,location,units,lags,config):

        self.raw\_data=raw\_data

        self.location=location

        self.units=units

        self.lags=lags

        self.config=config

        self.X\_train, self.y\_train, self.X\_test, self.y\_test, self.scaler = process\_data(self.raw\_data,self.location,self.lags)

Then train the gru\_model for our prepared dataset.

3.2.3 Results of GRU model

Finally，We use the trained model to make predictions on the test set and evaluate the results.

Here use MAE, MSE, RMSE, MAPE, R2, explained\_variance\_score several indicators to evaluate the regression prediction results.

def MAIN():

    lags = 4

    config = {"batch": 256, "epochs": 600}

    raw\_data = 'data/Radar\_Traffic\_Counts.csv'

    location='100 BLK S CONGRESS AVE (Congress Bridge)'

    units=[lags,64,64,1]

    pv=predict\_volume(raw\_data,location,units,lags,config)

    pv.training()

    pv.model\_evaluation()

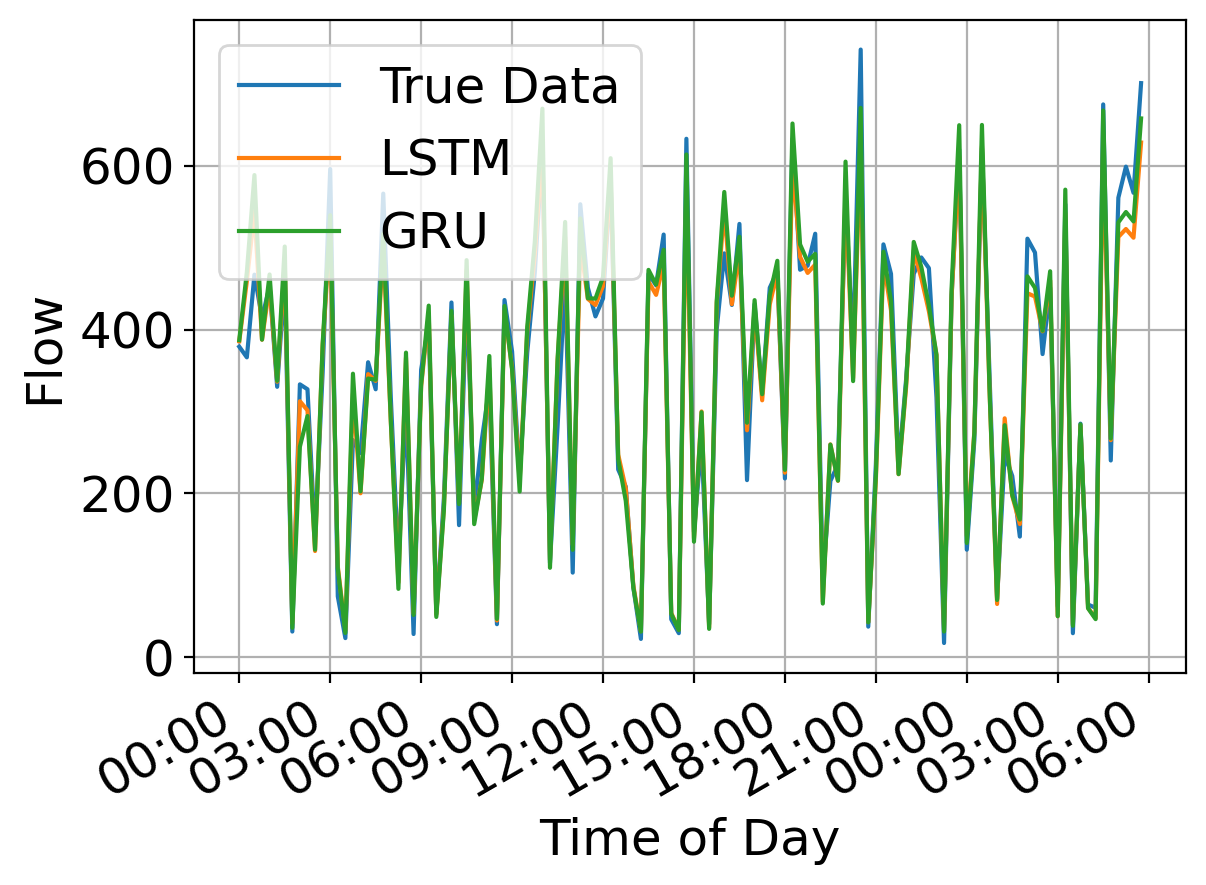
We evaluate the prediction error by serval assessment indicator as shown in the Table 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metrics** | **MAE** | **MSE** | **RMSE** | **MAPE** | **R2** | **Explained variance score** |
| GRU | 26.00 | 1284.67 | 35.84 | 12.25% | 0.9671 | 0.9671 |

*Table 3: GRU Model assessment*

### 3.3 Comparison

As shown in the Figure 5 below, the two types of RNN model have very similar performance. Both of them can achieve an accurate enough prediction results on traffic flow volume. By looking at the different model assessment indicators, we can get a more precise comparison. The two models have almost the same R squared value: around 0.96, which means 96% of the variance for the prediction target can be explained by the predictors in the deep learning model. In addition, GRU has a slightly lower MAPE, indicating that GRU shows a little better performance considering the percentage error.



*Figure 10: prediction plot of two models*

## 4 Conclusion

In this project we propose a LSTM model and a GRU model for traffic flow prediction. We compared the predictions of the LSTM and GRU models and found that in our research, the GRU NN model performed slightly better than the LSTM NN model. On average, the MAE and MSE of the GRU NN model are smaller than those of the LSTM NN model.

In future work, the influence of location factors can be considered. Combining the influence and relation of different locations and the factor of time, we can consider using methods such as Spatial-Temporal Graph Convolutional Networks for traffic flow prediction.

## 5 References

[1] course link: <https://members.loria.fr/CCerisara/#courses/machine_learning/>

[2] data link: <https://www.kaggle.com/vinayshanbhag/radar-traffic-data?select=Radar_Traffic_Counts.csv>

[3] Find codes of this project in github: <https://github.com/LinxueLAI/FinalProject>

## Annexes 1: map visualization

import folium

import pandas as pd

df = pd.read\_csv("/kaggle/input/radar-traffic-data/Radar\_Traffic\_Counts.csv")

# define the world map

world\_map = folium.Map()

# display world map

world\_map

lat\_lon = pd.DataFrame(df,columns=['location\_name','location\_latitude','location\_longitude'])

lat\_lon.drop\_duplicates(['location\_name','location\_latitude','location\_longitude'], keep='first', inplace=True)

print(lat\_lon)

lat\_lon.drop\_duplicates(['location\_latitude','location\_longitude'], keep='first', inplace=True)

print(lat\_lon)

print(len(lat\_lon))

# get the data in map

limit = len(lat\_lon)

data = lat\_lon.iloc[0:limit, :]

latitude,longitude = lat\_lon['location\_latitude'][0],lat\_lon['location\_longitude'][0]

# Instantiate a feature group in the dataframe

index = folium.map.FeatureGroup()

# Loop through the data and add each to the feature group

for lat, lng, in zip(lat\_lon.location\_latitude, lat\_lon.location\_longitude):

index.add\_child(

folium.CircleMarker(

[lat, lng],

radius=7, # define how big you want the circle markers to be

color='yellow',

fill=True,

fill\_color='red',

fill\_opacity=0.4

)

)

# Add to map

v\_map = folium.Map(location=[latitude, longitude], zoom\_start=12)

v\_map.add\_child(index)

## Annexes 2: main.py

import numpy as np

import pandas as pd

from data.data import process\_data

from evaluation import MAPE, eva\_regress, plot\_results

from model.model import get\_lstm, get\_gru

from train import train\_model

from tensorflow.keras.models import Model,load\_model

import matplotlib.pyplot as plt

class predict\_volume():

    def \_\_init\_\_(self,raw\_data,location,units,lags,config):

        self.raw\_data=raw\_data

        self.location=location

        self.units=units

        self.lags=lags

        self.config=config

        self.X\_train, self.y\_train, self.X\_test, self.y\_test, self.scaler = process\_data(self.raw\_data,self.location,self.lags)

    def training(self):

        gru\_model= get\_gru(self.units)

        train\_model(gru\_model, self.X\_train, self.y\_train, 'GRU', self.config)

        lstm\_model= get\_lstm(self.units)

        train\_model(lstm\_model, self.X\_train, self.y\_train, 'LSTM', self.config)

    def model\_evaluation(self):

        y\_preds = []

        lstm = load\_model('model/LSTM.h5')

        gru = load\_model('model/GRU.h5')

        models = [lstm, gru]

        names = ['LSTM', 'GRU']

        for name, model in zip(names, models):

            file = 'images/' + name + '.png'

            # plot\_model(model, to\_file=file, show\_shapes=True) # pydotplus.graphviz.InvocationException: GraphViz's executables not found

            predicted = model.predict(self.X\_test)

            predicted = self.scaler.inverse\_transform(predicted.reshape(-1, 1)).reshape(1, -1)[0]

            y\_preds.append(predicted[:120])

            print(name)

            eva\_regress(self.y\_test, predicted)

        plot\_results(self.y\_test[: 120], y\_preds, names)

def MAIN():

    lags = 4

    config = {"batch": 256, "epochs": 600}

    raw\_data = 'data/Radar\_Traffic\_Counts.csv'

    location='100 BLK S CONGRESS AVE (Congress Bridge)'

    units=[lags,64,64,1]

    pv=predict\_volume(raw\_data,location,units,lags,config)

    pv.training()

    pv.model\_evaluation()

if \_\_name\_\_ == '\_\_main\_\_':

    MAIN()