Data Preprocessing

To create a working dataset that we can use to train a Graph Recurrent Neural Network (GRNN) model, please follow these steps for the appropriate operating system (OS). This tutorial assumes the user knows the basic commands to use console.

Part 1: Setting Up the Environment

1. Install Python 3.6.8 and Pip. This is the Python version that we tested for Pytorch, which is a library that is required for training the GRNN.

   Windows: Download **Python 3.6.8** and follow the instructions in this **link** to install **Python** then follow the instructions from this **link** to install **Pip**. (**Green: 64-bit OS**, **Orange: 32-bit OS**)

   Note: The link shows the installation for Python 3.7.0, but it still applies for Python 3.6.8.

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		48f393a04c2e66c77bfc114e589ec630	23010188	SIG
XZ compressed source tarball	Source release		51aac91bdf8be95ec0a62d174890821a	17212420	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	eb1a23d762946329c2aa3448d256d421	33258809	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	786c4d9183c754f58751d52f509bc971	27073838	SIG
Windows help file	Windows		0b04278f5bdb8ee85ae5ae66af0430b2	7868305	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	73df7cb2f1500ff36d7dbeeac3968711	7276004	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	72f37686b7ab240ef70fdb931bdf3cb5	31830944	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	39dde5f535c16d642e84fc7a69f43e05	1331744	SIG
Windows x86 embeddable zip file	Windows		60470b4cceba52094121d43cd3f6ce3a	6560373	SIG
Windows x86 executable installer	Windows		9c7b1ebdd3a8df0eebfda2f107f1742c	30807656	SIG
Windows x86 web-based installer	Windows		80de96338691698e10a935ecd0bdaacb	1296064	SIG

   Mac: Follow this **link** to install **Homebrew, Python, and Pip**

2. Setup the Python Virtual Environment (PVE). PVE help manages the project's dependencies.

   Windows: Follow the instructions from this **link**

   Mac: Follow this **link**

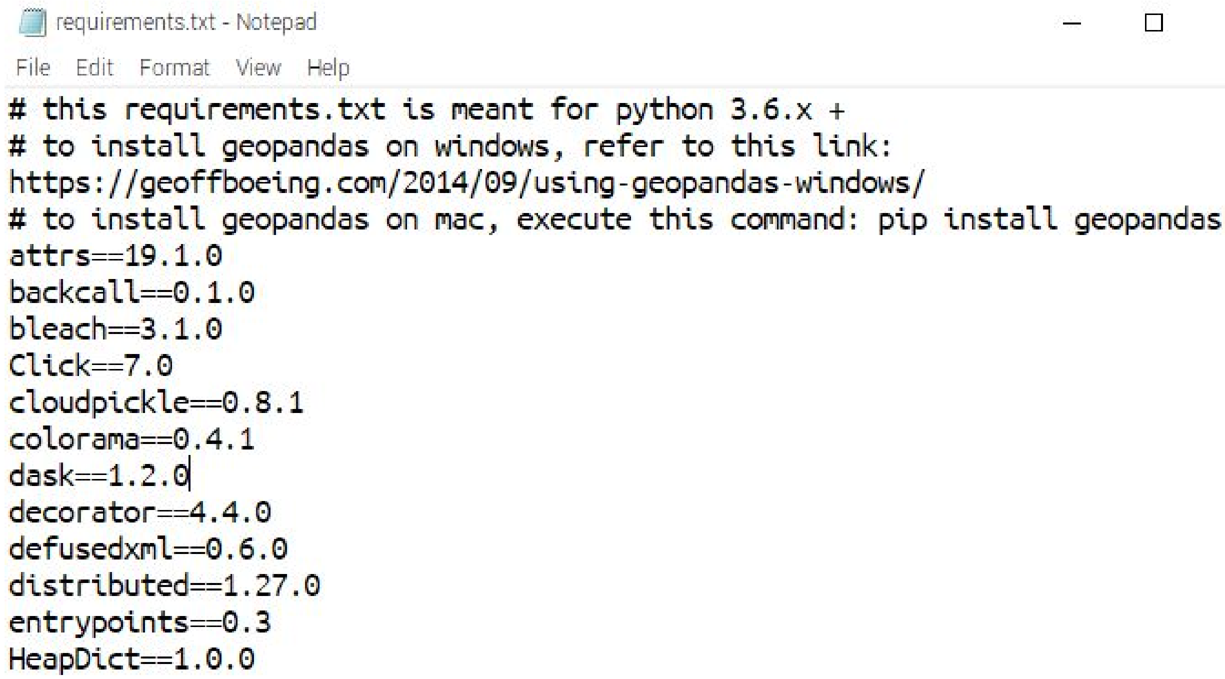
   Note: Make sure PVE is activated before continuing. **You should see the environment name in parenthesis**.

```
C:\Users\Hue-sama\Documents\venv\data_science\Scripts
λ activate

C:\Users\Hue-sama\Documents\venv\data_science\Scripts
(data_science) λ cd ..
```

3. Navigate to project directory and install the dependencies from **requirements.txt**

   Windows and Mac: **pip install -r requirements.txt**




```
# this requirements.txt is meant for python 3.6.x +
# to install geopandas on windows, refer to this link:
https://geoffboeing.com/2014/09/using-geopandas-windows/
# to install geopandas on mac, execute this command: pip install geopandas
attrs==19.1.0
backcall==0.1.0
bleach==3.1.0
Click==7.0
cloudpickle==0.8.1
colorama==0.4.1
dask==1.2.0
decorator==4.4.0
defusedxml==0.6.0
distributed==1.27.0
entrypoints==0.3
HeapDict==1.0.0
```

4. Install OSMNX library, which is required to create the road network.

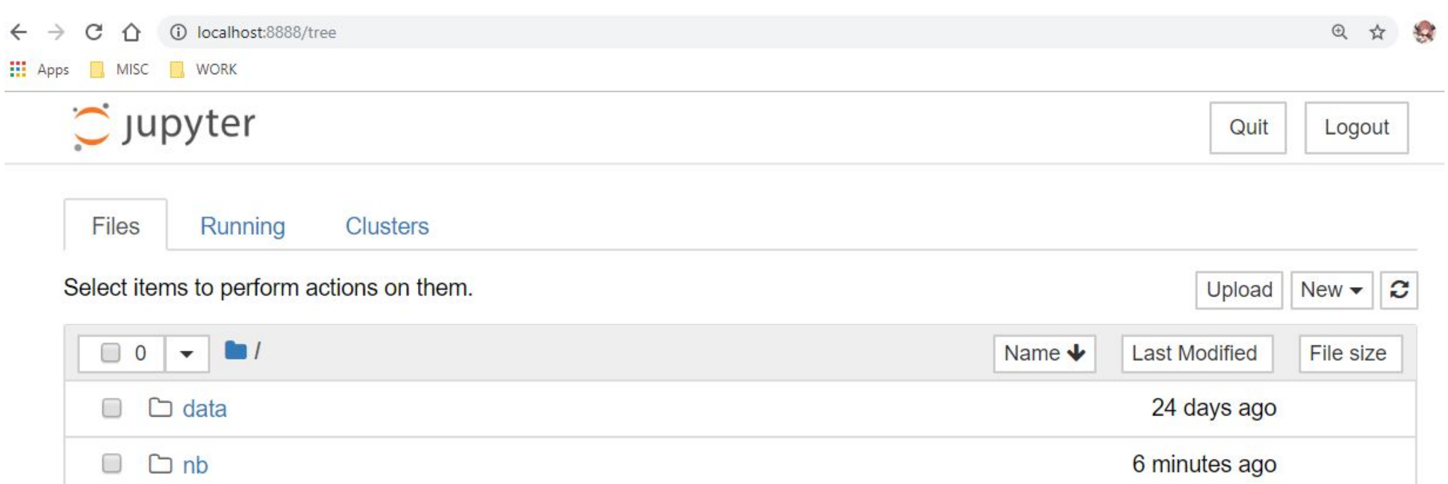
   Windows: Follow this [link](#) to install the **OSMNX** dependencies

   Mac: **pip install geopandas**

5. Start Jupyter Notebook from the console to run the data preprocessing scripts. Note: **Jupyter Notebook will open in a browser**.

   cd <project directory>

   jupyter notebook



Part 2: Processing the Data

1. Open **jams-data-preprocessing** in **nb**
2. Set the path to the directory where the data is stored

```python
# set path to input file
filepath = '../data/jams-head.csv'
```

3. Click **Run** in the toolbar to run every cells.

   Note: When all the cells finish running, you should have these files:

   1. **node.csv** contains all intersections of the road network where each row of data is the **OSM ID, latitude, longitude**

   node.csv

```
122814468,34.029596,-118.2915496
1842872326,33.974541,-118.4290464
122814472,34.029569,-118.2959172
1788084236,34.0336024,-118.2625987
123207702,34.2687559,-118.5875771
123207704,34.2689377,-118.5879709
1842872352,33.9748674,-118.4282765
1842872354,33.974929,-118.428114
954728505,34.1395527,-118.3805141
123207737,34.0892251,-118.1634525
123732037,34.245342,-118.3519254
123732039,34.2492822,-118.3519377
1734869065,34.063594,-118.2953941
6127616081,34.2180204,-118.5660758
6127616082,34.2180265,-118.5662466
```

   2. **segment.csv** contains all of the streets formed by two intersections found in node.csv. Each row of data has the **pseudo-edge ID, starting OSM ID, and ending OSM ID.**

segment.csv

```
0,122814468,122814472
1,122814468,123152289
2,122814468,122659220
3,1842872326,1842872352
4,1842872326,1842872317
5,122814472,122814468
6,122814472,122935689
7,122814472,122659224
8,1788084236,122648646
9,1788084236,1918477979
10,1788084236,1918477977
11,1788084236,123120142
12,123207702,123207704
13,123207702,122960426
14,123207702,122964398
```

3. **selSegs_1.csv** is a subset of the road segments in segment.csv that the model will train on. Each row contains the **pseudo-edge id**.

selSegs_1.csv

```
4219
6243
6244
7972
14036
15909
15933
15935
22754
26247
26248
33312
33313
41395
41419
```

4. **financial_district_10_knn.csv** is the dataset that we feed into GRNN model. It's created based on the pseudo-ids from selSegs_1.csv. Any missing data is imputed using KNN Regression, and scaled to a 10-minute interval for approximately 30 days. That means, each pseudo-id should have about 4320 data points.

Math: Total number of data points = ((30days * 24hrs * 60mins) / 10min) * number of selected segments. **Each row of data contains the time slot, pseudo-node id, average speed**

financial_district_10_knn.csv

```
2017-12-11 23:00:00,4214,1.7449066666666668
2017-12-11 23:10:00,4214,1.7449066666666668
2017-12-11 23:20:00,4214,1.7449066666666668
2017-12-11 23:30:00,4214,1.7449066666666668
2017-12-11 23:40:00,4214,1.7449066666666668
2017-12-11 23:50:00,4214,1.7449066666666668
2017-12-12 00:00:00,4214,1.833889
2017-12-12 00:10:00,4214,1.833889
2017-12-12 00:20:00,4214,1.833889
2017-12-12 00:30:00,4214,1.833889
2017-12-12 00:40:00,4214,1.833889
2017-12-12 00:50:00,4214,1.833889
2017-12-12 01:00:00,4214,1.833889
2017-12-12 01:10:00,4214,1.833889
2017-12-12 01:20:00,4214,1.833889
```

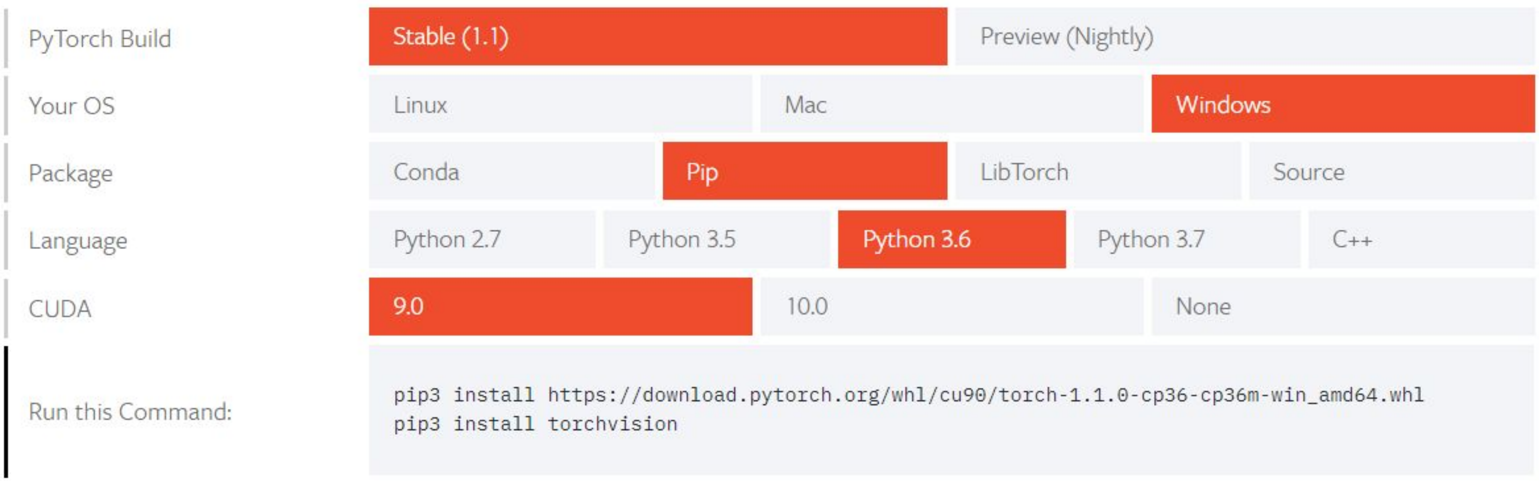
```
2017-12-11 23:00:00,6273,2.97187375
2017-12-11 23:10:00,6273,2.97187375
2017-12-11 23:20:00,6273,2.97187375
2017-12-11 23:30:00,6273,2.97187375
2017-12-11 23:40:00,6273,2.97187375
2017-12-11 23:50:00,6273,2.97187375
2017-12-12 00:00:00,6273,3.25486375
2017-12-12 00:10:00,6273,3.25486375
2017-12-12 00:20:00,6273,3.25486375
2017-12-12 00:30:00,6273,3.25486375
2017-12-12 00:40:00,6273,3.25486375
2017-12-12 00:50:00,6273,3.25486375
2017-12-12 01:00:00,6273,3.25486375
2017-12-12 01:10:00,6273,3.25486375
2017-12-12 01:20:00,6273,3.25486375
```

Training and Testing GRNN

The following steps are used to train the GRNN model using Pytorch. Since there were a lot of issue with Pytorch, we only tested and used Windows machine. DISCLAIMER: GRNN is RAM intensive! Use CUDA enabled GPU to train faster.

Part 1: Setting Up the Environment

1. Activate the Python Virtual Environment
2. Select the Pytorch environment settings from [here](#) then run the command in a console to install Pytorch



3. Verify that Pytorch is installed correctly.
    a. Using a console with the PVE activated, run these commands:

       python

       >> **import torch**

       >> **print(torch.__version__)**

       output: 1.0.1
4. (Optional) Install CUDA 9.0 (tested) to enable GPU processor with Pytorch, run these commands

   pip3 install https://download.pytorch.org/whl/cu90/torch-1.0.1-cp36-cp36m-win_amd64.whl

   pip3 install torchvision

```
C:\Users\Hue-sama\Documents
(data_science) λ python
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.__version__)
1.0.1
```

5. (Optional) To use cuDNN with CUDA, create a membership from [here](#) then download cuDNN 7.5.1 (tested) for CUDA 9.0

Download cuDNN v7.5.1 (April 22, 2019), for CUDA 9.0

Library for Windows, Mac, Linux, Ubuntu and RedHat/Centos

cuDNN Library for Windows 7

cuDNN Library for Windows 10

cuDNN Library for Linux

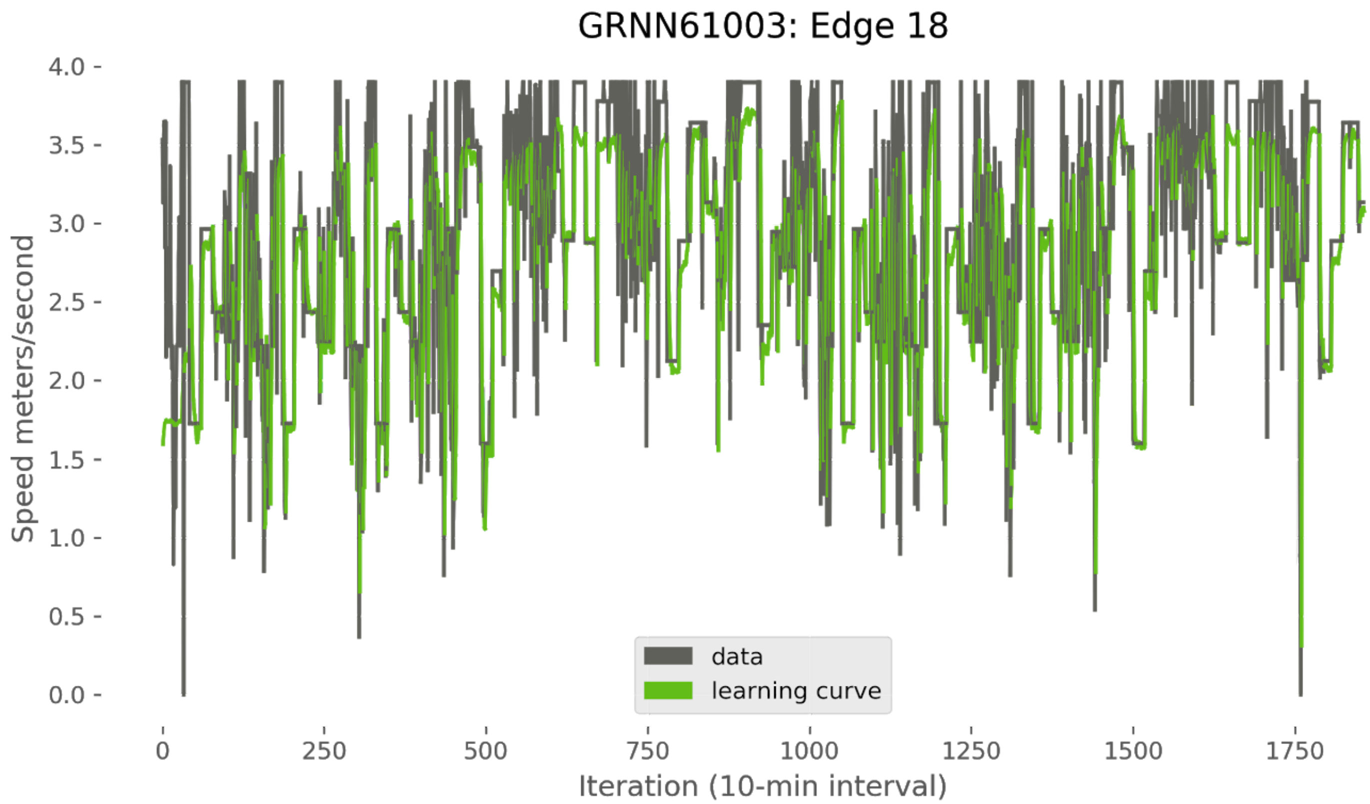
cuDNN Runtime Library for Ubuntu16.04 (Deb)

cuDNN Developer Library for Ubuntu16.04 (Deb)

Part 2: Train GRNN

6. Activate PVE, and install all the dependencies via Pip.
7. Navigate to **data_analysis/financial_district_pytorch** from project directory
8. Train GRNN. Note: The parameters after "python main.py" are hyperparameters that tune the GRNN model. In addition, the model will save at every 10 iterations until all the iterations are completed. A plot is displayed to show the progress of the model learning. The training will be incomplete if this process is interrupted. However, if you need to terminate the training, press **CTRL + C** in the console.

```
'--grnnID', type=int, default=grnn_id, help='GRNN model id')
'--taskID', type=int, default=1, help='traffic prediction task id')
'--finterval', type=int, default=10, help='interval of data')
'--alpha', type=float, default=0.1, help='traffic prediction task id') # regularization
'--batchSize', type=int, default=1, help='input batch size') # batch size per step
'--dimHidden', type=int, default=32, help='GRNN hidden state size')
'--truncate', type=int, default=144, help='BPTT length for GRNN') # interval; step size
'--nIter', type=int, default=2, help='number of epochs to train') # epoch
'--lr', type=float, default=0.01, help='learning rate')
'--showNum', type=int, default=None, help='prediction plot. None: no plot')
'--cuda', action='store_true', help='enables cuda')
'--verbal', action='store_true', help='print training info or not')
'--manualSeed', type=int, help='manual seed') # random seed is used for reproducing results
'--test', type=int, default=None, help='for several-node prediction testing')
```



python main.py --verbal --cuda --showNum 18 --dimHidden 25 --truncate 144 --lr 0.01

Part 3: Test GRNN

This part of the instructions show how to test the saved GRNN model. We run the assumption that the saved model completed the training on all iterations, so that Pytorch **model_state_dict** is properly saved. Please have the PVE activated before starting.

1. Navigate to **data_analysis/financial_district_pytorch** from project directory
2. Run this command in console
   jupyter notebook
3. Open **Validation.ipynb** and set the correct path of the saved model (**.pt file**)

```
file_path = './result/grnn29055-30int-1tid-0.0001a-48T-25D-4i-0.05lr-5487ms-1b-18sn.pt'
```

4. Run all the cells by clicking Run
   An **accuracy score** is generated for the saved model when all the cells finish running. This accuracy score measures how close the predicted and the actual values are related in percentage at each iteration. The final score is the average of all the percentage.

```
Accuracy scores: [88.24, 87.47, 88.46, 87.57, 87.04, 133.89, 134.28, 134.86, 134.34, 132.14, 133.05, 133.57, 133.36, 133.99,
132.71, 133.25, 133.45, 133.64, 133.08, 132.05, 134.46, 133.23, 134.13, 79.08, 80.2, 80.71, 79.89, 80.72, 81.49, 81.07, 81.4
5, 81.94, 80.49, 80.94, 80.92, 95.78, 94.95, 95.18, 95.23, 96.92, 97.25, 96.18, 96.92, 98.02, 98.29, 98.22, 99.15, 98.35, 99.
13, 99.47, 99.31, 99.68, 98.23, 99.49, 99.18, 101.09, 102.87, 103.96, 102.33, 84.57, 85.46, 85.77, 85.35, 85.76, 85.89, 84.9
3, 85.54, 85.77, 85.73, 84.77, 85.46, 83.42, 83.82, 84.22, 82.02, 84.39, 83.39, 83.69, 83.6, 82.58, 83.86, 83.95, 83.67, 83.3
1, 82.96, 83.47, 83.16, 83.41, 84.98, 110.34, 110.47, 108.83, 107.97, 108.27, 108.33, 107.2, 105.34, 106.16, 105.56, 106.58,
107.87, 107.46, 106.55, 106.99, 105.27, 106.75, 104.11, 83.66, 83.65, 83.58, 83.68, 82.89, 82.05, 82.36, 82.69, 82.88, 82.99,
82.27, 84.09, 84.89, 84.85, 83.92, 84.4, 83.27, 83.14, 90.88, 91.29, 91.29, 90.81, 91.47, 91.2, 92.05, 92.46, 92.67, 92.56, 9
1.25, 90.44, 89.46, 90.87, 91.25, 91.15, 91.86, 91.62, 92.82, 93.64, 94.28, 94.18, 93.71, 93.77, 105.64, 106.18, 106.24, 106.
0, 109.1, 105.36, 106.6, 105.27, 106.05, 109.53, 107.41, 108.01, 108.22, 106.99, 108.21, 106.99, 108.08, 109.11, 102.32, 102.
18, 103.68, 102.19, 103.67, 103.19, 104.6, 105.57, 105.8, 104.12, 104.31, 103.54, 93.09, 94.53, 94.27, 93.93, 96.22, 94.02, 9
4.41, 93.6, 94.44, 96.02, 95.0, 95.08, 95.32, 95.87, 92.7, 95.16, 95.33, 96.73, 95.83, 94.39, 94.35, 94.81, 95.11, 96.14, 86.
01, 86.42, 86.73, 87.06, 87.48, 86.86, 86.79, 87.45, 87.13, 87.21, 86.96, 88.32, 98.58, 96.17, 97.54, 96.85, 98.09, 97.04, 9
7.51, 97.03, 96.18, 97.54, 98.5, 97.63, 97.23, 96.7, 96.88, 96.74, 96.99, 96.18, 87.23, 87.18, 87.55, 87.23, 87.85, 87.64, 8
8.38, 87.85, 88.3, 88.21, 87.27, 87.36, 87.69, 88.96, 87.25, 89.14, 87.15, 88.43, 79.0, 78.96, 78.63, 78.35, 78.95, 78.07, 7
8.34, 77.3, 77.39, 77.7, 77.55, 78.01, 77.36, 77.28, 77.56, 77.94, 78.05, 77.88, 106.18, 106.34, 106.7, 107.3, 107.66, 106.9
3, 107.63, 107.33, 107.28, 108.38, 108.61, 108.41, 108.2, 108.31, 108.37, 107.91, 107.89, 108.29, 108.26, 108.11, 108.1, 107.
76, 108.65, 107.86, 104.45, 105.01, 103.07, 103.63, 104.35, 103.73, 103.03, 104.5, 105.32, 103.97, 104.4, 103.71, 102.64, 10
3.76, 103.27, 103.54, 102.53, 101.81, 95.78, 98.71, 97.45, 98.22, 97.23, 96.17, 96.14, 95.69, 96.63, 96.25, 95.64, 96.21, 10
0.11, 100.52, 99.74, 98.4, 98.84, 99.54, 98.15, 98.67, 98.24, 98.75, 98.04, 97.26, 97.46, 97.49, 98.46, 96.96, 97.47, 97.17,
```

```
average_accuracy
```

```
95.20841662283016
```

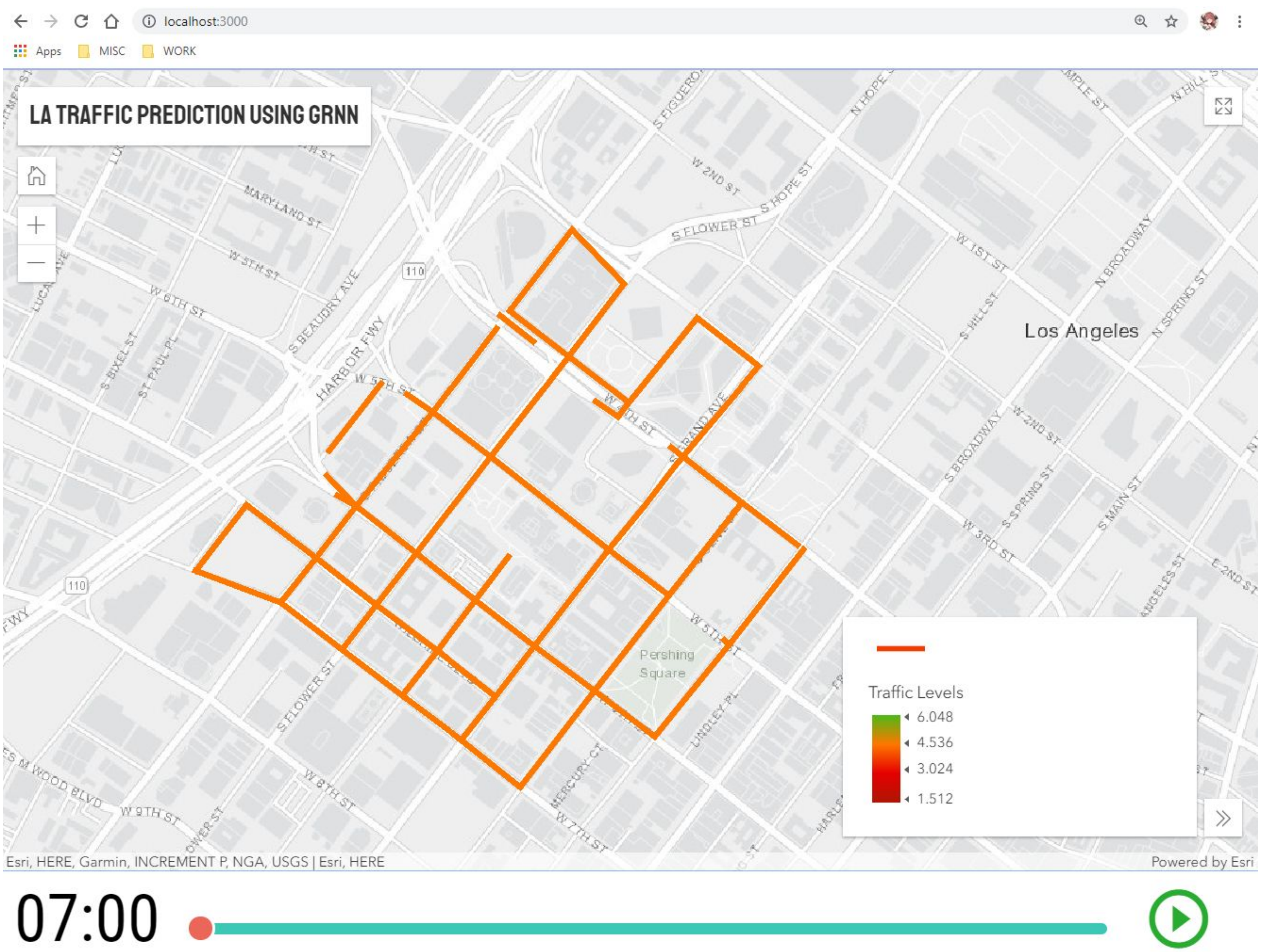
Visualization

This part of the instruction shows how to use the ArcGIS visualization. Applicable on any operating system.

1. Install Node from [here](#) (LTS version). Note: Installing Node will also install Node Package Management (NPM). If you run into trouble, follow this [tutorial](#).
2. Navigate to **arcgis_visualization** from project directory
3. In console, run this command to install the web application dependencies

   npm install
4. Start application by running this command

   node express.js
5. To view the application, open the browser and go to **localhost:3000**



6. To view the data hosted on the API server, visit **localhost:8080**

localhost:8080

Apps MISC WORK


```
// 20190517233755
// http://localhost:8080/

[
  {
    "id": 0,
    "timestamp": "2017-12-12T15:00:00.000Z",
    "start_lat": 34.047360999999995,
    "start_long": -118.2531095,
    "end_lat": 34.0487485,
    "end_long": -118.2517473,
    "avg_speed": 6.04841366332187
  },
  {
    "id": 1,
    "timestamp": "2017-12-12T15:10:00.000Z",
    "start_lat": 34.047360999999995,
    "start_long": -118.2531095,
    "end_lat": 34.0487485,
    "end_long": -118.2517473,
    "avg_speed": 6.219771645799379
  },
```


7. Terminate the application by pressing **CTRL + C** in console

Tip: To save time, once you know where the environment and project is created/located - save the commands in a text file so you can simply copy and paste the commands in the console next time!

```
# activate python environment
cd \Users\Hue-sama\Documents\venv\data_science\Scripts
activate

# nagivate to project directory
cd \Users\Hue-sama\Documents\github\sdg10_prototype\data_analysis
\data_preprocessing\nb

# start jupyter notebook
jupyter notebook
```