Notebook_Learning2

April 24, 2021

1 Machine Learning on Polutions from Transportation

In the following program, we would guide you through using Pandas to process the emission data for Tensorflow Machine Learning. Then we would teach you how to create and train your Tensorflow model. Answer the questions when you see Q; follow the steps in **To-do**. When you see something like ^{D1} or ^{M1} next to problems, you should refer to the rubrics to see how the the problems will be graded as those problems are worth points.

Note: Hit the "Run" button to run the program block by block. We don't recommend you to use "Run All" in "Cell" because the first few blocks only need to be run once and they take some time to run.

1.1 Import Libraries

The following block is used in Python to import necessary libraries. You might encounter error while trying to import tensorflow. This is becuase Tensorflow is not a default library that comes with the Python package you installed. Go to this link https://www.tensorflow.org/install/pip#system-install and follow the instructions on installing Tensorflow. If you encounter problems while trying to install Tensorflow you can add `--user` after `pip install`. This is because you did not create a virtual environment for your python packages. You can follow Step 2 on the website to create a virtual environment (recommended) or you can just install the package in your HOME environment. You might encounter error while trying to import other libraries. Please use the same `pip` method described above.

- `pandas` is used to process our data.
- `numpy` is a great tool for mathematical processing and array creations.
- `sklearn` is used to split the data into Training, Testing, and Validation set.

[2]:
```
# Import Libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
import seaborn as sns
from matplotlib import pyplot as plt
```

1.1.1 Import Tensorboard

[3]:
```
# Load the TensorBoard notebook extension.
%load_ext tensorboard
from datetime import datetime
from packaging import version

print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >= 2, \
    "This notebook requires TensorFlow 2.0 or above."
import tensorboard
tensorboard.__version__
```

```
TensorFlow version:  2.4.1
```

[3]:
```
'2.4.1'
```

1.2 Load and Clean up the Dataset

1.2.1 Load the Dataset

To process the data, save the .csv file you downloaded from the Google Drive to the same directory where this Notebook is at. * `pd.read_csv("file path")` reads the data into emission_train

```
* Note that we call `pd` directly becuase we import `pandas as pd`
```

- `.head()` returns the first 100 rows of data. Note that when displaying, some rows are truncated. It is normal since the rows are too long.
- `.describe()` shows statistical data for our data frame.

[4]:
```
# loading the large data set, it may takes a while.
emission_train = pd.read_csv("data/emission_comma.csv", delimiter=",", quoting␣
↪= 3)
```

Here is a link that contains information about meaning of the columns in "emission.csv": https://sumo.dlr.de/docs/Simulation/Output/EmissionOutput.html

[5]:
```
display(emission_train.head(100))
display(emission_train.describe())
```

```
    timestep_time  vehicle_CO  vehicle_CO2  vehicle_HC  vehicle_NOx  \
0             0.0       15.20      7380.56        0.00        84.89
1             0.0        0.00      2416.04        0.01         0.72
2             1.0       17.92      9898.93        0.00       103.38
3             1.0        0.00         0.00        0.00         0.00
4             1.0      164.78      2624.72        0.81         1.20
..            …          …          …           …           …
95            7.0       23.44      2578.06        0.15         0.64
96            7.0      732.32     18759.70        3.34         3.79
97            7.0      294.68      6949.38        1.29         1.47
```

```
98            7.0      236.07      4292.19        0.97         0.93
99            7.0      179.19      1228.61        0.64         0.31

    vehicle_PMx  vehicle_angle    vehicle_eclass  vehicle_electricity  \
0          2.21          50.28        HBEFA3/HDV                  0.0
1          0.01          42.25   HBEFA3/PC_G_EU4                  0.0
2          2.49          50.28        HBEFA3/HDV                  0.0
3          0.00          42.25   HBEFA3/PC_G_EU4                  0.0
4          0.07         357.00   HBEFA3/PC_G_EU4                  0.0
..          …             …              …                  …
95         0.05           0.13  HBEFA3/LDV_G_EU6                  0.0
96         1.19         179.93  HBEFA3/LDV_G_EU6                  0.0
97         0.43         179.93  HBEFA3/LDV_G_EU6                  0.0
98         0.30           1.91  HBEFA3/LDV_G_EU6                  0.0
99         0.17         180.06  HBEFA3/LDV_G_EU6                  0.0

    vehicle_fuel vehicle_id     vehicle_lane  vehicle_noise  vehicle_pos  \
0           3.13     truck0      5329992#5_0          67.11         7.20
1           1.04       veh0      5330181#0_0          65.15         5.10
2           4.20     truck0      5329992#5_0          73.20         8.21
3           0.00       veh0      5330181#0_0          62.72        18.85
4           1.13       veh1     -5338968#2_0          55.94         5.10
..           …         …              …             …          …
95          1.11      moto2    -5341858#10_0          63.24        35.78
96          8.07      moto3  -342586098#36_0          81.67        30.96
97          2.99      moto4      5331636#0_0          72.45        11.88
98          1.85      moto5      5340657#0_0          71.73         5.60
99          0.53      moto6      5339596#0_0          55.94         2.30

    vehicle_route  vehicle_speed     vehicle_type  vehicle_waiting  vehicle_x  \
0   !truck0!var#1           0.00      truck_truck              0.0   18275.04
1     !veh0!var#1          14.72    veh_passenger              0.0   18279.94
2   !truck0!var#1           1.01      truck_truck              0.0   18275.82
3     !veh0!var#1          13.75    veh_passenger              0.0   18289.19
4     !veh1!var#1           0.00    veh_passenger              0.0   29252.01
..            …             …              …                …        …
95   !moto2!var#1          11.62  moto_motorcycle              0.0   26468.26
96   !moto3!var#1          13.99  moto_motorcycle              0.0   24729.15
97   !moto4!var#1           6.37  moto_motorcycle              0.0   29159.96
98   !moto5!var#1           3.30  moto_motorcycle              0.0   24340.58
99   !moto6!var#1           0.00  moto_motorcycle              0.0   26577.70

    vehicle_y
0    26987.78
1    24533.12
2    26988.43
3    24543.30
4    24424.16
```

```
..        …
95   25548.47
96   27450.68
97   25066.29
98   28198.87
99   25847.92

[100 rows x 20 columns]
```

	timestep_time	vehicle_CO	vehicle_CO2	vehicle_HC	vehicle_NOx	\
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	
mean	4.112561e+03	5.764304e+01	4.919050e+03	7.284125e-01	1.769589e+01	
std	2.168986e+03	8.854365e+01	7.959043e+03	1.589816e+00	5.993168e+01	
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	2.291000e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
50%	4.133000e+03	2.017000e+01	2.624720e+03	1.500000e-01	1.200000e+00	
75%	5.903000e+03	1.034400e+02	6.161010e+03	7.600000e-01	2.710000e+00	
max	1.441800e+04	3.932950e+03	1.153026e+05	1.729000e+01	8.864200e+02	

	vehicle_PMx	vehicle_angle	vehicle_electricity	vehicle_fuel	\
count	1.633101e+07	1.633101e+07	16331007.0	1.633101e+07	
mean	4.227491e-01	1.633698e+02	0.0	2.105266e+00	
std	1.164065e+00	1.051232e+02	0.0	3.389028e+00	
min	0.000000e+00	0.000000e+00	0.0	0.000000e+00	
25%	0.000000e+00	9.031000e+01	0.0	0.000000e+00	
50%	6.000000e-02	1.799600e+02	0.0	1.130000e+00	
75%	1.500000e-01	2.703500e+02	0.0	2.650000e+00	
max	1.432000e+01	3.600000e+02	0.0	4.888000e+01	

	vehicle_noise	vehicle_pos	vehicle_speed	vehicle_waiting	\
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	
mean	6.636207e+01	2.162082e+02	1.331140e+01	3.385107e+00	
std	7.389330e+00	6.034189e+02	8.833069e+00	1.914152e+01	
min	1.258000e+01	0.000000e+00	0.000000e+00	0.000000e+00	
25%	6.249000e+01	2.383000e+01	6.550000e+00	0.000000e+00	
50%	6.711000e+01	7.199000e+01	1.337000e+01	0.000000e+00	
75%	7.112000e+01	1.780600e+02	1.999000e+01	0.000000e+00	
max	1.019600e+02	1.943554e+04	5.013000e+01	3.970000e+02	

	vehicle_x	vehicle_y
count	1.633101e+07	1.633101e+07
mean	2.458506e+04	2.496505e+04
std	4.016049e+03	3.045771e+03
min	9.960000e+00	-1.490000e+00
25%	2.219207e+04	2.349907e+04
50%	2.393805e+04	2.548033e+04
75%	2.691704e+04	2.672322e+04
max	4.492832e+04	4.753314e+04

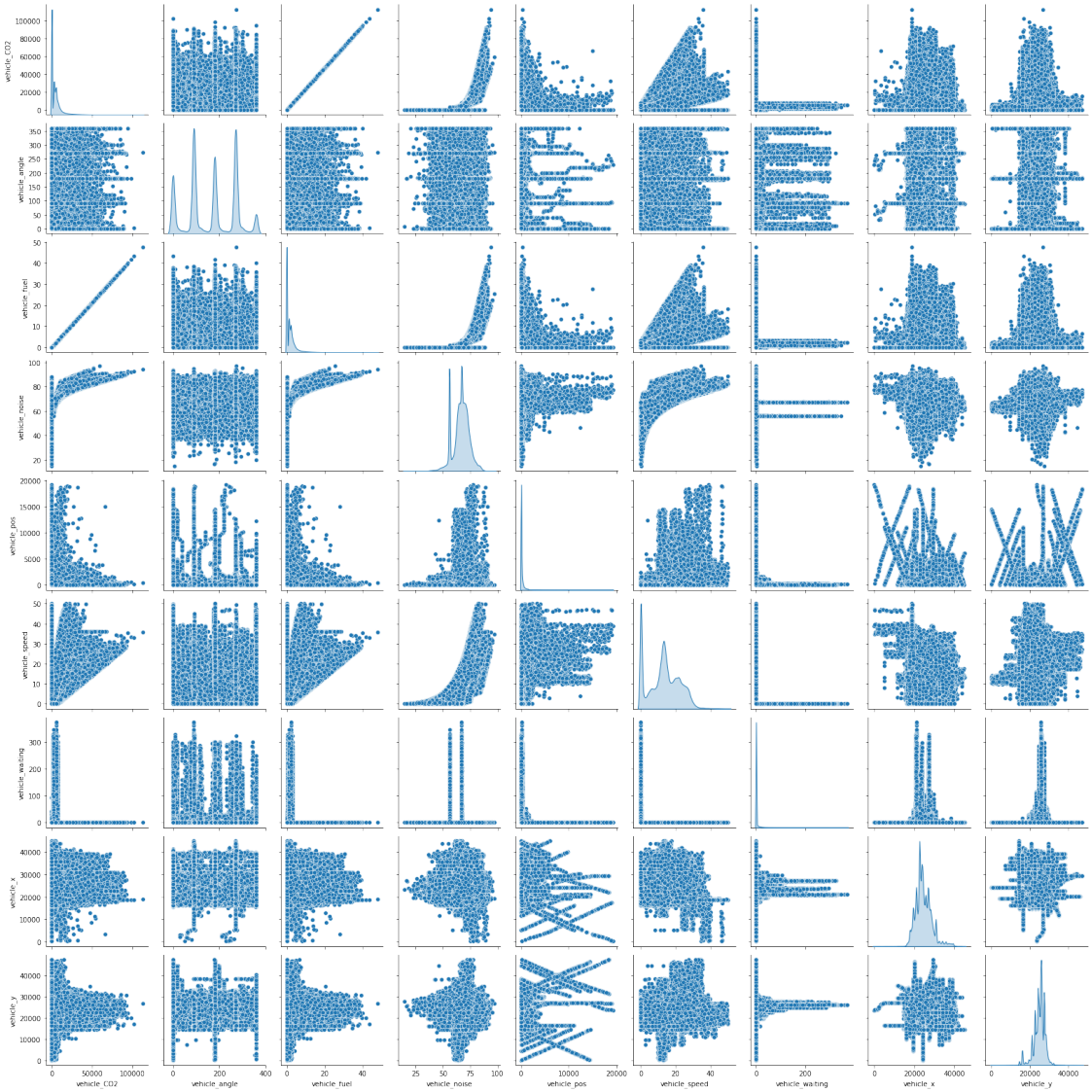
Vehicle_Speed x Vehicle_Noise looks interesting, in that the noise increases apparently quadratically with speed. I think this makes sense given kinetic energy is propto velocity squared.

Vehicle_Pos x Vehicle_Angle looks weird. I assume it is becuae there are a handful of trips that these cars were taking–for example if only one route had a left turn heading south you might see patterns like that.

Vehicle_x x Vehicle_Noise looks like a bunch of noise. That might suggest there is good dispersion of allowed speeds on the roads, and not like, say, a big highway going through campus.

[6]:
```
correlation_graph_data = emission_train.sample(frac=0.05).reset_index(drop=True)
print(len(emission_train), 'emission_train')
print(len(correlation_graph_data), 'correlation_graph_data')
sns.pairplot(correlation_graph_data[['vehicle_CO2', 'vehicle_angle',␣
 ↪'vehicle_fuel', 'vehicle_noise', 'vehicle_pos', 'vehicle_speed',␣
 ↪'vehicle_waiting', 'vehicle_x', 'vehicle_y']], diag_kind='kde')

#Free up memory for Python
del correlation_graph_data
```

```
16331008 emission_train
816550 correlation_graph_data
```



1.2.2 Clean up the Dataset

Note that there are emission data like `vehicle_CO`*,* `vehicle_CO2`*,* `vehicle_HC`*,* `vehicle_NOx`*,* `vehicle_PMx` *in the dataset. In this lab, we only want to look at* `vehicle_CO2`*.*

After looking at the data, you might notice there are a lot of data we don't want for our machine learning. For example, all the `vehicle_electricity` are zeros, and `vehicle_route` data are only used to keep track of the unique route each vehicle goes through.

Below, unwanted data are dropped. `vehicle_id` data are dropped because they are only used to keep track of different vehicles. `vehicle_lane` data are the name of the road. We dropped `vehicle_lane` data becuase we believed the data might not affect vehicle emissions. In practice, you should only drop the data if you have clear reasonings. For example `vehicle_electricity` are all zeros, so you can drop them. Even if you do not drop them, the machine learning program

might be able to figure the relationship out. `vehicle_route` data are dropped due to the reasoning above. `timestep_time` data are dropped becuase they are the simulation time.

To-do: 1. [D2]Drop the data we mentioned above. Also, drop the data that you think might not affect the machine learning. Q: Provide your reasonings.

Type your questions to Q:

I could make an argument to drop things like vehicle_x, vehicle_y, angle. These things describe the geographical position of the car and are unlikely themselves to affect emissions. However, I decided NOT to drop them for this exercise, because I believe these variables could be helpful from an explanatory persepctive. They are likely aliased with specific roads on campus, which in turn would be correlated with things like speed, traffic volume, which would then in turn be correlated with emissions.

I did drop vehicle_electricity because of the reasons mentioned in the text.

[5]:
```
emission_train = emission_train.drop(columns=["vehicle_CO", "vehicle_HC",␣
↪"vehicle_NOx", "vehicle_PMx",
                                              "timestep_time", "vehicle_id",␣
↪"vehicle_electricity"])
```

We seperated the block above from the block below becuase we don't want you to run `pd.read_csv` and `emission_train.drop()` twice. Reading a large csv file as you might have experienced a few minutes ago take up quite some RAM and CPU, and running `.drop()` twice will cause an error message to be printed out.

To-do: 1. [D3]Display the **last** 100 rows of your new `emission_train` data. It is okay if the displayed rows are truncated in the middle.

[6]:
```
display(emission_train.head(100))
display(emission_train.describe())

### Insert your code below ###
display(emission_train.tail(100))
```

	vehicle_CO2	vehicle_angle	vehicle_eclass	vehicle_fuel	\
0	7380.56	50.28	HBEFA3/HDV	3.13	
1	2416.04	42.25	HBEFA3/PC_G_EU4	1.04	
2	9898.93	50.28	HBEFA3/HDV	4.20	
3	0.00	42.25	HBEFA3/PC_G_EU4	0.00	
4	2624.72	357.00	HBEFA3/PC_G_EU4	1.13	
..	…	…	…	…	
95	2578.06	0.13	HBEFA3/LDV_G_EU6	1.11	
96	18759.70	179.93	HBEFA3/LDV_G_EU6	8.07	
97	6949.38	179.93	HBEFA3/LDV_G_EU6	2.99	
98	4292.19	1.91	HBEFA3/LDV_G_EU6	1.85	
99	1228.61	180.06	HBEFA3/LDV_G_EU6	0.53	

	vehicle_lane	vehicle_noise	vehicle_pos	vehicle_route	vehicle_speed	\
0	5329992#5_0	67.11	7.20	!truck0!var#1	0.00	

```
1       5330181#0_0          65.15         5.10    !veh0!var#1          14.72
2       5329992#5_0          73.20         8.21  !truck0!var#1           1.01
3       5330181#0_0          62.72        18.85    !veh0!var#1          13.75
4      -5338968#2_0          55.94         5.10    !veh1!var#1           0.00
..              …              …            …              …            …
95    -5341858#10_0          63.24        35.78   !moto2!var#1          11.62
96  -342586098#36_0          81.67        30.96   !moto3!var#1          13.99
97      5331636#0_0          72.45        11.88   !moto4!var#1           6.37
98      5340657#0_0          71.73         5.60   !moto5!var#1           3.30
99      5339596#0_0          55.94         2.30   !moto6!var#1           0.00

       vehicle_type  vehicle_waiting  vehicle_x  vehicle_y
0       truck_truck              0.0   18275.04   26987.78
1     veh_passenger              0.0   18279.94   24533.12
2       truck_truck              0.0   18275.82   26988.43
3     veh_passenger              0.0   18289.19   24543.30
4     veh_passenger              0.0   29252.01   24424.16
..              …              …          …          …
95  moto_motorcycle              0.0   26468.26   25548.47
96  moto_motorcycle              0.0   24729.15   27450.68
97  moto_motorcycle              0.0   29159.96   25066.29
98  moto_motorcycle              0.0   24340.58   28198.87
99  moto_motorcycle              0.0   26577.70   25847.92

[100 rows x 13 columns]

        vehicle_CO2  vehicle_angle  vehicle_fuel  vehicle_noise   vehicle_pos  \
count  1.633101e+07   1.633101e+07  1.633101e+07   1.633101e+07  1.633101e+07
mean   4.919050e+03   1.633698e+02  2.105266e+00   6.636207e+01  2.162082e+02
std    7.959043e+03   1.051232e+02  3.389028e+00   7.389330e+00  6.034189e+02
min    0.000000e+00   0.000000e+00  0.000000e+00   1.258000e+01  0.000000e+00
25%    0.000000e+00   9.031000e+01  0.000000e+00   6.249000e+01  2.383000e+01
50%    2.624720e+03   1.799600e+02  1.130000e+00   6.711000e+01  7.199000e+01
75%    6.161010e+03   2.703500e+02  2.650000e+00   7.112000e+01  1.780600e+02
max    1.153026e+05   3.600000e+02  4.888000e+01   1.019600e+02  1.943554e+04

       vehicle_speed  vehicle_waiting     vehicle_x     vehicle_y
count   1.633101e+07     1.633101e+07  1.633101e+07  1.633101e+07
mean    1.331140e+01     3.385107e+00  2.458506e+04  2.496505e+04
std     8.833069e+00     1.914152e+01  4.016049e+03  3.045771e+03
min     0.000000e+00     0.000000e+00  9.960000e+00 -1.490000e+00
25%     6.550000e+00     0.000000e+00  2.219207e+04  2.349907e+04
50%     1.337000e+01     0.000000e+00  2.393805e+04  2.548033e+04
75%     1.999000e+01     0.000000e+00  2.691704e+04  2.672322e+04
max     5.013000e+01     3.970000e+02  4.492832e+04  4.753314e+04

          vehicle_CO2  vehicle_angle vehicle_eclass  vehicle_fuel  \
16330908      5293.91           1.98     HBEFA3/Bus          2.26
16330909      6541.73           2.07     HBEFA3/Bus          2.79
```

```
16330910     10387.44           2.06     HBEFA3/Bus          4.43
16330911     12058.39           1.62     HBEFA3/Bus          5.14
16330912     13307.66           1.06     HBEFA3/Bus          5.67
…               …             …           …             …
16331003     19817.16           0.45     HBEFA3/Bus          8.45
16331004         0.00           0.45     HBEFA3/Bus          0.00
16331005     23192.37           0.45     HBEFA3/Bus          9.89
16331006         0.00           0.45     HBEFA3/Bus          0.00
16331007          NaN            NaN            NaN           NaN

            vehicle_lane  vehicle_noise  vehicle_pos vehicle_route  \
16330908  -737382716#5_0          67.19        77.83   pt_bus_5E:0
16330909   :38069188_4_0          71.21         0.69   pt_bus_5E:0
16330910   :38069188_4_0          74.53         2.58   pt_bus_5E:0
16330911   :38069188_4_0          73.88         5.45   pt_bus_5E:0
16330912   :38069188_4_0          73.64         9.19   pt_bus_5E:0
…                 …             …           …             …
16331003    -5334376#0_0          76.56       185.84   pt_bus_5E:0
16331004    -5334376#0_0          74.14       199.17   pt_bus_5E:0
16331005    -5334376#0_0          77.18       212.90   pt_bus_5E:0
16331006    -5334376#0_0          74.10       226.29   pt_bus_5E:0
16331007             NaN            NaN          NaN           NaN

          vehicle_speed vehicle_type  vehicle_waiting  vehicle_x  vehicle_y
16330908           0.01       pt_bus              1.0   30010.68   25278.35
16330909           0.70       pt_bus              0.0   30010.69   25279.04
16330910           1.88       pt_bus              0.0   30010.71   25280.93
16330911           2.87       pt_bus              0.0   30010.74   25283.80
16330912           3.74       pt_bus              0.0   30010.78   25287.54
…                 …             …               …         …          …
16331003          13.65       pt_bus              0.0   30107.48   25771.74
16331004          13.33       pt_bus              0.0   30107.59   25785.07
16331005          13.73       pt_bus              0.0   30107.70   25798.80
16331006          13.39       pt_bus              0.0   30107.80   25812.19
16331007            NaN          NaN              NaN        NaN        NaN

[100 rows x 13 columns]
```

By now, you would have already done some cleanups by dropping unwanted data. Below we used a `for` loop to cast the data in `vehicle_eclass` and `vehicle_type` to string. As you might notice that the values in both columns are texts. However, we found that the data in our csv file cannot be read correctly into Tensorflow so we added the for loop. * `.dropna().reset_index(drop=True)` drops the rows that contain NaN in any columns and reset the row index.

To-do: 1. ^{D4}Shuffle `emission_train` and save a new copy to `emission_train_shuffle`. *Hint: Look at the function we used to extract data for the correlation graph.* 2. ^{D5}Display the first 100 rows of the shuffled data. It is okay if the displayed rows are truncated in the middle. 3. ^{D6}Display the statistic (count, mean, std…) on the shuffled data. ^{D7}Q: Does anything change?

Type your answers to Q:

D6 – Nothing changes. This is a perfect shuffle without replacement, so as expected.

[7]:
```
for header in ["vehicle_eclass", "vehicle_type"]:
    emission_train[header] = emission_train[header].astype(str)

emission_train = emission_train.dropna().reset_index(drop=True)

# Shuffle the dataset
emission_train_shuffle = emission_train.sample(frac = 1).reset_index(drop=True)

### Insert your code below ###

# Display the data pre- and post- shuffle
display(emission_train.head(100))
display(emission_train_shuffle.head(100))
###FILL IN THE CODE

# Get info of the dataframe
display(emission_train.describe())
display(emission_train_shuffle.describe())
```

```
    vehicle_CO2  vehicle_angle    vehicle_eclass  vehicle_fuel  \
0       7380.56          50.28        HBEFA3/HDV          3.13
1       2416.04          42.25   HBEFA3/PC_G_EU4          1.04
2       9898.93          50.28        HBEFA3/HDV          4.20
3          0.00          42.25   HBEFA3/PC_G_EU4          0.00
4       2624.72         357.00   HBEFA3/PC_G_EU4          1.13
..          …              …               …           …
95      2578.06           0.13  HBEFA3/LDV_G_EU6          1.11
96     18759.70         179.93  HBEFA3/LDV_G_EU6          8.07
97      6949.38         179.93  HBEFA3/LDV_G_EU6          2.99
98      4292.19           1.91  HBEFA3/LDV_G_EU6          1.85
99      1228.61         180.06  HBEFA3/LDV_G_EU6          0.53

       vehicle_lane  vehicle_noise  vehicle_pos  vehicle_route  vehicle_speed  \
0       5329992#5_0          67.11         7.20  !truck0!var#1           0.00
1       5330181#0_0          65.15         5.10    !veh0!var#1          14.72
2       5329992#5_0          73.20         8.21  !truck0!var#1           1.01
3       5330181#0_0          62.72        18.85    !veh0!var#1          13.75
4      -5338968#2_0          55.94         5.10    !veh1!var#1           0.00
..              …              …            …              …              …
95    -5341858#10_0          63.24        35.78   !moto2!var#1          11.62
96  -342586098#36_0          81.67        30.96   !moto3!var#1          13.99
97      5331636#0_0          72.45        11.88   !moto4!var#1           6.37
98      5340657#0_0          71.73         5.60   !moto5!var#1           3.30
99      5339596#0_0          55.94         2.30   !moto6!var#1           0.00
```

```
       vehicle_type  vehicle_waiting  vehicle_x  vehicle_y
0       truck_truck              0.0   18275.04   26987.78
1     veh_passenger              0.0   18279.94   24533.12
2       truck_truck              0.0   18275.82   26988.43
3     veh_passenger              0.0   18289.19   24543.30
4     veh_passenger              0.0   29252.01   24424.16
..              …              …         …         …
95  moto_motorcycle              0.0   26468.26   25548.47
96  moto_motorcycle              0.0   24729.15   27450.68
97  moto_motorcycle              0.0   29159.96   25066.29
98  moto_motorcycle              0.0   24340.58   28198.87
99  moto_motorcycle              0.0   26577.70   25847.92

[100 rows x 13 columns]
```

```
    vehicle_CO2  vehicle_angle    vehicle_eclass  vehicle_fuel  \
0          0.00         269.99  HBEFA3/LDV_G_EU6          0.00
1       2624.72         179.98   HBEFA3/PC_G_EU4          1.13
2          0.00         180.59   HBEFA3/PC_G_EU4          0.00
3          0.00          91.21   HBEFA3/PC_G_EU4          0.00
4       3479.62          90.40   HBEFA3/PC_G_EU4          1.50
..          …              …               …            …
95         0.00         187.75   HBEFA3/PC_G_EU4          0.00
96         0.00          90.30   HBEFA3/PC_G_EU4          0.00
97      2894.79           9.58   HBEFA3/PC_G_EU4          1.24
98         0.00         270.29   HBEFA3/PC_G_EU4          0.00
99         0.00          90.42  HBEFA3/LDV_G_EU6          0.00

      vehicle_lane  vehicle_noise  vehicle_pos    vehicle_route  \
0    738660123#2_1          59.80        42.95  !moto6842!var#1
1   -737699690#2_0          55.94        92.75   !veh9455!var#1
2    234065269#0_0          62.26        26.13   !veh6414!var#1
3    742323290#6_0          70.14      1130.55   !veh6503!var#1
4   -256998523#1_0          65.55         0.66  !veh15033!var#1
..             …              …            …              …
95   741940360#7_0          65.47        93.71  !veh15765!var#1
96   245255393#3_0          65.64        42.25   !veh4509!var#1
97   35030181#10_0          60.92         0.79  !veh19107!var#1
98    -5341000#7_0          66.28      1063.31  !veh19648!var#1
99    -5337900#0_0          54.20       189.01  !moto1879!var#1

    vehicle_speed     vehicle_type  vehicle_waiting  vehicle_x  vehicle_y
0           11.69  moto_motorcycle              0.0   26206.37   26321.14
1            0.00    veh_passenger              3.0   25545.68   26333.23
2           13.15    veh_passenger              0.0   24288.05   25695.75
3           24.84    veh_passenger              0.0   22501.72   21101.93
4           14.25    veh_passenger              0.0   21763.59   25546.15
```

```
..            …              …                 …          …          …
95          16.54    veh_passenger               0.0   23731.99   24089.44
96          17.16    veh_passenger               0.0   24507.83   24970.23
97           6.11    veh_passenger               0.0   32412.25   29104.82
98          18.76    veh_passenger               0.0   31316.20   22653.87
99           0.17  moto_motorcycle               0.0   22652.29   25141.07

[100 rows x 13 columns]
```

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	\
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	
mean	4.919050e+03	1.633698e+02	2.105266e+00	6.636207e+01	2.162082e+02	
std	7.959043e+03	1.051232e+02	3.389028e+00	7.389330e+00	6.034189e+02	
min	0.000000e+00	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	
25%	0.000000e+00	9.031000e+01	0.000000e+00	6.249000e+01	2.383000e+01	
50%	2.624720e+03	1.799600e+02	1.130000e+00	6.711000e+01	7.199000e+01	
75%	6.161010e+03	2.703500e+02	2.650000e+00	7.112000e+01	1.780600e+02	
max	1.153026e+05	3.600000e+02	4.888000e+01	1.019600e+02	1.943554e+04	

	vehicle_speed	vehicle_waiting	vehicle_x	vehicle_y
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07
mean	1.331140e+01	3.385107e+00	2.458506e+04	2.496505e+04
std	8.833069e+00	1.914152e+01	4.016049e+03	3.045771e+03
min	0.000000e+00	0.000000e+00	9.960000e+00	-1.490000e+00
25%	6.550000e+00	0.000000e+00	2.219207e+04	2.349907e+04
50%	1.337000e+01	0.000000e+00	2.393805e+04	2.548033e+04
75%	1.999000e+01	0.000000e+00	2.691704e+04	2.672322e+04
max	5.013000e+01	3.970000e+02	4.492832e+04	4.753314e+04

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	\
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	
mean	4.919050e+03	1.633698e+02	2.105266e+00	6.636207e+01	2.162082e+02	
std	7.959043e+03	1.051232e+02	3.389028e+00	7.389330e+00	6.034189e+02	
min	0.000000e+00	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	
25%	0.000000e+00	9.031000e+01	0.000000e+00	6.249000e+01	2.383000e+01	
50%	2.624720e+03	1.799600e+02	1.130000e+00	6.711000e+01	7.199000e+01	
75%	6.161010e+03	2.703500e+02	2.650000e+00	7.112000e+01	1.780600e+02	
max	1.153026e+05	3.600000e+02	4.888000e+01	1.019600e+02	1.943554e+04	

	vehicle_speed	vehicle_waiting	vehicle_x	vehicle_y
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07
mean	1.331140e+01	3.385107e+00	2.458506e+04	2.496505e+04
std	8.833069e+00	1.914152e+01	4.016049e+03	3.045771e+03
min	0.000000e+00	0.000000e+00	9.960000e+00	-1.490000e+00
25%	6.550000e+00	0.000000e+00	2.219207e+04	2.349907e+04
50%	1.337000e+01	0.000000e+00	2.393805e+04	2.548033e+04
75%	1.999000e+01	0.000000e+00	2.691704e+04	2.672322e+04
max	5.013000e+01	3.970000e+02	4.492832e+04	4.753314e+04

1.3 *Stop*

Before you proceed, make sure you finish reading "Machine Learning Introduction" in Step 3 of the lab. You should complete the Tensorflow playground exercise and take a screenshot of your results.

Note–Screen shots of "Machine Learning Introduction" in report. Not reproduced here. -rdb4

1.4 Split Data for Machine Learning

In machine learning, we often want to split our data into Training Set, Validation Set, and Test Set. * **Training Set**: Training Set is used to train our machine learning model while the Validation and Test Set aren't. * **Validation Set**: Having a Validation Set prevents overfitting of our machine learning model. Overfitting is when our model is tuned perfectly for a specific set of data, but is fitted poorly for other set of data. Take our traffic emission data for example. If the data predicts CO_2 emission data within 10 mse (mean squared error) from Training Set, but predicts emission data over 50 mse from Validation data. Then we could see that the model is overfitted. * **Test Set**: Test set is used to evaluate the final model.

A typical workflow will be: 1. Train your model using *Training Set*. 2. Validate your model using *Validation Set*. 3. Adjust your model using results from *Validation Set*. 4. Pick the model that produces best results from using *Validation Set*. 5. Confirm your model with *Test Set*.

To-Do: 1. Don't change the `test_size=0.99` in the first split. 2. Tweak the `test_size=` values for spilitting `train_df`, `test_df`, and `val_df`. 3. You will come back and change some codes after you finish your first training. Instructions will be provided in the "Train the Model" section.

[8]:
```
#train_df, backup_df = train_test_split(emission_train_shuffle, test_size=0.99)
 ↪# Comment this line for large data training
# Edit the test_size below.

train_df, test_df = train_test_split(emission_train_shuffle, test_size=0.1) #
 ↪Uncomment for large dataset
train_df, test_df = train_test_split(train_df, test_size=0.1) # Comment for
 ↪large dataset
train_df, val_df = train_test_split(train_df, test_size=0.2)

#print(len(backup_df), 'backup data')
print(len(train_df), 'train examples')
print(len(val_df), 'validation examples')
print(len(test_df), 'test examples')

# Commented this out to make debugging easier.
#del emission_train
```

```
10582492 train examples
2645623 validation examples
1469791 test examples
```

1.5 Normalize the Input Data (Optional)

Sometimes when there are huge value differences between input features, we want to scale them to get a better training result. In this lab you are not required to use normalization. But if you cannot get a nice machine learning result, you can try normalizing the data. Below, we used Z normalization. It is just a normalization method. If you normalize your trainning data, make sure to also **normalize the validation and test data**. Note that `train_df_norm = train_df` won't copy `train_df` to `train_df_norm`. Changing the values in `train_df_norm` will affect the values in `train_df`. So if you decide to revert the normalization after you run the code block below, run the code block under "Split Data for Machine Learning" again and run only the `train_df_norm = train_df` below. (Comment out the code using `#` sign.)

Z Normalization Equation:

$$z = \frac{x - \mu}{\sigma} z : \text{Normalized Data} x : \text{Original Data} \mu : \text{Mean of } x \sigma : \text{Standard Deviation of } x$$

[9]:
```
# # Z-Score Normalizing
train_df_norm = train_df
test_df_norm = test_df
valid_df_norm = val_df

### Insert your code below (optional) ###
# Normalize the validation data
# Normalize the test data

# Note I am subtracting the train mean and stddev from the test and validation␣
 ↪datasets
# to ensure that the normalization is the same.  Best practice might suggest to␣
 ↪do this
# to the entire dataset before splitting.

for header in ["vehicle_angle", "vehicle_fuel", "vehicle_noise", "vehicle_pos",␣
 ↪"vehicle_speed", "vehicle_waiting", "vehicle_x", "vehicle_y"]:

    mu = train_df[header].mean()
    std = train_df[header].std()
    print("For variable {} the normalizing constants are mu {} std {}".
 ↪format(header, mu, std))

    train_df_norm[header] = (train_df[header] - mu) / std
    test_df_norm[header] = (test_df_norm[header] - mu) / std
    valid_df_norm[header] = (valid_df_norm[header] - mu) / std

    train_df_norm[header] = train_df_norm[header].fillna(0)
    test_df_norm[header] = test_df_norm[header].fillna(0)
    valid_df_norm[header] = valid_df_norm[header].fillna(0)

display(train_df_norm.describe())
```

```
For variable vehicle_angle the normalizing constants are mu 163.37757630433362
std 105.12412100972757
For variable vehicle_fuel the normalizing constants are mu 2.104949905938977 std
3.389260801093057
For variable vehicle_noise the normalizing constants are mu 66.36261798449742
std 7.387811354100563
For variable vehicle_pos the normalizing constants are mu 216.21983845203948 std
603.6075449609833
For variable vehicle_speed the normalizing constants are mu 13.309540455121525
std 8.833066574543095
For variable vehicle_waiting the normalizing constants are mu 3.3925428906537327
std 19.170472861628465
For variable vehicle_x the normalizing constants are mu 24584.475173528124 std
4016.505593817021
For variable vehicle_y the normalizing constants are mu 24964.97717733594 std
3046.446868311109
```

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	\
count	1.058249e+07	1.058249e+07	1.058249e+07	1.058249e+07	1.058249e+07	
mean	4.918321e+03	-4.696770e-17	5.932172e-16	-7.869382e-16	-3.553556e-17	
std	7.959589e+03	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	
min	0.000000e+00	-1.554140e+00	-6.210646e-01	-7.279912e+00	-3.582126e-01	
25%	0.000000e+00	-6.950600e-01	-6.210646e-01	-5.241902e-01	-3.187333e-01	
50%	2.624720e+03	1.577414e-01	-2.876586e-01	1.011642e-01	-2.389464e-01	
75%	6.159812e+03	1.017582e+00	1.608168e-01	6.439501e-01	-6.316992e-02	
max	1.126353e+05	1.870384e+00	1.346755e+01	4.818393e+00	3.182536e+01	

	vehicle_speed	vehicle_waiting	vehicle_x	vehicle_y
count	1.058249e+07	1.058249e+07	1.058249e+07	1.058249e+07
mean	1.145251e-15	-2.453850e-17	3.051687e-16	-4.798907e-15
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.506786e+00	-1.769671e-01	-6.116759e+00	-8.195271e+00
25%	-7.652541e-01	-1.769671e-01	-5.967364e-01	-4.813467e-01
50%	6.844683e-03	-1.769671e-01	-1.610144e-01	1.692473e-01
75%	7.563013e-01	-1.769671e-01	5.810212e-01	5.770535e-01
max	4.168480e+00	2.053196e+01	5.065061e+00	7.408028e+00

1.6 Organize Features

1.6.1 Classify Features

We need to define our feature columns so that the program knows what type of features are used in the training. In emission data, there are two types of features: numeric (floating point, int, etc.) and categorical/indicator (for example, 'color', 'gender'; 'color' column can contain 'red', 'blue', etc.).

To Do: 1. ^{M1}Organize the numeric columns. Also fill in the numeric columns' names in your dataset. Remember that you dropped some values already. Only put the names of the columns that are still in your dataset. Refer to "Classify structured data with

feature columns" under "Tensorflow Tutorials" section on the Tensorflow website. Link: https://www.tensorflow.org/tutorials/structured_data/feature_columns

[10]:
```
# Create an empty list
feature_cols = []


# Numeric Columns
for header in ["vehicle_angle", "vehicle_fuel", "vehicle_speed",␣
↪"vehicle_waiting", "vehicle_noise", "vehicle_pos", "vehicle_x", "vehicle_y"␣
↪]:
    ### Insert your code ###
    col = tf.feature_column.numeric_column(header)
    feature_cols.append(col)

# Indicator Columns
indicator_col_names = ["vehicle_eclass", "vehicle_type"]
for col_name in indicator_col_names:
    categorical_column = tf.feature_column.
↪categorical_column_with_vocabulary_list(col_name,
                                                                        ␣
↪    train_df[col_name].unique())
    indicator_column = tf.feature_column.indicator_column(categorical_column)
    feature_cols.append(indicator_column)

print("Feature columns: ", feature_cols, "\n")
```

```
Feature columns:  [NumericColumn(key='vehicle_angle', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None),
NumericColumn(key='vehicle_fuel', shape=(1,), default_value=None,
dtype=tf.float32, normalizer_fn=None), NumericColumn(key='vehicle_speed',
shape=(1,), default_value=None, dtype=tf.float32, normalizer_fn=None),
NumericColumn(key='vehicle_waiting', shape=(1,), default_value=None,
dtype=tf.float32, normalizer_fn=None), NumericColumn(key='vehicle_noise',
shape=(1,), default_value=None, dtype=tf.float32, normalizer_fn=None),
NumericColumn(key='vehicle_pos', shape=(1,), default_value=None,
dtype=tf.float32, normalizer_fn=None), NumericColumn(key='vehicle_x',
shape=(1,), default_value=None, dtype=tf.float32, normalizer_fn=None),
NumericColumn(key='vehicle_y', shape=(1,), default_value=None, dtype=tf.float32,
normalizer_fn=None), IndicatorColumn(categorical_column=VocabularyListCategorica
lColumn(key='vehicle_eclass', vocabulary_list=('HBEFA3/PC_G_EU4', 'HBEFA3/HDV',
'HBEFA3/LDV_G_EU6', 'HBEFA3/Bus'), dtype=tf.string, default_value=-1,
num_oov_buckets=0)), IndicatorColumn(categorical_column=VocabularyListCategorica
lColumn(key='vehicle_type', vocabulary_list=('veh_passenger', 'truck_truck',
'moto_motorcycle', 'pt_bus', 'bus_bus'), dtype=tf.string, default_value=-1,
num_oov_buckets=0))]
```

1.6.2 Create a Feature Layer

Feature layer will the input to our machine learning. We need to create a feature layer to be added into the machine learning model.

[11]:
```
# Create a feature layer for tf
feature_layer = tf.keras.layers.DenseFeatures(feature_cols, name='Features')
```

1.7 Create and Train the Model

1.7.1 Create Model

- `model.add()`: add layer to model
- In `tf.keras.layers.Dense()`
  - `units`: number of nodes in that layer
  - `activation`: activation function used in that layer
  - `kernel_regularizer`: regularization function used in that layer
  - `name`: is just for us to keep track and debug
- In `model.compile()`
  - `optimizer=tf.keras.optimizers.Adam(lr=learning_rate)`: Used to improve performance of the training
  - `Adam`: stochastic gradient descent method
  - `loss`: update the model according to specified loss function
  - `metrics`: evaluate the model according specified metrics

1.7.2 Train the Model

- We first split our Pandas dataframe into features and labels.
- Then `model.fit()` trains our model.
- `logdir`, `tensorboard_callback` is to save training logs to be used in Tensorboard.
- Notice that there are 2 `model.fit()` function calls with one being commented out. The one without `callbacks=[tensorboard_callback]` is used in this program for large dataset training.

1.7.3 Instructions for Training Small and Large Data

As we mentioned in the lab document, hyperparameters affect the performance of your model. In the following blocks, you would be training your model. We also want you to experience training both a small dataset and a large dataset.

To-do: * **Small Dataset:** 1. The program cells you ran until now prepare you for small dataset training. You don't need to adjust the `test_size=0.99` in "Split Data for Machine Learning".

```
2. Adjust the Hyperparameters (learning rate, batch size, epochs, hidden layer number, node nu

3. In the function definitions (previous code block):
    * Press the stop button (**interrupt the kernal**) next to Run before you change the value
    * Add or reduce Hidden layers if your model turns our poorly.
    * Adjust the amount of nodes in each Hidden layer.
    * Try out different activation functions.
    * Try different regularizers.
    * You should aim to get an **mse < 100**. **Note, we will grade your results based on mse.*

4. $^{M2}$Once you get a result with nice mse, run the block `%tensorboard --logdir logs`. The
```

- **Large Dataset:**
    1. Adjust the codes in "Split Data for Machine Learning" so that no data go to `backup_df`.
    2. Go to previous code block and use the `model.fit()` without `callbacks=[tensorboard_callback]`. Remember to comment out the one with `callbacks=[tensorboard_callback]`.
    3. Adjust the Hyperparameters (learning rate, batch size, epochs, hidden layer number, node number). Remember, a large learning rate might cause the model to never converge, but a very small learning rate would cause the model to converge very slow. If your mse (mean squared error) is decreasing but your program finishes before the mse reaches a small number, increase your epochs. Smaller batch size often gives a better training result. A large batch size often causes poor convergence, and it might also lead to poor generalization and slow training speed. Try batch sizes of 1000, 10000, 200000. ^{M3}Q: Do you notice any difference between using batch sizes of 1000, 10000, 200000?
    4. In the function definitions:
        - Press the stop button (**interrupt the kernal**) next to Run before you change the values in the functions above.
        - Add or reduce Hidden layers if your model turns our poorly.
        - Adjust the amount of nodes in each Hidden layer.
        - Try out different activation functions.
        - Try different regularizers.
        - You should aim to get an **mse < 200**. **Note, we will grade your results based on mse.**
    5. ^{M4}The program will run for a longer time with large dataset input. Once you get a result with nice mse, you don't have to run `%tensorboard --logdir logs`. Move on to sections below. We would have you save a PDF once you reach the end of this Notebook. We will look at your training for the large dataset based on the logs printed out during each epoch.

Note: Ignore the warnings at the beginning and at the end.

Type your answers to Q:

M2 - Screen shots for small dataset in run.
M3 - I ran full model run with batch size of 200k. I eventually got valid_MSE of 48, and there was still signs model wasn't fully converged by epoch 200.

Not changing anything else, I later ran model with batch size of 10000. Model converges much faster, in less than 10 epochs, but runs very slowly. When terminated early, the valid MSE was reported to be 48.9258

Finally, I ran model with batch size of 1000. Training runs glacially slow–over an hour per epoch. I let it run overnight and stopped it at the 10th epoch, where it reported a valid MSE of 35.

[12]:
```
# Hyperparameters
learning_rate = 0.01 ### FILL IN A NUMBER
epochs = 50 ### FILL IN A NUMBER
batch_size = 200000 ### FILL IN A NUMBER

# Label
label_name = "vehicle_CO2"
shuffle = True

#---Create a sequential model---#
model = tf.keras.models.Sequential([
    # Add the feature layer
    feature_layer,

    # First hidden layer with 20 nodes
    tf.keras.layers.Dense(units=20,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=0.1),
                          name='Hidden1'),

    # First hidden layer with 20 nodes
    tf.keras.layers.Dense(units=20,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=0.1),
                          name='Hidden3'),

    # Output layer
    tf.keras.layers.Dense(units=1,
                          activation='linear',
                          name='Output')

])


model.compile(optimizer=tf.keras.optimizers.Adam(lr=learning_rate),
              loss=tf.keras.losses.MeanSquaredError(),
              metrics=['mse'])


#---Train the Model---#
# Keras TensorBoard callback.
```

```
logdir = "logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

train_lbl = np.array(train_df_norm["vehicle_CO2"])
#train_df = train_df.drop(columns=["vehicle_CO2"])
# Split the datasets into features and label.
train_ft = {name:np.array(value) for name, value in train_df_norm.items()}
# train_lbl = np.array(train_ft.pop(label_name))

val_lbl =  np.array(valid_df_norm["vehicle_CO2"])
#val_df = val_df.drop(columns=["vehicle_CO2"])
val_ft = {name:np.array(value) for name, value in valid_df_norm.items()}


# Keras TensorBoard callback.
logdir = "logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

model.fit(x=train_ft, y=train_lbl, batch_size=batch_size,
          epochs=epochs, callbacks=[tensorboard_callback],␣
 ↪validation_data=(val_ft, val_lbl), shuffle=shuffle)

# Training function for large training set
#model.fit(x=train_ft, y=train_lbl, batch_size=batch_size,
#           epochs=epochs, verbose=2, validation_data=(val_ft, val_lbl),␣
 ↪shuffle=shuffle)
```

```
Epoch 1/50
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'ExpandDims_1:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'ExpandDims_2:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'ExpandDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'ExpandDims_4:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'ExpandDims_5:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'ExpandDims_6:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'ExpandDims_7:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'ExpandDims_8:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'ExpandDims_9:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'ExpandDims_10:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'ExpandDims_11:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'ExpandDims_12:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
```

```
'ExpandDims_1:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'ExpandDims_2:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'ExpandDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'ExpandDims_4:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'ExpandDims_5:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'ExpandDims_6:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'ExpandDims_7:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'ExpandDims_8:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'ExpandDims_9:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'ExpandDims_10:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'ExpandDims_11:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'ExpandDims_12:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
53/53 [==============================] - ETA: 0s - loss: 87324493.4340 - mse:
87324475.4717WARNING:tensorflow:Layers in a Sequential model should only have a
single input tensor, but we receive a <class 'dict'> input: {'vehicle_CO2':
<tf.Tensor 'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_angle':
<tf.Tensor 'ExpandDims_1:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass':
<tf.Tensor 'ExpandDims_2:0' shape=(None, 1) dtype=string>, 'vehicle_fuel':
<tf.Tensor 'ExpandDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_lane':
<tf.Tensor 'ExpandDims_4:0' shape=(None, 1) dtype=string>, 'vehicle_noise':
<tf.Tensor 'ExpandDims_5:0' shape=(None, 1) dtype=float32>, 'vehicle_pos':
<tf.Tensor 'ExpandDims_6:0' shape=(None, 1) dtype=float32>, 'vehicle_route':
<tf.Tensor 'ExpandDims_7:0' shape=(None, 1) dtype=string>, 'vehicle_speed':
<tf.Tensor 'ExpandDims_8:0' shape=(None, 1) dtype=float32>, 'vehicle_type':
<tf.Tensor 'ExpandDims_9:0' shape=(None, 1) dtype=string>, 'vehicle_waiting':
<tf.Tensor 'ExpandDims_10:0' shape=(None, 1) dtype=float32>, 'vehicle_x':
<tf.Tensor 'ExpandDims_11:0' shape=(None, 1) dtype=float32>, 'vehicle_y':
<tf.Tensor 'ExpandDims_12:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
53/53 [==============================] - 16s 289ms/step - loss: 87308850.2222 -
mse: 87308832.1481 - val_loss: 82505912.0000 - val_mse: 82505880.0000
Epoch 2/50
53/53 [==============================] - 11s 220ms/step - loss: 74812528.2222 -
mse: 74812472.3704 - val_loss: 40586880.0000 - val_mse: 40586784.0000
Epoch 3/50
53/53 [==============================] - 13s 247ms/step - loss: 31576300.2963 -
mse: 31576200.7778 - val_loss: 12943863.0000 - val_mse: 12943754.0000
Epoch 4/50
53/53 [==============================] - 13s 247ms/step - loss: 10520327.5556 -
mse: 10520218.2963 - val_loss: 6114198.0000 - val_mse: 6114092.0000
Epoch 5/50
53/53 [==============================] - 13s 247ms/step - loss: 5266579.1296 -
mse: 5266473.9630 - val_loss: 3098683.7500 - val_mse: 3098580.5000
Epoch 6/50
53/53 [==============================] - 13s 246ms/step - loss: 2597767.9213 -
mse: 2597665.7407 - val_loss: 1408794.0000 - val_mse: 1408693.6250
Epoch 7/50
```

```
53/53 [==============================] - 13s 249ms/step - loss: 1180940.8032 -
mse: 1180840.6331 - val_loss: 681938.9375 - val_mse: 681839.6250
Epoch 8/50
53/53 [==============================] - 13s 249ms/step - loss: 599307.9375 -
mse: 599208.8310 - val_loss: 423453.8438 - val_mse: 423355.1562
Epoch 9/50
53/53 [==============================] - 13s 248ms/step - loss: 389497.8466 -
mse: 389399.2459 - val_loss: 298922.7188 - val_mse: 298824.0938
Epoch 10/50
53/53 [==============================] - 13s 248ms/step - loss: 275301.2552 -
mse: 275202.5833 - val_loss: 214468.4531 - val_mse: 214369.6406
Epoch 11/50
53/53 [==============================] - 13s 246ms/step - loss: 202555.5958 -
mse: 202456.6846 - val_loss: 170087.4375 - val_mse: 169988.2656
Epoch 12/50
53/53 [==============================] - 13s 248ms/step - loss: 160603.1788 -
mse: 160503.8906 - val_loss: 132593.4062 - val_mse: 132493.8594
Epoch 13/50
53/53 [==============================] - 13s 247ms/step - loss: 125742.8573 -
mse: 125643.2134 - val_loss: 107198.6953 - val_mse: 107098.8438
Epoch 14/50
53/53 [==============================] - 13s 249ms/step - loss: 102510.8720 -
mse: 102410.9751 - val_loss: 88654.2734 - val_mse: 88554.2500
Epoch 15/50
53/53 [==============================] - 13s 251ms/step - loss: 84614.5395 -
mse: 84514.4900 - val_loss: 73104.3047 - val_mse: 73004.2109
Epoch 16/50
53/53 [==============================] - 13s 248ms/step - loss: 70161.0909 -
mse: 70060.9900 - val_loss: 62137.9180 - val_mse: 62037.8008
Epoch 17/50
53/53 [==============================] - 13s 248ms/step - loss: 59936.3827 -
mse: 59836.2456 - val_loss: 53681.7891 - val_mse: 53581.6055
Epoch 18/50
53/53 [==============================] - 13s 251ms/step - loss: 51867.9411 -
mse: 51767.7436 - val_loss: 46592.5430 - val_mse: 46492.2656
Epoch 19/50
53/53 [==============================] - 14s 264ms/step - loss: 44939.7113 -
mse: 44839.4044 - val_loss: 40338.7305 - val_mse: 40238.3398
Epoch 20/50
53/53 [==============================] - 14s 262ms/step - loss: 38772.9746 -
mse: 38672.2508 - val_loss: 33388.8008 - val_mse: 33286.3477
Epoch 21/50
53/53 [==============================] - 13s 254ms/step - loss: 31426.7190 -
mse: 31323.5967 - val_loss: 25450.8242 - val_mse: 25345.7520
Epoch 22/50
53/53 [==============================] - 13s 254ms/step - loss: 23616.8819 -
mse: 23511.2489 - val_loss: 18445.2773 - val_mse: 18338.2617
Epoch 23/50
```

```
53/53 [==============================] - 13s 252ms/step - loss: 16913.1130 -
mse: 16805.6795 - val_loss: 12556.3398 - val_mse: 12447.6113
Epoch 24/50
53/53 [==============================] - 13s 256ms/step - loss: 11309.1596 -
mse: 11199.9994 - val_loss: 7977.0767 - val_mse: 7866.6445
Epoch 25/50
53/53 [==============================] - 15s 282ms/step - loss: 7085.8878 - mse:
6975.0710 - val_loss: 4761.7622 - val_mse: 4649.8560
Epoch 26/50
53/53 [==============================] - 15s 283ms/step - loss: 4190.3076 - mse:
4078.0734 - val_loss: 2824.0562 - val_mse: 2710.9126
Epoch 27/50
53/53 [==============================] - 14s 262ms/step - loss: 2528.2945 - mse:
2414.9048 - val_loss: 1854.5094 - val_mse: 1740.4944
Epoch 28/50
53/53 [==============================] - 13s 256ms/step - loss: 1711.7513 - mse:
1597.5798 - val_loss: 1393.4979 - val_mse: 1278.9240
Epoch 29/50
53/53 [==============================] - 13s 253ms/step - loss: 1321.1102 - mse:
1206.4350 - val_loss: 1157.9301 - val_mse: 1043.0011
Epoch 30/50
53/53 [==============================] - 13s 253ms/step - loss: 1120.2914 - mse:
1005.3026 - val_loss: 1022.1488 - val_mse: 907.0037
Epoch 31/50
53/53 [==============================] - 13s 253ms/step - loss: 995.6774 - mse:
880.4911 - val_loss: 929.3960 - val_mse: 814.0994
Epoch 32/50
53/53 [==============================] - 13s 254ms/step - loss: 907.7142 - mse:
792.3859 - val_loss: 860.4371 - val_mse: 745.0278
Epoch 33/50
53/53 [==============================] - 13s 254ms/step - loss: 844.9407 - mse:
729.5102 - val_loss: 807.0644 - val_mse: 691.5776
Epoch 34/50
53/53 [==============================] - 13s 253ms/step - loss: 795.5613 - mse:
680.0591 - val_loss: 762.9628 - val_mse: 647.4185
Epoch 35/50
53/53 [==============================] - 13s 256ms/step - loss: 752.3882 - mse:
636.8322 - val_loss: 727.3923 - val_mse: 611.8051
Epoch 36/50
53/53 [==============================] - 13s 254ms/step - loss: 718.5864 - mse:
602.9905 - val_loss: 697.5903 - val_mse: 581.9691
Epoch 37/50
53/53 [==============================] - 13s 255ms/step - loss: 692.3597 - mse:
576.7310 - val_loss: 671.0828 - val_mse: 555.4346
Epoch 38/50
53/53 [==============================] - 13s 255ms/step - loss: 666.4672 - mse:
550.8136 - val_loss: 644.4548 - val_mse: 528.7877
Epoch 39/50
```

```
53/53 [==============================] - 13s 252ms/step - loss: 637.5691 - mse:
521.8989 - val_loss: 621.4003 - val_mse: 505.7236
Epoch 40/50
53/53 [==============================] - 13s 253ms/step - loss: 617.4844 - mse:
501.8070 - val_loss: 605.7040 - val_mse: 490.0274
Epoch 41/50
53/53 [==============================] - 13s 254ms/step - loss: 600.9585 - mse:
485.2831 - val_loss: 592.2870 - val_mse: 476.6194
Epoch 42/50
53/53 [==============================] - 13s 254ms/step - loss: 590.1871 - mse:
474.5226 - val_loss: 581.7750 - val_mse: 466.1237
Epoch 43/50
53/53 [==============================] - 13s 253ms/step - loss: 579.7219 - mse:
464.0760 - val_loss: 572.0312 - val_mse: 456.4015
Epoch 44/50
53/53 [==============================] - 13s 254ms/step - loss: 570.2881 - mse:
454.6645 - val_loss: 563.5604 - val_mse: 447.9557
Epoch 45/50
53/53 [==============================] - 14s 262ms/step - loss: 560.9060 - mse:
445.3081 - val_loss: 555.8336 - val_mse: 440.2568
Epoch 46/50
53/53 [==============================] - 14s 262ms/step - loss: 554.5244 - mse:
438.9557 - val_loss: 545.8380 - val_mse: 430.2927
Epoch 47/50
53/53 [==============================] - 14s 262ms/step - loss: 543.2735 - mse:
427.7371 - val_loss: 533.7949 - val_mse: 418.2845
Epoch 48/50
53/53 [==============================] - 14s 263ms/step - loss: 528.4394 - mse:
412.9382 - val_loss: 517.0682 - val_mse: 401.5942
Epoch 49/50
53/53 [==============================] - 14s 260ms/step - loss: 511.9860 - mse:
396.5207 - val_loss: 500.6501 - val_mse: 385.2102
Epoch 50/50
53/53 [==============================] - 13s 256ms/step - loss: 497.3003 - mse:
381.8677 - val_loss: 489.1540 - val_mse: 373.7435
```

[12]:
```
<tensorflow.python.keras.callbacks.History at 0x7f554acb2520>
```

1.7.4 Evaluate the Model with Test Data

Below you will evaluate the performance of your model using the test data.

[246]:
```
test_lbl = np.array(test_df_norm["vehicle_CO2"])
#test_df = test_df.drop(columns=["vehicle_CO2"])
test_ft = {key:np.array(value) for key, value in test_df_norm.items()}
# test_lbl = np.array(test_ft.pop(label_name))
print("Model evaluation: \n")
model.evaluate(x=test_ft, y=test_lbl, batch_size=batch_size)
```

```
Model evaluation:

8/8 [==============================] - 0s 33ms/step - loss: 165.3179 - mse:
48.9905
```

[246]:
```
[165.31785583496094, 48.99047088623047]
```

[247]:
```
#Get a summary of your model
model.summary()
```

```
Model: "sequential_111"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Features (DenseFeatures)     multiple                  0
_________________________________________________________________
Hidden1 (Dense)              multiple                  360
_________________________________________________________________
Hidden3 (Dense)              multiple                  420
_________________________________________________________________
Output (Dense)               multiple                  21
=================================================================
Total params: 801
Trainable params: 801
Non-trainable params: 0
_________________________________________________________________
```

1.8 Use the Trained-Model and Visualize the results

Below we provide you with tables and figures for you to visualize your training results.

1.8.1 TensorBoard

From TensorBoard, you can see the loss and mse curve of your training. Go to graph and under "Tag", select "keras". You can see your network. Note that you will see error under "Tag: Default". You can ignore the warning.

[248]:
```
%tensorboard --logdir logs
```

```
<IPython.core.display.HTML object>
```

1.8.2 Predict CO_2 From Trained-Model

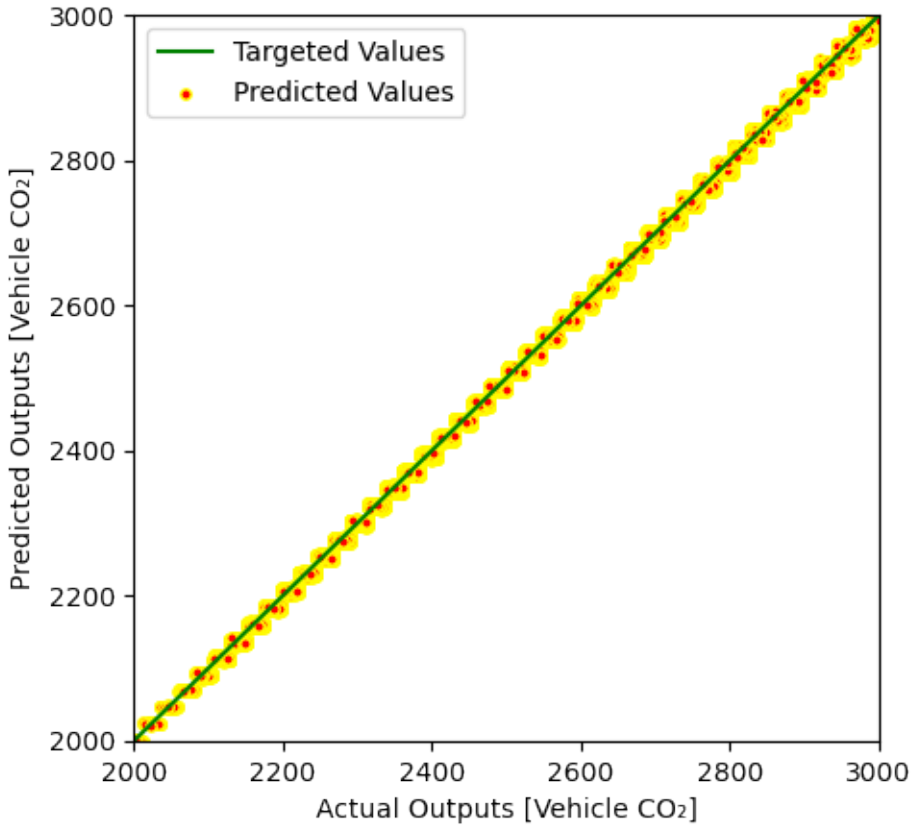
Below, your trained-model is used to make prediction on the test set. Remember, test set is not used in training the model so it would give you a nice indication of how your model is doing. * `.predict()`: predicts the output values from features given.

- `predicted_labels`: contains the values (CO_2) our model predicts. After the predicted and actual values are obtained. We create a plot for you to visualize the results. The dots show the predicted values and the line shows the targeted values.

[254]:
```
%%time

# Get the features from the test set
test_features = test_ft
# Get the actual CO2 output for the test set
actual_labels = test_lbl

# Make prediction on the test set
# NB I added batch_size which significantly sped this up from over 2 hrs to a␣
 ↪few seconds.
# My guess is that there was some O(n^2) list stuff going on otherwise.
predicted_labels = model.predict(x=test_features, batch_size=len(test_lbl)).
 ↪flatten()

# Define the graph
Figure1 = plt.figure(figsize=(5,5), dpi=100)
plt.xlabel('Actual Outputs [Vehicle CO\u2082]')
plt.ylabel('Predicted Outputs [Vehicle CO\u2082]')
plt.scatter(actual_labels, predicted_labels, s=15, c='Red',␣
 ↪edgecolors='Yellow', label='Predicted Values')

# Take the output data from 2000 to 3000 as an instance to visualize
lims = [2000, 3000]
plt.xlim(lims)
plt.ylim(lims)
plt.plot(lims, lims, color='Green', label='Targeted Values')
plt.legend()
```

```
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'ExpandDims:0' shape=(1469791, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'ExpandDims_1:0' shape=(1469791, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'ExpandDims_2:0' shape=(1469791, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'ExpandDims_3:0' shape=(1469791, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'ExpandDims_4:0' shape=(1469791, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'ExpandDims_5:0' shape=(1469791, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'ExpandDims_6:0' shape=(1469791, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'ExpandDims_7:0' shape=(1469791, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'ExpandDims_8:0' shape=(1469791, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'ExpandDims_9:0' shape=(1469791, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'ExpandDims_10:0' shape=(1469791, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'ExpandDims_11:0' shape=(1469791, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'ExpandDims_12:0' shape=(1469791, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
CPU times: user 1.36 s, sys: 1.31 s, total: 2.67 s
Wall time: 1.34 s
```

[254]: <matplotlib.legend.Legend at 0x7f8c64715be0>

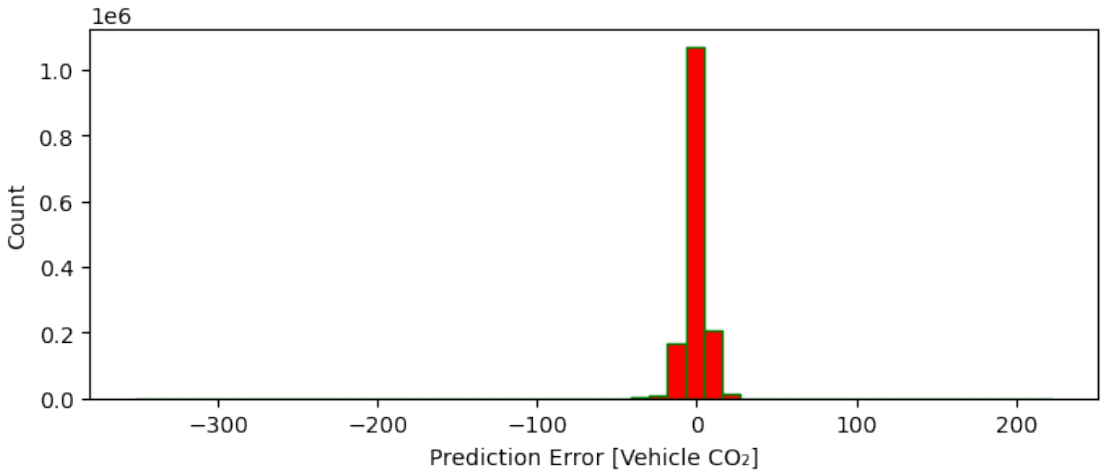


1.8.3 Error Count Histogram

Below, the graph shows a Histogram of errors between predicted and actual values. If the error counts locate mostly around 0, the trained-model is pretty accurate.

[255]:
```
error = actual_labels - predicted_labels
Figure2 = plt.figure(figsize=(8,3), dpi=100)
plt.hist(error, bins=50, color='Red', edgecolor='Green')
plt.xlabel('Prediction Error [Vehicle CO\u2082]')
plt.ylabel('Count')
```

[255]: Text(0, 0.5, 'Count')



1.8.4 Table of Actual and Predicted Values

Below, a table puts the actual and predicted values side by side. Html is used in this case.

[256]:
```
from IPython.display import HTML, display

def display_table(data_x, data_y):
    html = "<table>"
    html += "<tr>"
    html += "<td><h3>%s</h3><td>"%"Actual Vehicle CO\u2082"
    html += "<td><h3>%s</h3><td>"%"Predicted Vehicle CO\u2082"
    html += "</tr>"
    for i in range(len(data_x)):
        html += "<tr>"
        html += "<td><h4>%s</h4><td>"%(int(data_x[i]))
        html += "<td><h4>%s</h4><td>"%(int(data_y[i]))
        html += "</tr>"
    html += "</table>"
    display(HTML(html))

display_table(actual_labels[0:100], predicted_labels[0:100])
```

```
<IPython.core.display.HTML object>
```

1.9 Well Done!

Congradulation on finishing the lab. Please click on "File -> Print Preview" and a separate page should open. Press Cmd/Ctrl + p to print. Select "Save as PDF". Submit this .ipnyb Notebook file, the PDF, and loss graph screenshots to the link specified in the Google Doc.

[257]:
```
predicted_labels.std()
```

[257]: 7966.273

[14]:
```
# Export Model for scoring in part 2
!mkdir saved_model
model.save("saved_model/my_model")
```

```
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'vehicle_CO2:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'vehicle_angle:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'vehicle_eclass:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'vehicle_fuel:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'vehicle_lane:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'vehicle_noise:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'vehicle_pos:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'vehicle_route:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'vehicle_speed:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'vehicle_type:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'vehicle_waiting:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'vehicle_x:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'vehicle_y:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'vehicle_CO2:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'vehicle_angle:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'vehicle_eclass:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'vehicle_fuel:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'vehicle_lane:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'vehicle_noise:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'vehicle_pos:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'vehicle_route:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'vehicle_speed:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'vehicle_type:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'vehicle_waiting:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'vehicle_x:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'vehicle_y:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'vehicle_CO2:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'vehicle_angle:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'vehicle_eclass:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'vehicle_fuel:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'vehicle_lane:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
```

```
'vehicle_noise:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'vehicle_pos:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'vehicle_route:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'vehicle_speed:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'vehicle_type:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'vehicle_waiting:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'vehicle_x:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'vehicle_y:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'inputs:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'inputs_1:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'inputs_2:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'inputs_3:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'inputs_4:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'inputs_5:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'inputs_6:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'inputs_7:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'inputs_8:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'inputs_9:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'inputs_10:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'inputs_11:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'inputs_12:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'inputs:0' shape=(None, 1) dtype=float32>, 'vehicle_angle': <tf.Tensor
'inputs_1:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor
'inputs_2:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor
'inputs_3:0' shape=(None, 1) dtype=float32>, 'vehicle_lane': <tf.Tensor
'inputs_4:0' shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'inputs_5:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor
'inputs_6:0' shape=(None, 1) dtype=float32>, 'vehicle_route': <tf.Tensor
'inputs_7:0' shape=(None, 1) dtype=string>, 'vehicle_speed': <tf.Tensor
'inputs_8:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor
'inputs_9:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'inputs_10:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor
'inputs_11:0' shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor
'inputs_12:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'inputs/vehicle_CO2:0' shape=(None, 1) dtype=float32>, 'vehicle_angle':
<tf.Tensor 'inputs/vehicle_angle:0' shape=(None, 1) dtype=float32>,
'vehicle_eclass': <tf.Tensor 'inputs/vehicle_eclass:0' shape=(None, 1)
dtype=string>, 'vehicle_fuel': <tf.Tensor 'inputs/vehicle_fuel:0' shape=(None,
1) dtype=float32>, 'vehicle_lane': <tf.Tensor 'inputs/vehicle_lane:0'
```

```
shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'inputs/vehicle_noise:0' shape=(None, 1) dtype=float32>, 'vehicle_pos':
<tf.Tensor 'inputs/vehicle_pos:0' shape=(None, 1) dtype=float32>,
'vehicle_route': <tf.Tensor 'inputs/vehicle_route:0' shape=(None, 1)
dtype=string>, 'vehicle_speed': <tf.Tensor 'inputs/vehicle_speed:0' shape=(None,
1) dtype=float32>, 'vehicle_type': <tf.Tensor 'inputs/vehicle_type:0'
shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'inputs/vehicle_waiting:0' shape=(None, 1) dtype=float32>, 'vehicle_x':
<tf.Tensor 'inputs/vehicle_x:0' shape=(None, 1) dtype=float32>, 'vehicle_y':
<tf.Tensor 'inputs/vehicle_y:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'vehicle_CO2': <tf.Tensor
'inputs/vehicle_CO2:0' shape=(None, 1) dtype=float32>, 'vehicle_angle':
<tf.Tensor 'inputs/vehicle_angle:0' shape=(None, 1) dtype=float32>,
'vehicle_eclass': <tf.Tensor 'inputs/vehicle_eclass:0' shape=(None, 1)
dtype=string>, 'vehicle_fuel': <tf.Tensor 'inputs/vehicle_fuel:0' shape=(None,
1) dtype=float32>, 'vehicle_lane': <tf.Tensor 'inputs/vehicle_lane:0'
shape=(None, 1) dtype=string>, 'vehicle_noise': <tf.Tensor
'inputs/vehicle_noise:0' shape=(None, 1) dtype=float32>, 'vehicle_pos':
<tf.Tensor 'inputs/vehicle_pos:0' shape=(None, 1) dtype=float32>,
'vehicle_route': <tf.Tensor 'inputs/vehicle_route:0' shape=(None, 1)
dtype=string>, 'vehicle_speed': <tf.Tensor 'inputs/vehicle_speed:0' shape=(None,
1) dtype=float32>, 'vehicle_type': <tf.Tensor 'inputs/vehicle_type:0'
shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor
'inputs/vehicle_waiting:0' shape=(None, 1) dtype=float32>, 'vehicle_x':
<tf.Tensor 'inputs/vehicle_x:0' shape=(None, 1) dtype=float32>, 'vehicle_y':
<tf.Tensor 'inputs/vehicle_y:0' shape=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
INFO:tensorflow:Assets written to: saved_model/my_model/assets
```

[15]:
```
print(test[1:10])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-328b1073666e> in <module>
----> 1 print(test[1:10])

NameError: name 'test' is not defined
```

[16]:
```
foo = {name:np.array(value) for name, value in test_df_norm.items()}
```

[17]:
```
foo
```

[17]:
```
{'vehicle_CO2': array([ 5286.11, 15350.65,  2624.72, …,  2829.15,  3462.13,
0.  ]),
```

```
 'vehicle_angle': array([ 0.16601731,  0.16601731,  1.24521777, …,
-1.55375926,
        -0.69258678, -0.6952503 ]),
 'vehicle_eclass': array(['HBEFA3/Bus', 'HBEFA3/Bus', 'HBEFA3/PC_G_EU4', …,
        'HBEFA3/PC_G_EU4', 'HBEFA3/PC_G_EU4', 'HBEFA3/PC_G_EU4'],
       dtype=object),
 'vehicle_fuel': array([ 0.04279697,  1.31151019, -0.28765857, …, -0.2611041 ,
        -0.18144072, -0.6210646 ]),
 'vehicle_lane': array(['-256998089#27_0', '729125680#4_0', '652968028#0_0',
…,
        '-339174375#2_0', '5341106#0_0', '-339174378#6_0'], dtype=object),
 'vehicle_noise': array([ 0.1011642 ,  1.18944321, -1.41078561, …,
-0.37123552,
        -0.17767346,  0.20809709]),
 'vehicle_pos': array([-0.30534714, -0.29615243, -0.30213313, …, -0.2254608 ,
        -0.15110785,  0.15737073]),
 'vehicle_route': array(['pt_bus_2U:0', 'pt_bus_14E:0', '!veh13338!var#1', …,
        '!veh14653!var#1', '!veh8489!var#1', '!veh3143!var#1'],
       dtype=object),
 'vehicle_speed': array([-1.50678593, -1.07205582, -1.50678593, …,
-0.17655708,
         0.00344835,  0.63969398]),
 'vehicle_type': array(['pt_bus', 'pt_bus', 'veh_passenger', …,
'veh_passenger',
        'veh_passenger', 'veh_passenger'], dtype=object),
 'vehicle_waiting': array([-0.17696709, -0.17696709,  2.11822929, …,
-0.17696709,
        -0.17696709, -0.17696709]),
 'vehicle_x': array([-0.17059983, -0.13776283, -0.43480711, …,  0.03606987,
        -0.10274732,  0.12744283]),
 'vehicle_y': array([ 0.99653234, -0.3013567 , -0.02697804, …,  0.21022288,
         0.00280091, -0.75090664])}
```

[ ]: