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.

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.

In [1]:

```python
# 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
```

Import Tensorboard

In [2]:
```
# 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:  2.6.1
```

Out[2]:
```
'2.7.0'
```

Load and Clean up the Dataset

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.

In [3]:
```
# loading the large data set, it may takes a while.
emission_train = pd.read_csv("UC-Emission.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

In [4]:
```
display(emission_train.head(100))
display(emission_train.describe())
```

	timestep_time	vehicle_CO	vehicle_CO2	vehicle_HC	vehicle_NOx	vehicle_PMx	vehicle_a
0	0.0	15.20	7380.56	0.00	84.89	2.21	5
1	0.0	0.00	2416.04	0.01	0.72	0.01	4
2	1.0	17.92	9898.93	0.00	103.38	2.49	5
3	1.0	0.00	0.00	0.00	0.00	0.00	4
4	1.0	164.78	2624.72	0.81	1.20	0.07	35
...	...	...	...	...	...	...	
95	7.0	23.44	2578.06	0.15	0.64	0.05	
96	7.0	732.32	18759.70	3.34	3.79	1.19	17
97	7.0	294.68	6949.38	1.29	1.47	0.43	17
98	7.0	236.07	4292.19	0.97	0.93	0.30	
99	7.0	179.19	1228.61	0.64	0.31	0.17	18

100 rows × 20 columns

	timestep_time	vehicle_CO	vehicle_CO2	vehicle_HC	vehicle_NOx	vehicle_PMx
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+0
mean	4.112561e+03	5.764304e+01	4.919050e+03	7.284125e-01	1.769589e+01	4.227491e-0
std	2.168986e+03	8.854365e+01	7.959043e+03	1.589816e+00	5.993168e+01	1.164065e+0
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0
25%	2.291000e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0
50%	4.133000e+03	2.017000e+01	2.624720e+03	1.500000e-01	1.200000e+00	6.000000e-0
75%	5.903000e+03	1.034400e+02	6.161010e+03	7.600000e-01	2.710000e+00	1.500000e-0
max	1.441800e+04	3.932950e+03	1.153026e+05	1.729000e+01	8.864200e+02	1.432000e+0

Visualize the Dataset

Below we use `sns.pairplot()` to show you the 2D plots between datasets. We only use 0.5% of the randomly extracted data from `emission_train` to make plots becuase using too many data might crash the program. `.sample(frac=0.01)` takes a fraction of sample from DataFrame randomly.

- `del` frees up memory for Python. However, it won't release memory back to the computer.

From the pair plots you can visualize the relationships between the data in the dataset. For example, `vehicle_CO2` and `vehicle_fuel` have a linear relationship. `vehicle_CO2` and `vehicle_pos` have a parabolic or exponential like relationship. Some data might have a relationship that is not easily identified from pair plots.

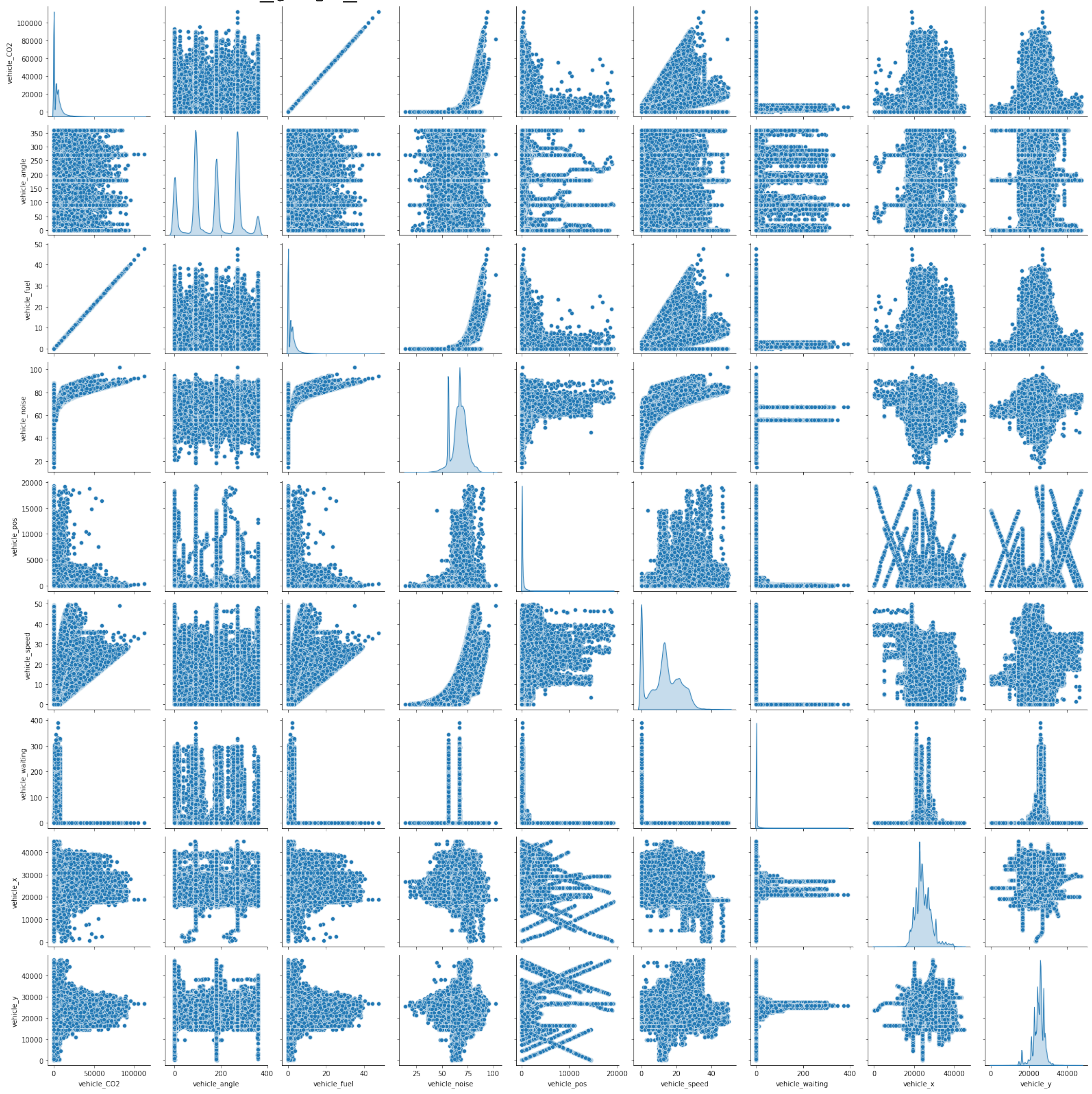
^{D1}Q: What do you find from the Pairplot? Find three pairs of data and list what you observe from their pair plots.

Type your questions to Q: Upon observating the various plots generated below, we observed trends between the data. Three observations include:

- We observed a directly proportional linear relationship between vehicle fuel and vehicle CO2, where vehicle fuel increased by 20 for every 50,000 increase in vehicle CO2.
- Vehicle noise generally tends to increase with vehicle fuel, and vehicle noise also tends to increase with vehicle CO2.
- A majority of vehicle waiting tends to occur in a certain cross-section (near the center of the city), where x-coordinates range from 20,000 to 30,000 and y-coordinates range from 20,000 to 30,000.

In [5]:
```
correlation_graph_data = emission_train.sample(frac=0.05).reset_index(drop=Tr
print(len(emission_train), 'emission_train')
print(len(correlation_graph_data), 'correlation_graph_data')
sns.pairplot(correlation_graph_data[['vehicle_CO2', 'vehicle_angle', 'vehicle_

#Free up memory for Python
del correlation_graph_data
```

16331008 emission_train
816550 correlation_graph_data



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: In addition to dropping columns `vehicle_CO`, `vehicle_HC`, `vehicle_NOx`, `vehicle_PMx`, `timestep_time`, and `vehicle_id`, we decided to drop `vehicle_route`, `vehicle_lane`, and `vehicle_electricity` since these values were respectively used to keep track of each vehicle's unique routes, the name of the road, and `vehicle_electricity` data were all zeros. Dropping these data fields will not affect `vehicle_CO2` output, and thus will not affect the machine learning.

In [6]:
```
emission_train = emission_train.drop(columns=["vehicle_CO", "vehicle_HC", "ve
                                              "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.

In [7]:
```
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	vehicle_noise	vehicle_pos
0	7380.56	50.28	HBEFA3/HDV	3.13	67.11	7.20
1	2416.04	42.25	HBEFA3/PC_G_EU4	1.04	65.15	5.10
2	9898.93	50.28	HBEFA3/HDV	4.20	73.20	8.21
3	0.00	42.25	HBEFA3/PC_G_EU4	0.00	62.72	18.85
4	2624.72	357.00	HBEFA3/PC_G_EU4	1.13	55.94	5.10
...	...	...	...	...	...	...
95	2578.06	0.13	HBEFA3/LDV_G_EU6	1.11	63.24	35.78
96	18759.70	179.93	HBEFA3/LDV_G_EU6	8.07	81.67	30.96
97	6949.38	179.93	HBEFA3/LDV_G_EU6	2.99	72.45	11.88
98	4292.19	1.91	HBEFA3/LDV_G_EU6	1.85	71.73	5.60
99	1228.61	180.06	HBEFA3/LDV_G_EU6	0.53	55.94	2.30

100 rows × 11 columns

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed
count	1.633101e+07	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	1.331140e+01
std	7.959043e+03	1.051232e+02	3.389028e+00	7.389330e+00	6.034189e+02	8.833069e+0
min	0.000000e+00	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	0.000000e+0
25%	0.000000e+00	9.031000e+01	0.000000e+00	6.249000e+01	2.383000e+01	6.550000e+0
50%	2.624720e+03	1.799600e+02	1.130000e+00	6.711000e+01	7.199000e+01	1.337000e+01
75%	6.161010e+03	2.703500e+02	2.650000e+00	7.112000e+01	1.780600e+02	1.999000e+01
max	1.153026e+05	3.600000e+02	4.888000e+01	1.019600e+02	1.943554e+04	5.013000e+01

	vehicle_CO2	vehicle_angle	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_pos
16330908	5293.91	1.98	HBEFA3/Bus	2.26	67.19	77.83
16330909	6541.73	2.07	HBEFA3/Bus	2.79	71.21	0.69
16330910	10387.44	2.06	HBEFA3/Bus	4.43	74.53	2.58
16330911	12058.39	1.62	HBEFA3/Bus	5.14	73.88	5.45
16330912	13307.66	1.06	HBEFA3/Bus	5.67	73.64	9.19
...	...	...	...	...	...	...
16331003	19817.16	0.45	HBEFA3/Bus	8.45	76.56	185.84
16331004	0.00	0.45	HBEFA3/Bus	0.00	74.14	199.17
16331005	23192.37	0.45	HBEFA3/Bus	9.89	77.18	212.90
16331006	0.00	0.45	HBEFA3/Bus	0.00	74.10	226.29
16331007	NaN	NaN	NaN	NaN	NaN	NaN

100 rows × 11 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: After comparing the statistics like count, mean, std, min, 25%, 50%, 75%, and max for both the unshuffled and shuffled data, we observed no changes; the statistics remained the same when frac=1. However, if frac is set to another value, say 0.5, certain statistics like count, mean, and std change.

In [8]:
```
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) #FILL IN THE CODE

### Insert your code below ###

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

# Get info of the dataframe
###FILL IN THE CODE
display(emission_train_shuffle.describe())
```

	vehicle_CO2	vehicle_angle	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_pos	v
0	7380.56	50.28	HBEFA3/HDV	3.13	67.11	7.20	
1	2416.04	42.25	HBEFA3/PC_G_EU4	1.04	65.15	5.10	
2	9898.93	50.28	HBEFA3/HDV	4.20	73.20	8.21	
3	0.00	42.25	HBEFA3/PC_G_EU4	0.00	62.72	18.85	

4	2624.72	357.00	HBEFA3/PC_G_EU4	1.13	55.94	5.10
...	...	...	...	...	...	...
95	2578.06	0.13	HBEFA3/LDV_G_EU6	1.11	63.24	35.78
96	18759.70	179.93	HBEFA3/LDV_G_EU6	8.07	81.67	30.96
97	6949.38	179.93	HBEFA3/LDV_G_EU6	2.99	72.45	11.88
98	4292.19	1.91	HBEFA3/LDV_G_EU6	1.85	71.73	5.60
99	1228.61	180.06	HBEFA3/LDV_G_EU6	0.53	55.94	2.30

100 rows × 11 columns

	vehicle_CO2	vehicle_angle	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_
3978722	8044.33	89.99	HBEFA3/PC_G_EU4	3.46	69.44	3
14299841	5286.11	180.88	HBEFA3/Bus	2.25	67.11	3
15893775	4512.21	270.75	HBEFA3/PC_G_EU4	1.94	66.21	23
503083	0.00	2.25	HBEFA3/PC_G_EU4	0.00	70.22	1
14370069	6478.55	271.96	HBEFA3/PC_G_EU4	2.78	67.99	6
...	...	...	...	...	...	
8269035	0.00	42.10	HBEFA3/PC_G_EU4	0.00	70.32	12
12209957	2624.72	269.58	HBEFA3/PC_G_EU4	1.13	55.94	2
11490803	3477.95	19.42	HBEFA3/PC_G_EU4	1.50	64.35	13
10901379	2517.98	179.39	HBEFA3/PC_G_EU4	1.08	56.37	5
14371642	2672.82	180.86	HBEFA3/PC_G_EU4	1.15	64.28	1

100 rows × 11 columns

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed
count	1.633101e+07	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	1.331140e+0
std	7.959043e+03	1.051232e+02	3.389028e+00	7.389330e+00	6.034189e+02	8.833069e+0
min	0.000000e+00	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	0.000000e+0
25%	0.000000e+00	9.031000e+01	0.000000e+00	6.249000e+01	2.383000e+01	6.550000e+0
50%	2.624720e+03	1.799600e+02	1.130000e+00	6.711000e+01	7.199000e+01	1.337000e+0
75%	6.161010e+03	2.703500e+02	2.650000e+00	7.112000e+01	1.780600e+02	1.999000e+0
max	1.153026e+05	3.600000e+02	4.888000e+01	1.019600e+02	1.943554e+04	5.013000e+0

In [9]:

```
emission_train_shuffle = emission_train.sample(frac=0.5) #Re-set frac value
display(emission_train.describe())
display(emission_train_shuffle.describe())
```

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed
count	1.633101e+07	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	1.331140e+01
std	7.959043e+03	1.051232e+02	3.389028e+00	7.389330e+00	6.034189e+02	8.833069e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	0.000000e+00
25%	0.000000e+00	9.031000e+01	0.000000e+00	6.249000e+01	2.383000e+01	6.550000e+00
50%	2.624720e+03	1.799600e+02	1.130000e+00	6.711000e+01	7.199000e+01	1.337000e+01
75%	6.161010e+03	2.703500e+02	2.650000e+00	7.112000e+01	1.780600e+02	1.999000e+01
max	1.153026e+05	3.600000e+02	4.888000e+01	1.019600e+02	1.943554e+04	5.013000e+01

	vehicle_CO2	vehicle_angle	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed
count	8.165504e+06	8.165504e+06	8.165504e+06	8.165504e+06	8.165504e+06	8.165504e+06
mean	4.919889e+03	1.633551e+02	2.105626e+00	6.636404e+01	2.161897e+02	1.331291e+01
std	7.961184e+03	1.051165e+02	3.389941e+00	7.388797e+00	6.042514e+02	8.835867e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	1.327000e+01	0.000000e+00	0.000000e+00
25%	0.000000e+00	9.031000e+01	0.000000e+00	6.250000e+01	2.382000e+01	6.550000e+00
50%	2.624720e+03	1.799600e+02	1.130000e+00	6.711000e+01	7.199000e+01	1.337000e+01
75%	6.160890e+03	2.703500e+02	2.650000e+00	7.112000e+01	1.780100e+02	2.000000e+01
max	1.153026e+05	3.600000e+02	4.888000e+01	1.019600e+02	1.943554e+04	5.013000e+01

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.

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.

In [43]:
```
# train_df, backup_df = train_test_split(emission_train_shuffle, test_size=0.
# Edit the test_size below.

train_df, test_df = train_test_split(emission_train_shuffle, test_size=0.1) #
# train_df, test_df = train_test_split(train_df, test_size=0.1) # Comment for
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')

# del emission_train
```

```
5879162 train examples
1469791 validation examples
816551 test examples
```

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:

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

In [44]:

```
# Z-Score Normalizing
# train_df_norm = train_df

# for header in ["vehicle_electricity", "vehicle_fuel", "vehicle_noise", "veh

#     train_df_norm[header] = (train_df[header] - train_df[header].mean()) /
#     train_df_norm[header] = train_df_norm[header].fillna(0)

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


# Normalize the test data


# print(train_df_norm.head())
```

Organize Features

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

In [45]:

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


# Numeric Columns
numeric_col_names = ["vehicle_fuel", "vehicle_speed"]
for header in numeric_col_names: ### Finish the list on the left
    ### Insert your code ###
    numeric_column = tf.feature_column.numeric_column(header)
    feature_cols.append(numeric_column)

# Indicator Columns
indicator_col_names = ["vehicle_type"] # removed "vehicle_eclass"
for col_name in indicator_col_names:
    categorical_column = tf.feature_column.categorical_column_with_vocabulary_

    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_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), Indi
catorColumn(categorical_column=VocabularyListCategoricalColumn(key='vehicle_ty
pe', vocabulary_list=('truck_truck', 'veh_passenger', 'moto_motorcycle', 'bus_
bus', 'pt_bus'), dtype=tf.string, default_value=-1, num_oov_buckets=0))]
```

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.

In [46]:
```
# Create a feature layer for tf
feature_layer = tf.keras.layers.DenseFeatures(feature_cols, name='Features')
```

Create and Train the Model

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

Train the Model

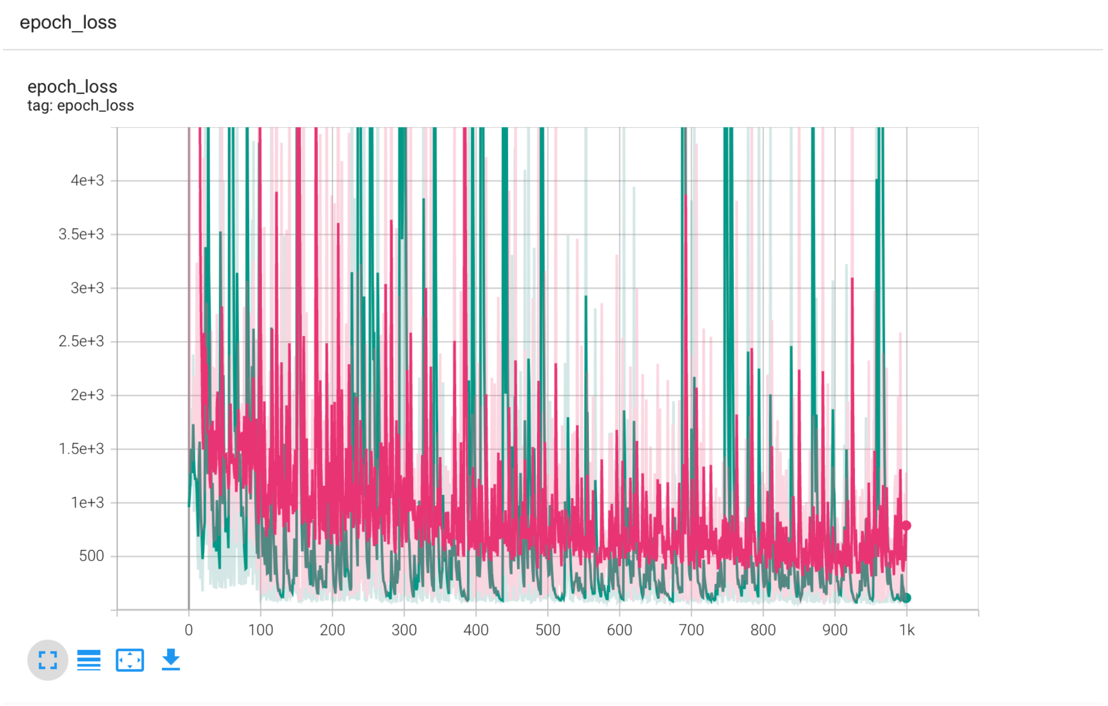
- We first split our Pandas dataframe into features and labels.
- Then `model.fit()` trains our model.
- `logdir` , `tensorboard_callback` is to save training logs to be used in Tensorboard.
- Notice that there are 2 `model.fit()` function calls with one being commented out. The one without `callbacks=[tensorboard_callback]`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

Instructions for Training Small and Large Data

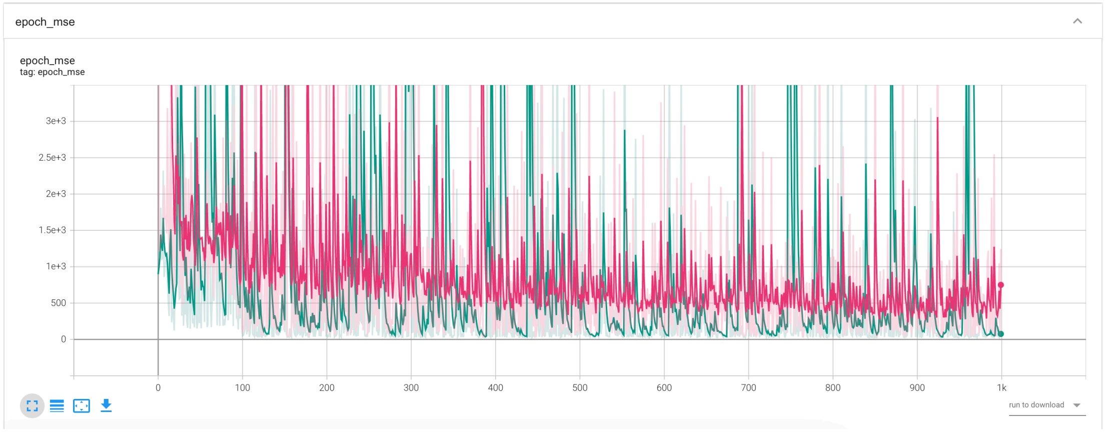
As we mentioned in the lab document, hyperparameters affect the performance of your model. In the following blocks, you would be training your model. We also want you to experience training both a small dataset and a large dataset.

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 number). Add in additional hidden layers as needed. 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. Lastly, start with a small batch size. 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 100, 500, 1000.
    3. In the function definitions (previous code block):
        - 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 < 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`. Then take screenshots that show your **epoch_loss** and your **epoch_mse**.
        - **Screenshot showing Epoch Loss**



- **Screenshot showing Epoch MSE**



- **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: While adjusting the hyperparameters, including learning rate, epochs, batch size, the number of hidden layers and the node numbers, we observed that training with smaller batch sizes took much longer to train. For instance, a batch size of 1000 takes around 400-500 seconds per epoch during training. Training with larger batch sizes like 20,000 was much faster, averaging ~28 seconds per epoch. Meanwhile, training with a batch size of 200,000 took 3-4 seconds per epoch. The batch size I settled on (10,000) took ~50 seconds per epoch. Each test with these varying batch sizes had a parameter value of 50 epochs, and we observe that larger batch sizes (20,000 and 200,000) yielded test mse of 793.2882 and 4994.9922 respectively. Larger batch sizes require more epochs to converge and reach the desired mse.

In [47]:
```
# Hyperparameters
learning_rate = 0.003 ### FILL IN A NUMBER
epochs = 50 ### FILL IN A NUMBER
batch_size = 10000 ### 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 10 nodes
    tf.keras.layers.Dense(units=10,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=0.03)
                          name='Hidden1'),

    # Additional hidden layers
    tf.keras.layers.Dense(units=7,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=0.03)
                          name='Hidden2'),

    tf.keras.layers.Dense(units=4,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=0.03)
                          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)
print(logdir)
train_lbl = np.array(train_df["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.items()}
# train_lbl = np.array(train_ft.pop(label_name))

val_lbl =  np.array(val_df["vehicle_CO2"])
val_df = val_df.drop(columns=["vehicle_CO2"])
val_ft = {name:np.array(value) for name, value in val_df.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)
print(logdir)
# model.fit(x=train_ft, y=train_lbl, batch_size=batch_size,
#           epochs=epochs, callbacks=[tensorboard_callback], validation_data=

# 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), shuffl
```

```
logs/fit/20211120-134638
logs/fit/20211120-134638
```

```
Epoch 1/50
WARNING:tensorflow:Layers in a Sequential model should only have a single inpu
t tensor, but we receive a <class 'dict'> input: {'vehicle_angle': <tf.Tensor
'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor 'E
xpandDims_1:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor 'Expa
ndDims_2:0' shape=(None, 1) dtype=float32>, 'vehicle_noise': <tf.Tensor 'Expan
dDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor 'ExpandDi
ms_4:0' shape=(None, 1) dtype=float32>, 'vehicle_speed': <tf.Tensor 'ExpandDim
s_5:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor 'ExpandDims_
6:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor 'ExpandDims_
7:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor 'ExpandDims_8:0'
shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor 'ExpandDims_9:0' 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 inpu
t tensor, but we receive a <class 'dict'> input: {'vehicle_angle': <tf.Tensor
'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor 'E
xpandDims_1:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor 'Expa
ndDims_2:0' shape=(None, 1) dtype=float32>, 'vehicle_noise': <tf.Tensor 'Expan
dDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor 'ExpandDi
ms_4:0' shape=(None, 1) dtype=float32>, 'vehicle_speed': <tf.Tensor 'ExpandDim
s_5:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor 'ExpandDims_
6:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor 'ExpandDims_
7:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor 'ExpandDims_8:0'
shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor 'ExpandDims_9:0' 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 inpu
t tensor, but we receive a <class 'dict'> input: {'vehicle_angle': <tf.Tensor
'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor 'E
xpandDims_1:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor 'Expa
ndDims_2:0' shape=(None, 1) dtype=float32>, 'vehicle_noise': <tf.Tensor 'Expan
dDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor 'ExpandDi
ms_4:0' shape=(None, 1) dtype=float32>, 'vehicle_speed': <tf.Tensor 'ExpandDim
s_5:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor 'ExpandDims_
6:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor 'ExpandDims_
7:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor 'ExpandDims_8:0'
shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor 'ExpandDims_9:0' shape
=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
588/588 - 63s - loss: 45212280.0000 - mse: 45212280.0000 - val_loss: 333993.75
00 - val_mse: 333986.8438
Epoch 2/50
588/588 - 57s - loss: 108973.7188 - mse: 108966.5078 - val_loss: 21574.1934 -
val_mse: 21566.5898
Epoch 3/50
588/588 - 57s - loss: 9536.5186 - mse: 9528.8271 - val_loss: 3862.8413 - val_m
se: 3855.1025
Epoch 4/50
588/588 - 57s - loss: 2230.3320 - mse: 2222.5818 - val_loss: 1339.5249 - val_m
se: 1331.7650
Epoch 5/50
588/588 - 57s - loss: 1003.3908 - mse: 995.6254 - val_loss: 812.9393 - val_mse
: 805.1710
Epoch 6/50
588/588 - 58s - loss: 724.5302 - mse: 716.7615 - val_loss: 677.3175 - val_mse:
669.5501
Epoch 7/50
588/588 - 59s - loss: 646.8738 - mse: 639.1088 - val_loss: 634.1823 - val_mse:
626.4214
Epoch 8/50
```

```
588/588 - 57s - loss: 616.0852 - mse: 608.3276 - val_loss: 611.0554 - val_mse:
603.3022
Epoch 9/50
588/588 - 57s - loss: 595.8103 - mse: 588.0613 - val_loss: 596.3203 - val_mse:
588.5761
Epoch 10/50
588/588 - 57s - loss: 582.5709 - mse: 574.8306 - val_loss: 581.9381 - val_mse:
574.2005
Epoch 11/50
588/588 - 57s - loss: 573.1579 - mse: 565.4229 - val_loss: 581.9992 - val_mse:
574.2669
Epoch 12/50
588/588 - 57s - loss: 314.6253 - mse: 306.7744 - val_loss: 265.0700 - val_mse:
257.2042
Epoch 13/50
588/588 - 57s - loss: 259.4537 - mse: 251.5933 - val_loss: 258.7079 - val_mse:
250.8478
Epoch 14/50
588/588 - 57s - loss: 247.2422 - mse: 239.3869 - val_loss: 243.8819 - val_mse:
236.0306
Epoch 15/50
588/588 - 57s - loss: 233.0850 - mse: 225.2346 - val_loss: 217.9836 - val_mse:
210.1268
Epoch 16/50
588/588 - 57s - loss: 205.8447 - mse: 197.9735 - val_loss: 192.0312 - val_mse:
184.1456
Epoch 17/50
588/588 - 57s - loss: 183.7385 - mse: 175.8407 - val_loss: 172.5700 - val_mse:
164.6611
Epoch 18/50
588/588 - 57s - loss: 175.2177 - mse: 167.3003 - val_loss: 170.3764 - val_mse:
162.4516
Epoch 19/50
588/588 - 59s - loss: 171.3920 - mse: 163.4616 - val_loss: 162.2280 - val_mse:
154.2929
Epoch 20/50
588/588 - 57s - loss: 168.7560 - mse: 160.8175 - val_loss: 158.8363 - val_mse:
150.8947
Epoch 21/50
588/588 - 56s - loss: 167.7133 - mse: 159.7692 - val_loss: 163.7721 - val_mse:
155.8276
Epoch 22/50
588/588 - 44s - loss: 166.2849 - mse: 158.3399 - val_loss: 156.1001 - val_mse:
148.1546
Epoch 23/50
588/588 - 44s - loss: 161.9686 - mse: 154.0224 - val_loss: 155.6878 - val_mse:
147.7402
Epoch 24/50
588/588 - 44s - loss: 162.4070 - mse: 154.4594 - val_loss: 173.4089 - val_mse:
165.4607
Epoch 25/50
588/588 - 44s - loss: 159.6629 - mse: 151.7146 - val_loss: 147.2769 - val_mse:
139.3285
Epoch 26/50
588/588 - 44s - loss: 155.4250 - mse: 147.4757 - val_loss: 170.2400 - val_mse:
162.2895
Epoch 27/50
588/588 - 45s - loss: 154.2717 - mse: 146.3192 - val_loss: 149.4628 - val_mse:
141.5077
Epoch 28/50
588/588 - 45s - loss: 152.5602 - mse: 144.5930 - val_loss: 146.6997 - val_mse:
```

```
138.7227
Epoch 29/50
588/588 - 41s - loss: 149.1538 - mse: 141.1734 - val_loss: 139.1765 - val_mse:
131.1915
Epoch 30/50
588/588 - 43s - loss: 148.3573 - mse: 140.3676 - val_loss: 153.4761 - val_mse:
145.4818
Epoch 31/50
588/588 - 41s - loss: 146.3187 - mse: 138.3202 - val_loss: 142.0038 - val_mse:
134.0011
Epoch 32/50
588/588 - 41s - loss: 143.1375 - mse: 135.1306 - val_loss: 142.6298 - val_mse:
134.6187
Epoch 33/50
588/588 - 43s - loss: 140.5096 - mse: 132.4947 - val_loss: 177.0830 - val_mse:
169.0647
Epoch 34/50
588/588 - 44s - loss: 142.3383 - mse: 134.3173 - val_loss: 134.5811 - val_mse:
126.5579
Epoch 35/50
588/588 - 44s - loss: 139.4714 - mse: 131.4355 - val_loss: 133.8746 - val_mse:
125.8319
Epoch 36/50
588/588 - 44s - loss: 138.0908 - mse: 130.0464 - val_loss: 148.8189 - val_mse:
140.7709
Epoch 37/50
588/588 - 44s - loss: 137.2615 - mse: 129.2124 - val_loss: 127.6591 - val_mse:
119.6062
Epoch 38/50
588/588 - 44s - loss: 135.5978 - mse: 127.5431 - val_loss: 137.6424 - val_mse:
129.5742
Epoch 39/50
588/588 - 44s - loss: 134.7056 - mse: 126.6323 - val_loss: 129.0390 - val_mse:
120.9626
Epoch 40/50
588/588 - 44s - loss: 132.8154 - mse: 124.7381 - val_loss: 123.2225 - val_mse:
115.1417
Epoch 41/50
588/588 - 44s - loss: 133.6584 - mse: 125.5769 - val_loss: 130.5829 - val_mse:
122.4996
Epoch 42/50
588/588 - 44s - loss: 132.9434 - mse: 124.8585 - val_loss: 129.6786 - val_mse:
121.5843
Epoch 43/50
588/588 - 44s - loss: 131.6401 - mse: 123.5460 - val_loss: 136.1933 - val_mse:
128.0992
Epoch 44/50
588/588 - 44s - loss: 131.5917 - mse: 123.4890 - val_loss: 131.8686 - val_mse:
123.7620
Epoch 45/50
588/588 - 44s - loss: 129.7118 - mse: 121.6050 - val_loss: 126.6633 - val_mse:
118.5566
Epoch 46/50
588/588 - 44s - loss: 130.5663 - mse: 122.4539 - val_loss: 129.8120 - val_mse:
121.6893
Epoch 47/50
588/588 - 44s - loss: 130.9231 - mse: 122.7984 - val_loss: 165.9403 - val_mse:
157.8141
Epoch 48/50
588/588 - 44s - loss: 129.5071 - mse: 121.3807 - val_loss: 135.7780 - val_mse:
127.6507
```

```
Epoch 49/50
588/588 - 44s - loss: 128.7061 - mse: 120.5782 - val_loss: 134.7392 - val_mse:
126.6103
Epoch 50/50
588/588 - 44s - loss: 131.1464 - mse: 123.0168 - val_loss: 158.4046 - val_mse:
150.2738
```

Out[47]:

```
<keras.callbacks.History at 0x7f96a0cf3190>
```

Evaluate the Model with Test Data

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

In [48]:

```
test_lbl = np.array(test_df["vehicle_CO2"])
test_df = test_df.drop(columns=["vehicle_CO2"])
test_ft = {key:np.array(value) for key, value in test_df.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:

82/82 [==============================] - 1s 11ms/step - loss: 159.2563 - mse:
151.1256
```

Out[48]:

```
[159.25631713867188, 151.12557983398438]
```

In [49]:

```
#Get a summary of your model
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Features (DenseFeatures)     multiple                  0
_________________________________________________________________
Hidden1 (Dense)              multiple                  80
_________________________________________________________________
Hidden2 (Dense)              multiple                  77
_________________________________________________________________
Hidden3 (Dense)              multiple                  32
_________________________________________________________________
Output (Dense)               multiple                  5
=================================================================
Total params: 194
Trainable params: 194
Non-trainable params: 0
_________________________________________________________________
```

Use the Trained-Model and Visualize the results

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

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.

In [50]:
```
%tensorboard --logdir logs
```

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.

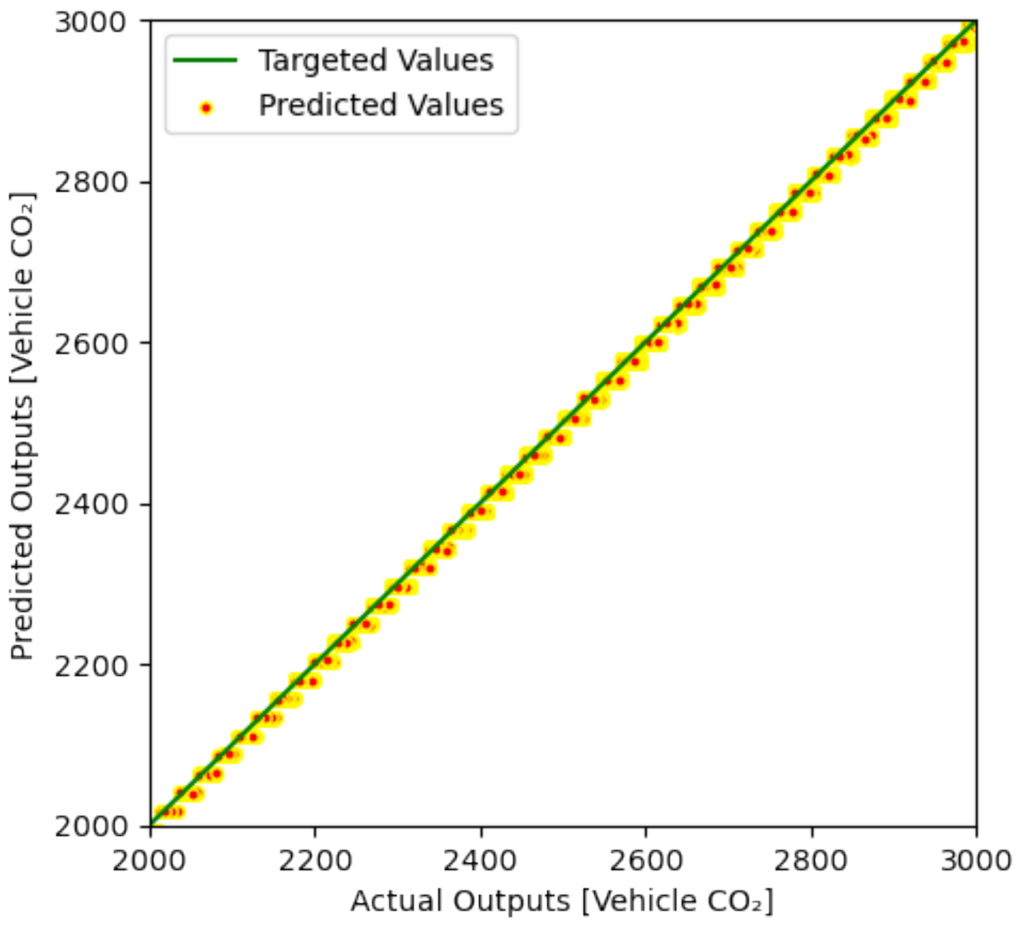
- `.predict()` : predicts the output values from features given.
- `predicted_labels` : contains the values (CO_2) our model predicts. After the predicted and actual values are obtained. We create a plot for you to visualize the results. The dots show the predicted values and the line shows the targeted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

In [51]:

```
%%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
predicted_labels = model.predict(x=test_features).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='Yello

# 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 inpu
t tensor, but we receive a <class 'dict'> input: {'vehicle_angle': <tf.Tensor
'ExpandDims:0' shape=(None, 1) dtype=float32>, 'vehicle_eclass': <tf.Tensor 'E
xpandDims_1:0' shape=(None, 1) dtype=string>, 'vehicle_fuel': <tf.Tensor 'Expa
ndDims_2:0' shape=(None, 1) dtype=float32>, 'vehicle_noise': <tf.Tensor 'Expan
dDims_3:0' shape=(None, 1) dtype=float32>, 'vehicle_pos': <tf.Tensor 'ExpandDi
ms_4:0' shape=(None, 1) dtype=float32>, 'vehicle_speed': <tf.Tensor 'ExpandDim
s_5:0' shape=(None, 1) dtype=float32>, 'vehicle_type': <tf.Tensor 'ExpandDims_
6:0' shape=(None, 1) dtype=string>, 'vehicle_waiting': <tf.Tensor 'ExpandDims_
7:0' shape=(None, 1) dtype=float32>, 'vehicle_x': <tf.Tensor 'ExpandDims_8:0'
shape=(None, 1) dtype=float32>, 'vehicle_y': <tf.Tensor 'ExpandDims_9:0' shape
=(None, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
CPU times: user 4min 58s, sys: 5.92 s, total: 5min 4s
Wall time: 4min 38s
```

```
Out[51]: <matplotlib.legend.Legend at 0x7f96ba958370>
```



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.

```
In [52]:
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')
```

```
Out[52]: Text(0, 0.5, 'Count')
```

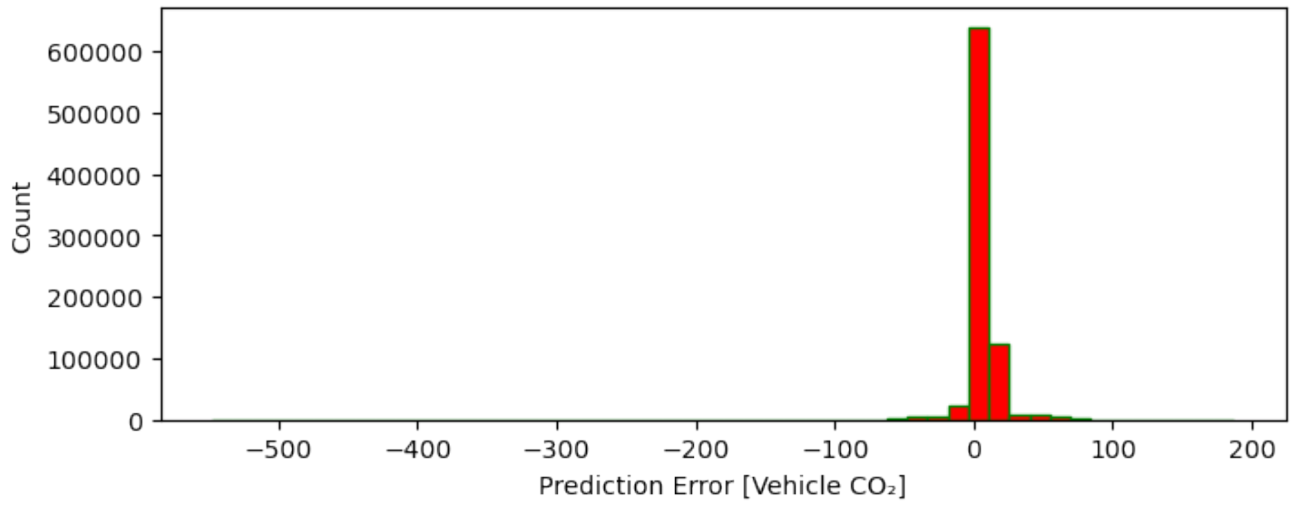


Table of Actual and Predicted Values

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

```
In [53]:
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])
```

Actual Vehicle CO_2	Predicted Vehicle CO_2
7506	7508
0	0

4562	4553
4450	4436
10575	10577
2624	2625
0	0
22828	22836
0	0
0	0
0	0
0	0
0	0
0	0
2624	2625
13031	13020
0	0
7380	7372
0	0
3789	3784
0	0
11141	11136
0	0
0	0

34951	34933
3762	3760
7641	7623
17286	17239
4664	4671
7630	7622
8240	8229
8786	8776
13317	13298
2624	2625
0	0
6350	6341
0	0
0	0
0	0
0	0
12115	12112
16141	16136
0	0
2624	2625
8647	8644
5286	5332
0	0

12960	12950
0	0
0	0
5671	5671
3500	3484
0	0
0	0
0	0
0	0
7547	7531
13131	13112
3016	3017
4741	4737
0	0
5837	5832
0	0
0	0
37510	37474
5290	5287
0	0
9601	9601
0	0
0	0

0	0
6054	6039
0	0
0	0
0	0
4699	4692
4626	4624
4907	4900
17561	17555
2955	2950
5154	5157
0	0
2624	2625
38856	38852
5676	5666
0	0
7901	7900
0	0
40600	40605
35198	35213
0	0
17966	17994

3427	3413
2733	2715
3929	3927
13209	13206
2624	2625
0	0
3697	3692
0	0

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.