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 (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 [46]:
```
# 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 [47]:
```
# 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__
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
TensorFlow version:  2.7.0
```

Out[47]:
```
'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 [48]:
```
# 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 (https://sumo.dlr.de/docs/Simulation/Output/EmissionOutput.html)

In [49]:
```
display(emission_train.head(100))
display(emission_train.describe())
```

	timestep_time	vehicle_CO	vehicle_CO2	vehicle_HC	vehicle_NOx	vehicle_PMx	vehicle_angle
0	0.0	15.20	7380.56	0.00	84.89	2.21	50.28
1	0.0	0.00	2416.04	0.01	0.72	0.01	42.25
2	1.0	17.92	9898.93	0.00	103.38	2.49	50.28
3	1.0	0.00	0.00	0.00	0.00	0.00	42.25
4	1.0	164.78	2624.72	0.81	1.20	0.07	357.00
...	...	...	...	...	...	...	...
95	7.0	23.44	2578.06	0.15	0.64	0.05	0.13
96	7.0	732.32	18759.70	3.34	3.79	1.19	179.93
97	7.0	294.68	6949.38	1.29	1.47	0.43	179.93
98	7.0	236.07	4292.19	0.97	0.93	0.30	1.91
99	7.0	179.19	1228.61	0.64	0.31	0.17	180.06

100 rows × 20 columns

	timestep_time	vehicle_CO	vehicle_CO2	vehicle_HC	vehicle_NOx	vehicle_PMx	ve
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.
mean	4.112561e+03	5.764304e+01	4.919050e+03	7.284125e-01	1.769589e+01	4.227491e-01	1.
std	2.168986e+03	8.854365e+01	7.959043e+03	1.589816e+00	5.993168e+01	1.164065e+00	1.
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.
25%	2.291000e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	9.
50%	4.133000e+03	2.017000e+01	2.624720e+03	1.500000e-01	1.200000e+00	6.000000e-02	1.
75%	5.903000e+03	1.034400e+02	6.161010e+03	7.600000e-01	2.710000e+00	1.500000e-01	2.
max	1.441800e+04	3.932950e+03	1.153026e+05	1.729000e+01	8.864200e+02	1.432000e+01	3.

Visualiz 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 answers to Q:

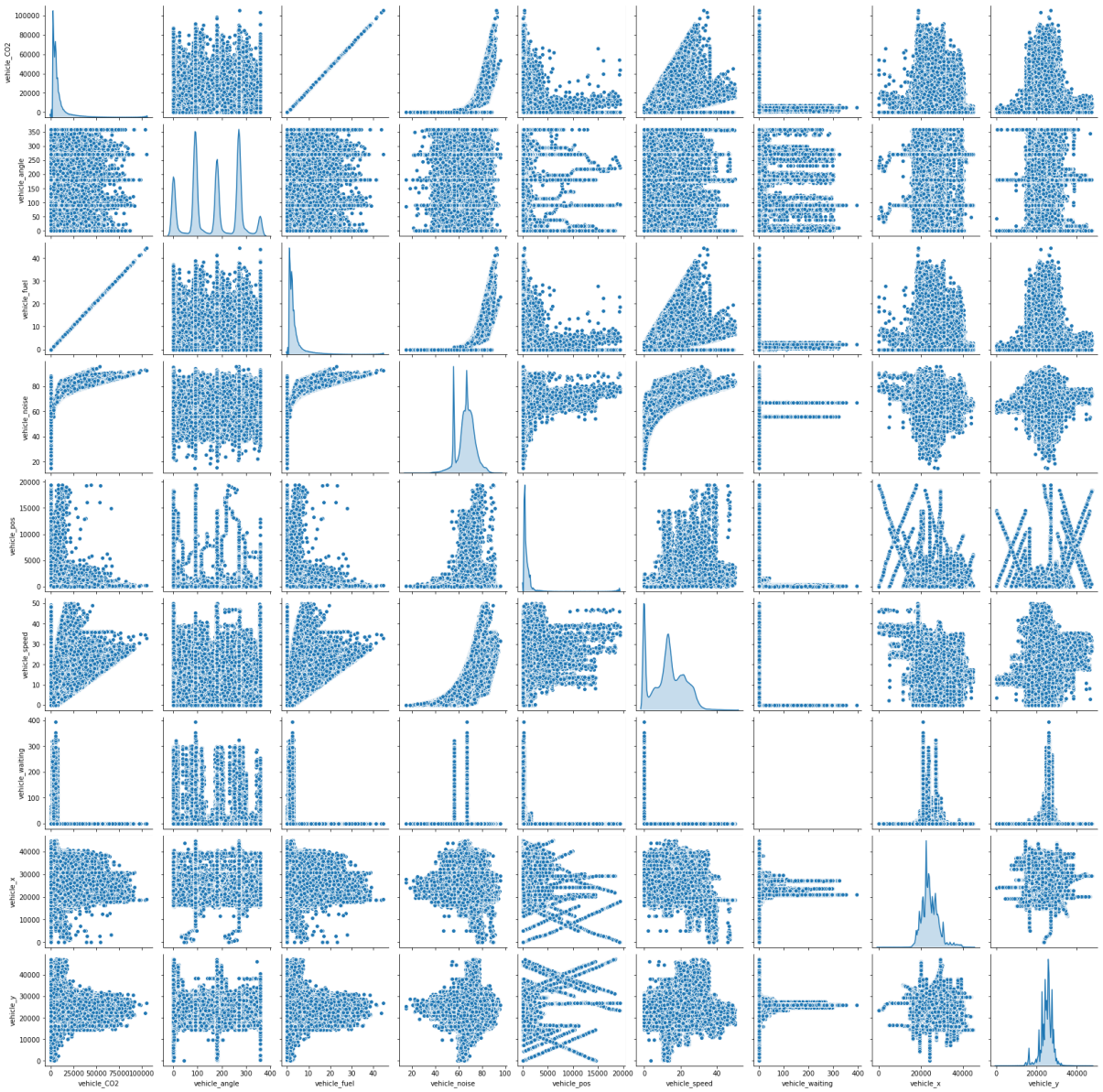
1) Vehicle Fuel and Vehicle C02 has a linear relationship, seems like the more fuel used, the more C02 emissions emitted, in a very coorelated manner.

2) Vehicle Speed and Vehicle C02 have a linear upper-bound. Can probably use this fact as feature input for out model.

3) Vehicle C02 and Vehicle Angle seems to have very little coorelation, we can probably omit this feature from our trainable feature data.

In [23]:
```
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

/Users/ryan/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```



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 am dropping any data that does not have an obvious coorelation from the graphs in the previous step. If there is no coorelation in the data towards vehicle_C02, then the extra added data will just result in noise and will make it harder to train an more accurate model. It is wise to only keep data that is coorelated towards the target value that we want. The data columns "vehicle_CO", "vehicle_HC", "vehicle_NOx", "vehicle_PMx", "vehicle_angle","timestep_time", "vehicle_id","vehicle_route", "vehicle_lane" and "vehicle_electricity" seem to have no coorelation so we will drop them below.

In [50]:
```
emission_train = emission_train.drop(columns=["vehicle_CO", "vehicle_HC"
, "vehicle_NOx", "vehicle_PMx", "vehicle_angle",
                                              "timestep_time", "vehicle_
id","vehicle_route", "vehicle_lane", "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 [51]:
```
display(emission_train.head(100))
display(emission_train.describe())
display(emission_train.tail(100))
```

	vehicle_CO2	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed	
0	7380.56	HBEFA3/HDV	3.13	67.11	7.20	0.00	
1	2416.04	HBEFA3/PC_G_EU4	1.04	65.15	5.10	14.72	v
2	9898.93	HBEFA3/HDV	4.20	73.20	8.21	1.01	
3	0.00	HBEFA3/PC_G_EU4	0.00	62.72	18.85	13.75	v
4	2624.72	HBEFA3/PC_G_EU4	1.13	55.94	5.10	0.00	v
...	...	...	...	...	...	...	
95	2578.06	HBEFA3/LDV_G_EU6	1.11	63.24	35.78	11.62	mot
96	18759.70	HBEFA3/LDV_G_EU6	8.07	81.67	30.96	13.99	mot
97	6949.38	HBEFA3/LDV_G_EU6	2.99	72.45	11.88	6.37	mot
98	4292.19	HBEFA3/LDV_G_EU6	1.85	71.73	5.60	3.30	mot
99	1228.61	HBEFA3/LDV_G_EU6	0.53	55.94	2.30	0.00	mot

100 rows × 10 columns

	vehicle_CO2	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed	vehicle_waiting	
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	
mean	4.919050e+03	2.105266e+00	6.636207e+01	2.162082e+02	1.331140e+01	3.385107e+00	2
std	7.959043e+03	3.389028e+00	7.389330e+00	6.034189e+02	8.833069e+00	1.914152e+01	4
min	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	0.000000e+00	0.000000e+00	9
25%	0.000000e+00	0.000000e+00	6.249000e+01	2.383000e+01	6.550000e+00	0.000000e+00	2
50%	2.624720e+03	1.130000e+00	6.711000e+01	7.199000e+01	1.337000e+01	0.000000e+00	2
75%	6.161010e+03	2.650000e+00	7.112000e+01	1.780600e+02	1.999000e+01	0.000000e+00	2
max	1.153026e+05	4.888000e+01	1.019600e+02	1.943554e+04	5.013000e+01	3.970000e+02	4

	vehicle_CO2	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed	ve
16330908	5293.91	HBEFA3/Bus	2.26	67.19	77.83	0.01	
16330909	6541.73	HBEFA3/Bus	2.79	71.21	0.69	0.70	
16330910	10387.44	HBEFA3/Bus	4.43	74.53	2.58	1.88	
16330911	12058.39	HBEFA3/Bus	5.14	73.88	5.45	2.87	
16330912	13307.66	HBEFA3/Bus	5.67	73.64	9.19	3.74	
...	...	...	...	...	...	...	
16331003	19817.16	HBEFA3/Bus	8.45	76.56	185.84	13.65	
16331004	0.00	HBEFA3/Bus	0.00	74.14	199.17	13.33	
16331005	23192.37	HBEFA3/Bus	9.89	77.18	212.90	13.73	
16331006	0.00	HBEFA3/Bus	0.00	74.10	226.29	13.39	
16331007	NaN	NaN	NaN	NaN	NaN	NaN	

100 rows × 10 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:

The order of the rows are different from comparing the head and tails of the pre and post shuffled data, however the aggregate values such as count, mean, std, etc... stay the same, likely due to the fact that even though the data is shuffled, it is still the same pool of data.

In [52]:
```
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)

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

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

	vehicle_CO2	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed	
0	7380.56	HBEFA3/HDV	3.13	67.11	7.20	0.00	
1	2416.04	HBEFA3/PC_G_EU4	1.04	65.15	5.10	14.72	v
2	9898.93	HBEFA3/HDV	4.20	73.20	8.21	1.01	
3	0.00	HBEFA3/PC_G_EU4	0.00	62.72	18.85	13.75	v
4	2624.72	HBEFA3/PC_G_EU4	1.13	55.94	5.10	0.00	v
...	...	...	...	...	...	...	
95	2578.06	HBEFA3/LDV_G_EU6	1.11	63.24	35.78	11.62	mot
96	18759.70	HBEFA3/LDV_G_EU6	8.07	81.67	30.96	13.99	mot
97	6949.38	HBEFA3/LDV_G_EU6	2.99	72.45	11.88	6.37	mot
98	4292.19	HBEFA3/LDV_G_EU6	1.85	71.73	5.60	3.30	mot
99	1228.61	HBEFA3/LDV_G_EU6	0.53	55.94	2.30	0.00	mot

100 rows × 10 columns

	vehicle_CO2	vehicle_eclass	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed
3108234	7653.56	HBEFA3/PC_G_EU4	3.29	71.77	182.84	23.52
14975653	14186.70	HBEFA3/HDV	6.01	73.19	133.35	3.18
1506697	55988.03	HBEFA3/HDV	23.74	83.34	162.88	19.06
10080547	13878.76	HBEFA3/PC_G_EU4	5.97	74.48	61.44	18.30
5215588	13010.81	HBEFA3/PC_G_EU4	5.59	73.93	64.15	16.98
...	...	...	...	...	...	..
7561174	0.00	HBEFA3/PC_G_EU4	0.00	61.63	45.29	11.34
8836656	3613.78	HBEFA3/PC_G_EU4	1.55	63.85	5.39	3.49
9170308	2624.72	HBEFA3/PC_G_EU4	1.13	55.94	24.92	0.00
10792601	5286.11	HBEFA3/Bus	2.25	67.11	35.35	0.00
4054514	9942.27	HBEFA3/PC_G_EU4	4.27	71.45	33.30	18.25

100 rows × 10 columns

	vehicle_CO2	vehicle_fuel	vehicle_noise	vehicle_pos	vehicle_speed	vehicle_waiting
count	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07	1.633101e+07
mean	4.919050e+03	2.105266e+00	6.636207e+01	2.162082e+02	1.331140e+01	3.385107e+00
std	7.959043e+03	3.389028e+00	7.389330e+00	6.034189e+02	8.833069e+00	1.914152e+01
min	0.000000e+00	0.000000e+00	1.258000e+01	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	6.249000e+01	2.383000e+01	6.550000e+00	0.000000e+00
50%	2.624720e+03	1.130000e+00	6.711000e+01	7.199000e+01	1.337000e+01	0.000000e+00
75%	6.161010e+03	2.650000e+00	7.112000e+01	1.780600e+02	1.999000e+01	0.000000e+00
max	1.153026e+05	4.888000e+01	1.019600e+02	1.943554e+04	5.013000e+01	3.970000e+02

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 [37]:
```
#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=0.2) # 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')

del emission_train
```

```
11758324 train examples
2939582 validation examples
1633101 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 : Normalized Data

x : Original Data

μ : Mean of x

σ : Standard Deviation of x

In [10]:
```
# # # Z-Score Normalizing
# train_df_norm = train_df

# for header in ["vehicle_noise", "vehicle_speed"]:

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

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

# for header in ["vehicle_noise",  "vehicle_speed"]:

#     val_df_norm[header] = (val_df[header] - val_df[header].mean()) / v
al_df[header].std()
#     val_df_norm[header] = val_df_norm[header].fillna(0)

# #Normalize the test data
# test_df_norm = test_df

# for header in ["vehicle_noise",  "vehicle_speed"]:

#     test_df_norm[header] = (test_df[header] - test_df[header].mean())
 / test_df[header].std()
#     test_df_norm[header] = test_df_norm[header].fillna(0)

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

```
          vehicle_CO2    vehicle_eclass  vehicle_fuel  vehicle_noise  \
13356652         0.00   HBEFA3/PC_G_EU4          0.00      -2.334472
8026016       8074.51  HBEFA3/LDV_G_EU6          3.47       0.531469
9663869       2624.72   HBEFA3/PC_G_EU4          1.13      -1.410281
9666402          0.00   HBEFA3/PC_G_EU4          0.00       0.293317
3697402          0.00   HBEFA3/PC_G_EU4          0.00       0.596419

          vehicle_pos  vehicle_speed     vehicle_type  vehicle_waiting
\
13356652       515.50      -1.000843    veh_passenger              0.0
8026016         20.21      -0.343075  moto_motorcycle              0.0
9663869         88.28      -1.506905    veh_passenger             16.0
9666402        492.94       0.898871    veh_passenger              0.0
3697402          6.56       1.324552    veh_passenger              0.0

          vehicle_x  vehicle_y
13356652   20260.27   27634.40
8026016    21288.00   25274.66
9663869    23561.58   22705.34
9666402    29625.38   22654.08
3697402    30758.20   25069.53
```

```
<ipython-input-10-b4534cc387a3>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df_norm[header] = (test_df[header] - test_df[header].mean()) / t
est_df[header].std()
<ipython-input-10-b4534cc387a3>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df_norm[header] = test_df_norm[header].fillna(0)
```

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 (https://www.tensorflow.org/tutorials/structured_data/feature_columns)

In [38]:
```
# Create an empty list
feature_cols = []


# Numeric Columns
for header in ["vehicle_fuel", "vehicle_speed","vehicle_noise" ]: ### Fi
nish the list on the left
     feature_cols.append(tf.feature_column.numeric_column(header))

# Indicator Columns
indicator_col_names = ["vehicle_eclass", "vehicle_type"]
for col_name in indicator_col_names:
    categorical_column = tf.feature_column.categorical_column_with_vocab
ulary_list(col_name,

train_df[col_name].unique())
    indicator_column = tf.feature_column.indicator_column(categorical_co
lumn)
    feature_cols.append(indicator_column)

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

```
Feature columns:  [NumericColumn(key='vehicle_fuel', shape=(1,), defaul
t_value=None, dtype=tf.float32, normalizer_fn=None), NumericColumn(key
='vehicle_speed', shape=(1,), default_value=None, dtype=tf.float32, nor
malizer_fn=None), NumericColumn(key='vehicle_noise', shape=(1,), defaul
t_value=None, dtype=tf.float32, normalizer_fn=None), IndicatorColumn(ca
tegorical_column=VocabularyListCategoricalColumn(key='vehicle_eclass',
vocabulary_list=('HBEFA3/PC_G_EU4', 'HBEFA3/LDV_G_EU6', 'HBEFA3/HDV',
'HBEFA3/Bus'), dtype=tf.string, default_value=-1, num_oov_buckets=0)),
IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key
='vehicle_type', vocabulary_list=('veh_passenger', 'moto_motorcycle',
'truck_truck', '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 [39]:
```
# 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**.

- **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:

Yes, higher batch sizes seem to execute through the entire data-set faster however it results in less accurate results. Finding a sweet-spot batch-size that maximizes accuracy and efficiency is the goal.

In [40]:
```
# Hyperparameters
learning_rate = 0.001
epochs = 3
batch_size = 1000

# 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=10,
#                           activation='relu',
#                           kernel_regularizer=tf.keras.regularizers.l1
(l=0.3),
#                           name='Hidden1'),

    tf.keras.layers.Dense(units=10,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=
0.3),
                          name='Hidden2'),

    tf.keras.layers.Dense(units=20,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=
0.3),
                          name='Hidden3'),

    tf.keras.layers.Dense(units=8,
                          activation='relu',
                          kernel_regularizer=tf.keras.regularizers.l1(l=
0.3),
                          name='Hidden4'),

#     tf.keras.layers.Dense(units=4,
#                           activation='relu',
#                           kernel_regularizer=tf.keras.regularizers.l1
(l=0.3),
#                           name='Hidden5'),

    # Additional hidden layers

    # Output layer
    tf.keras.layers.Dense(units=1,
                          activation='softplus',
                          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["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)

# 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=1, validation_data=(val_ft, val_lbl), s
huffle=shuffle)
```

```
/Users/ryan/.local/lib/python3.8/site-packages/keras/optimizer_v2/adam.
py:105: UserWarning: The `lr` argument is deprecated, use `learning_rat
e` instead.
  super(Adam, self).__init__(name, **kwargs)

Epoch 1/3
WARNING:tensorflow:Layers in a Sequential model should only have a sing
le input tensor, but we receive a <class 'dict'> input: {'vehicle_eclas
s': <tf.Tensor 'IteratorGetNext:0' shape=(None,) dtype=string>, 'vehicl
e_fuel': <tf.Tensor 'IteratorGetNext:1' shape=(None,) dtype=float32>,
'vehicle_noise': <tf.Tensor 'IteratorGetNext:2' shape=(None,) dtype=flo
at32>, 'vehicle_pos': <tf.Tensor 'IteratorGetNext:3' shape=(None,) dtyp
e=float32>, 'vehicle_speed': <tf.Tensor 'IteratorGetNext:4' shape=(Non
e,) dtype=float32>, 'vehicle_type': <tf.Tensor 'IteratorGetNext:5' shap
e=(None,) dtype=string>, 'vehicle_waiting': <tf.Tensor 'IteratorGetNex
t:6' shape=(None,) dtype=float32>, 'vehicle_x': <tf.Tensor 'IteratorGet
Next:7' shape=(None,) dtype=float32>, 'vehicle_y': <tf.Tensor 'Iterator
GetNext:8' shape=(None,) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a sing
le input tensor, but we receive a <class 'dict'> input: {'vehicle_eclas
s': <tf.Tensor 'IteratorGetNext:0' shape=(None,) dtype=string>, 'vehicl
e_fuel': <tf.Tensor 'IteratorGetNext:1' shape=(None,) dtype=float32>,
'vehicle_noise': <tf.Tensor 'IteratorGetNext:2' shape=(None,) dtype=flo
at32>, 'vehicle_pos': <tf.Tensor 'IteratorGetNext:3' shape=(None,) dtyp
e=float32>, 'vehicle_speed': <tf.Tensor 'IteratorGetNext:4' shape=(Non
e,) dtype=float32>, 'vehicle_type': <tf.Tensor 'IteratorGetNext:5' shap
e=(None,) dtype=string>, 'vehicle_waiting': <tf.Tensor 'IteratorGetNex
t:6' shape=(None,) dtype=float32>, 'vehicle_x': <tf.Tensor 'IteratorGet
Next:7' shape=(None,) dtype=float32>, 'vehicle_y': <tf.Tensor 'Iterator
GetNext:8' shape=(None,) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a sing
le input tensor, but we receive a <class 'dict'> input: {'vehicle_eclas
s': <tf.Tensor 'IteratorGetNext:0' shape=(None,) dtype=string>, 'vehicl
e_fuel': <tf.Tensor 'IteratorGetNext:1' shape=(None,) dtype=float32>,
'vehicle_noise': <tf.Tensor 'IteratorGetNext:2' shape=(None,) dtype=flo
at32>, 'vehicle_pos': <tf.Tensor 'IteratorGetNext:3' shape=(None,) dtyp
e=float32>, 'vehicle_speed': <tf.Tensor 'IteratorGetNext:4' shape=(Non
e,) dtype=float32>, 'vehicle_type': <tf.Tensor 'IteratorGetNext:5' shap
e=(None,) dtype=string>, 'vehicle_waiting': <tf.Tensor 'IteratorGetNex
t:6' shape=(None,) dtype=float32>, 'vehicle_x': <tf.Tensor 'IteratorGet
Next:7' shape=(None,) dtype=float32>, 'vehicle_y': <tf.Tensor 'Iterator
GetNext:8' shape=(None,) dtype=float32>}
Consider rewriting this model with the Functional API.
11759/11759 - 1267s - loss: 5465434.0000 - mse: 5465413.5000 - val_los
s: 606.0892 - val_mse: 503.4232 - 1267s/epoch - 108ms/step
Epoch 2/3
11759/11759 - 1238s - loss: 391.0412 - mse: 288.0979 - val_loss: 266.86
62 - val_mse: 164.9962 - 1238s/epoch - 105ms/step
Epoch 3/3
11759/11759 - 1230s - loss: 268.9817 - mse: 167.8427 - val_loss: 240.27
03 - val_mse: 139.5141 - 1230s/epoch - 105ms/step
```

Out[40]:
```
<keras.callbacks.History at 0x7fdb784037c0>
```

Evaluate the Model with Test Data

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

In [41]:
```
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:

1634/1634 [==============================] - 23s 14ms/step - loss: 241.
6248 - mse: 140.8686
```

Out[41]:
```
[241.62481689453125, 140.8685760498047]
```

In [42]:
```
#Get a summary of your model
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 Features (DenseFeatures)     multiple                  0

 Hidden2 (Dense)              multiple                  130

 Hidden3 (Dense)              multiple                  220

 Hidden4 (Dense)              multiple                  168

 Output (Dense)               multiple                  9

=================================================================
Total params: 527
Trainable params: 527
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 [ ]:
```
%reload_ext tensorboard
```

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

!!!NOTE: Tensorboard commands were ran in terminal (sceenshots along with this report) because ipython couldnt resolve the path

Predict CO_2 From Trained-Model

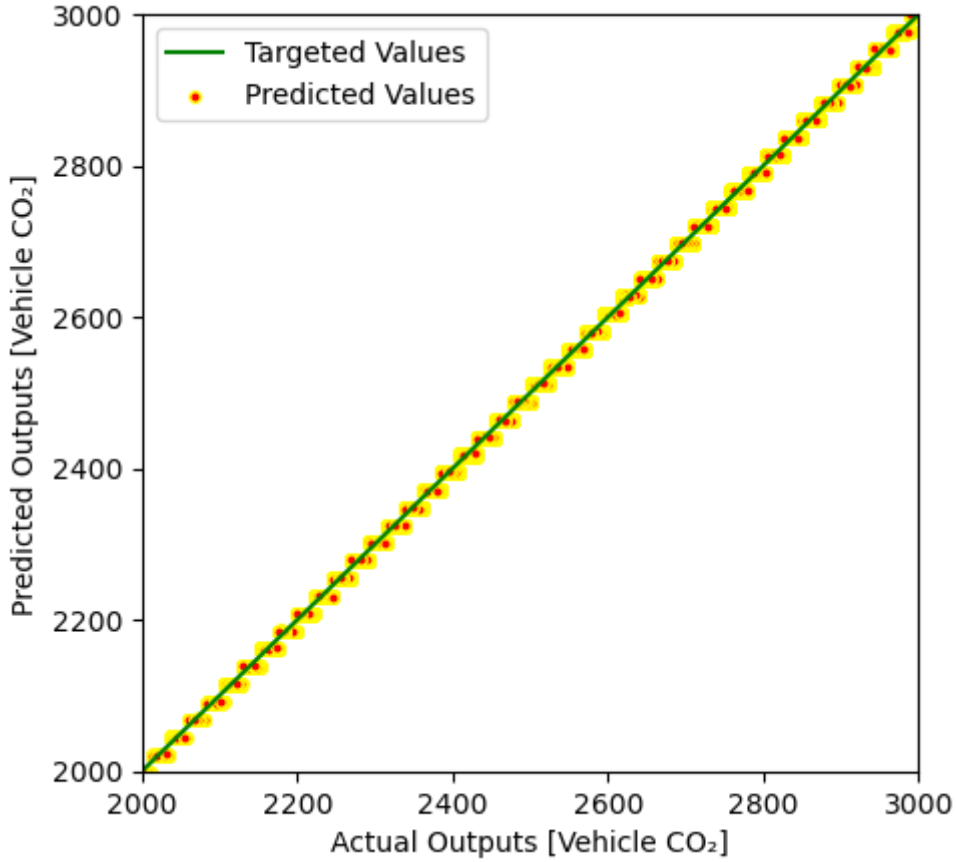
Below, your trained-model is used to make prediction on the test set. Remember, test set is not used in training the model so it would give you a nice indication of how your model is doing.

- `.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 [43]:
```
%%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=
'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 sing
le input tensor, but we receive a <class 'dict'> input: {'vehicle_eclas
s': <tf.Tensor 'IteratorGetNext:0' shape=(None,) dtype=string>, 'vehicl
e_fuel': <tf.Tensor 'IteratorGetNext:1' shape=(None,) dtype=float32>,
'vehicle_noise': <tf.Tensor 'IteratorGetNext:2' shape=(None,) dtype=flo
at32>, 'vehicle_pos': <tf.Tensor 'IteratorGetNext:3' shape=(None,) dtyp
e=float32>, 'vehicle_speed': <tf.Tensor 'IteratorGetNext:4' shape=(Non
e,) dtype=float32>, 'vehicle_type': <tf.Tensor 'IteratorGetNext:5' shap
e=(None,) dtype=string>, 'vehicle_waiting': <tf.Tensor 'IteratorGetNex
t:6' shape=(None,) dtype=float32>, 'vehicle_x': <tf.Tensor 'IteratorGet
Next:7' shape=(None,) dtype=float32>, 'vehicle_y': <tf.Tensor 'Iterator
GetNext:8' shape=(None,) dtype=float32>}
Consider rewriting this model with the Functional API.
CPU times: user 12min 59s, sys: 15.1 s, total: 13min 14s
Wall time: 12min 5s
```

Out[43]: <matplotlib.legend.Legend at 0x7fdb4028ebe0>



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 [44]:
```
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[44]: Text(0, 0.5, 'Count')

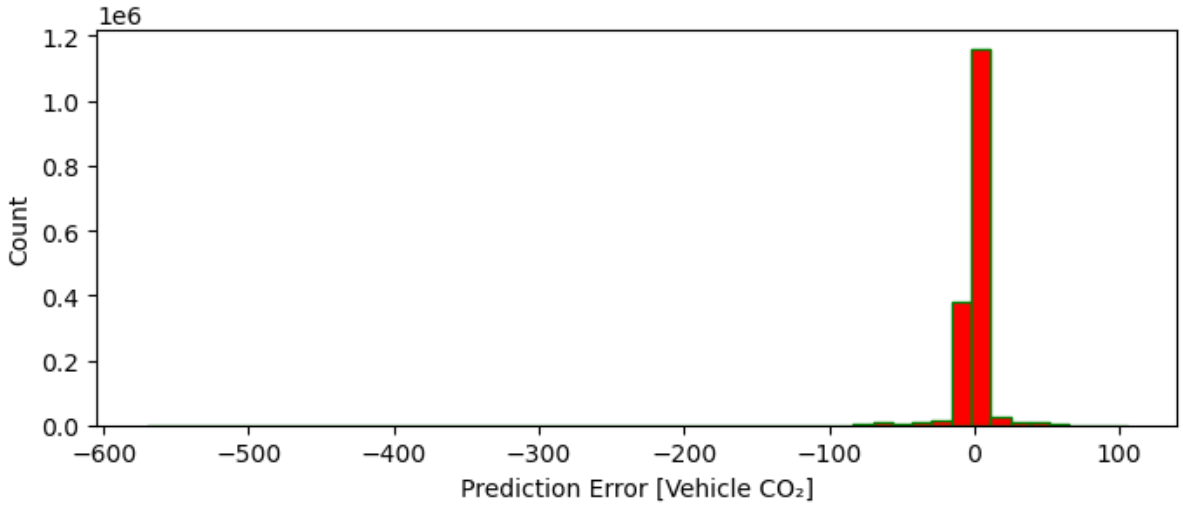


Table of Actual and Predicted Values

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

In [45]:
```
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
2624	2625
4906	4910
18986	18987
6283	6282
10901	10913
10367	10377
4170	4165
5711	5723
26498	26488
11410	11402
3626	3629
8460	8470
9804	9796
0	0
8047	8051
12280	12248
3778	3768
2565	2558
5781	5794
0	0
2801	2790
0	0
10559	10562
0	0
0	0
0	0

3839	3839
7380	7437
9027	9028
15041	15055
6580	6585
29976	29985
3060	3070
7682	7646
11749	11751
2622	2625
54890	55009
0	0
5286	5290
6730	6726
7842	7841
5286	5270
2624	2625
0	0
0	0
15203	15218
7380	7437
13073	13077
0	0
5286	5290
29106	29113
42042	42224
7854	7865

0	0
0	0
5186	5189
13626	13635
0	0
0	0
2299	2302
17025	17032
0	0
7577	7586
0	0
5229	5235
0	0
5309	5304
0	0
0	0
22117	22137
3362	3373
4252	4258
0	0
3702	3698
0	0
8927	8935
0	0
5984	5980
23858	23864
0	0

3620	3629
17230	17246
0	0
4877	4885
0	0
2624	2625
9157	9170
4249	4257
4541	4536
0	0
0	0
17056	17058
0	0
3815	3815
0	0
0	0
5269	5282
5286	5290
6985	6980
2236	2232

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.

In [ ]: