FleetGeneration

March 29, 2021

1 Fleet Generation

[3]:
```
import pandas as pd
```

[81]:
```
import sys
import os
import numpy as np
```

[3]:
```
sys.path.append(os.path.join(*os.getcwd().split('/')[:3], 'midas-applied-ds',␣
↪'Data'))
```

[4]:
```
import glob
```

[5]:
```
glob.glob(os.path.join(*os.getcwd().split('/')[1:3], '*'))
```

[5]:
```
[]
```

[6]:
```
glob.glob('/home/aysola/midas-applied-ds/Data/Processed/ICE_trips/all*')
```

[6]:
```
['/home/aysola/midas-applied-ds/Data/Processed/ICE_trips/alltrips.csv',
 '/home/aysola/midas-applied-
ds/Data/Processed/ICE_trips/alltrips_unprocessed.csv',
 '/home/aysola/midas-applied-ds/Data/Processed/ICE_trips/alltrips2.csv',
 '/home/aysola/midas-applied-
ds/Data/Processed/ICE_trips/alltrips_with_weight_and_disp.csv']
```

[57]:
```
data = pd.read_csv('/home/aysola/midas-applied-ds/Data/Processed/ICE_trips/
↪alltrips_with_weight_and_disp.csv')
```

[8]:
```
data.shape
```

[8]:
```
(11804, 8)
```

[58]:
```
data
```

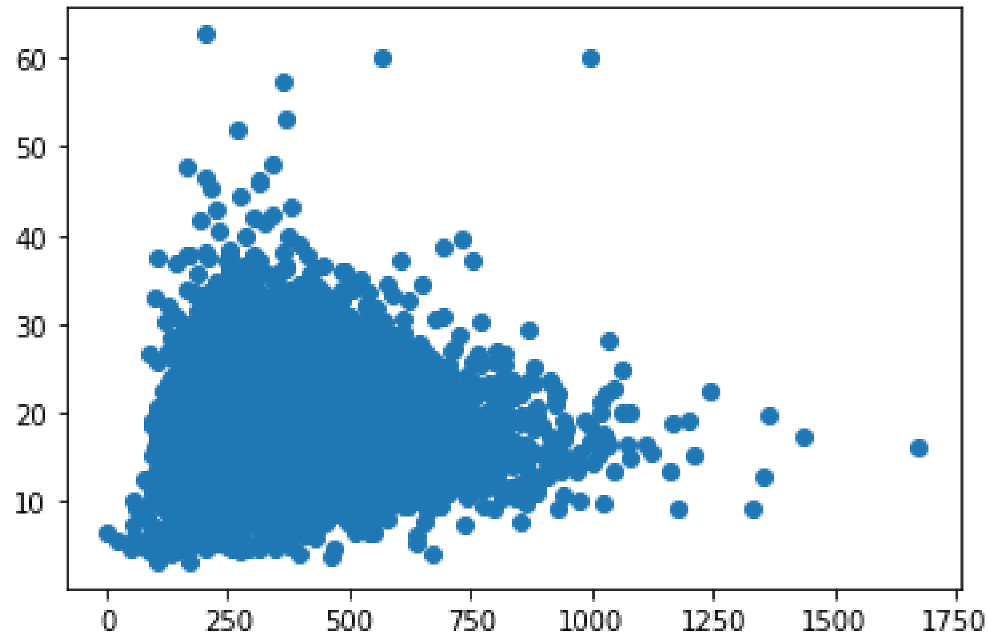
[58]:
```
       Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1  TripId  VehId  Aggressivity  \
0               0             0               0       0    123    475.645508
```

```
1            1          1            1      1    135   269.540959
2            2          2            2      2    521   459.151574
3            3          3            3      3    259   401.982744
4            4          4            4      4    575   266.112770
…          …          …          …      …      …       …
17540    17540      17540        17540  17540    266   409.468672
17541    17541      17541        17541  17541    282   307.183274
17542    17542      17542        17542  17542    244   274.127964
17543    17543      17543        17543  17543    528   301.006582
17544    17544      17544        17544  17544     12   300.598065

       Aggressiveness  Fuel Consumed[L]  Distance[km]  Weight  Displacement  \
0       242212.485095          0.895646      9.225222  2500.0           1.8
1       101741.261711          0.441901      2.223611  3500.0           2.5
2       199787.209526          0.000000      2.603500  4500.0           3.5
3       145957.042566          0.000000      8.514889  3500.0           2.7
4       117044.120748          1.008958      6.323556  4000.0           2.4
…                 …               …             …      …             …
17540    48638.660755          0.000000     39.254278  3500.0           2.4
17541   121877.568423          0.000000      6.124056  3500.0           2.5
17542   148747.893217          0.531646      3.259972  3500.0           2.5
17543   173741.356366          0.000000     10.645278  4500.0           3.3
17544   113279.279566          1.012153      8.735472  2500.0           1.8

       Fuel Economy[mpg]
0              24.227270
1              11.835816
2                    inf
3                    inf
4              14.741860
…                    …
17540                inf
17541                inf
17542          14.422995
17543                inf
17544          20.300389

[17545 rows x 12 columns]
```

[10]:
```python
import matplotlib.pyplot as plt
```

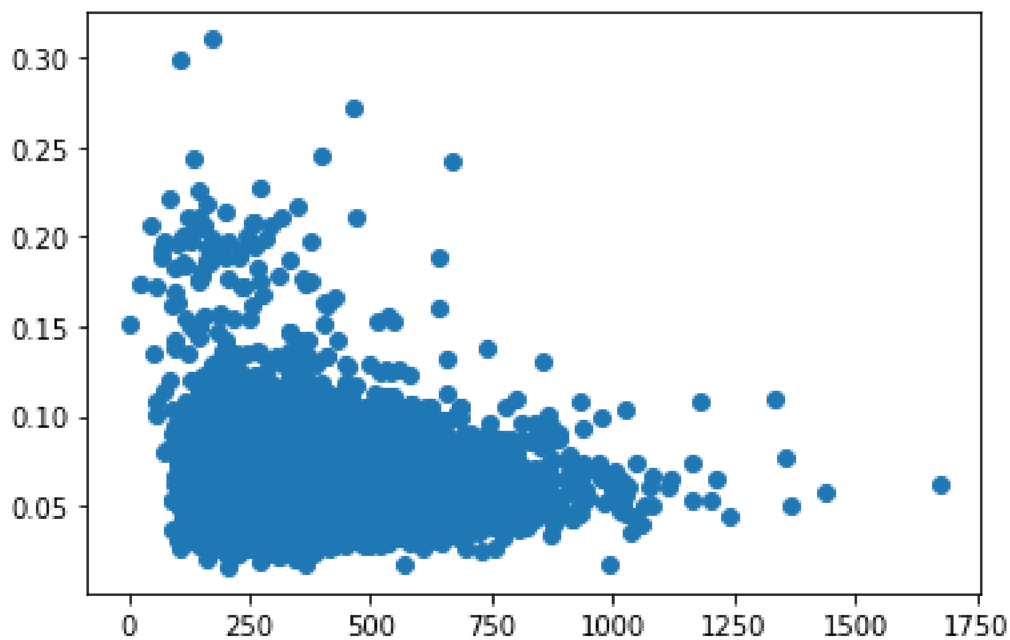
[11]:
```python
plt.scatter(data['Aggressivity'], data['Fuel Economy[mpg]'])
```

[11]: <matplotlib.collections.PathCollection at 0x2b41d702ab50>



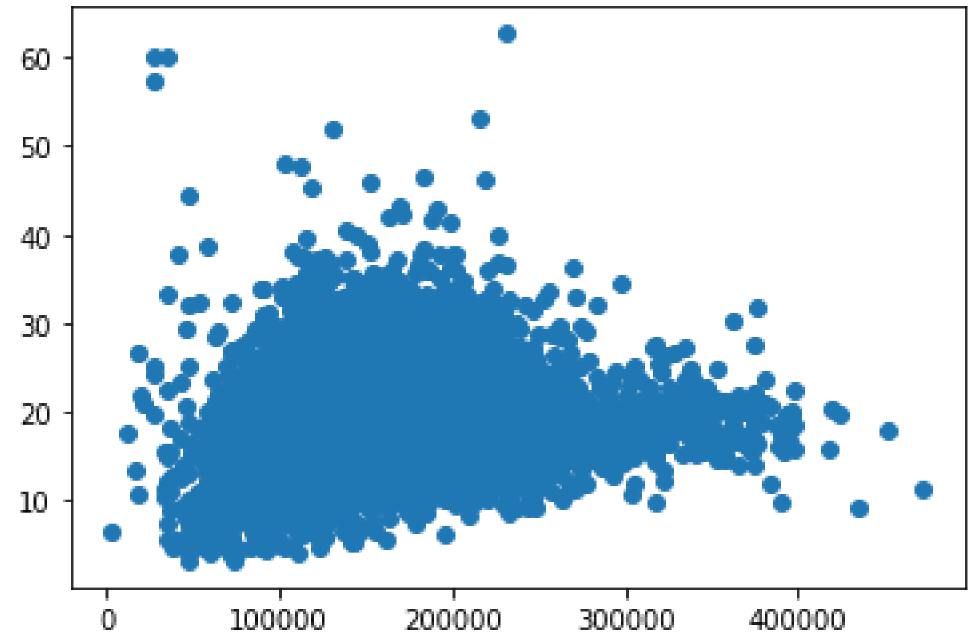

```
[12]: plt.scatter(data['Aggressivity'], (1/data['Fuel Economy[mpg]']))
```

```
[12]: <matplotlib.collections.PathCollection at 0x2b41d93b5490>
```



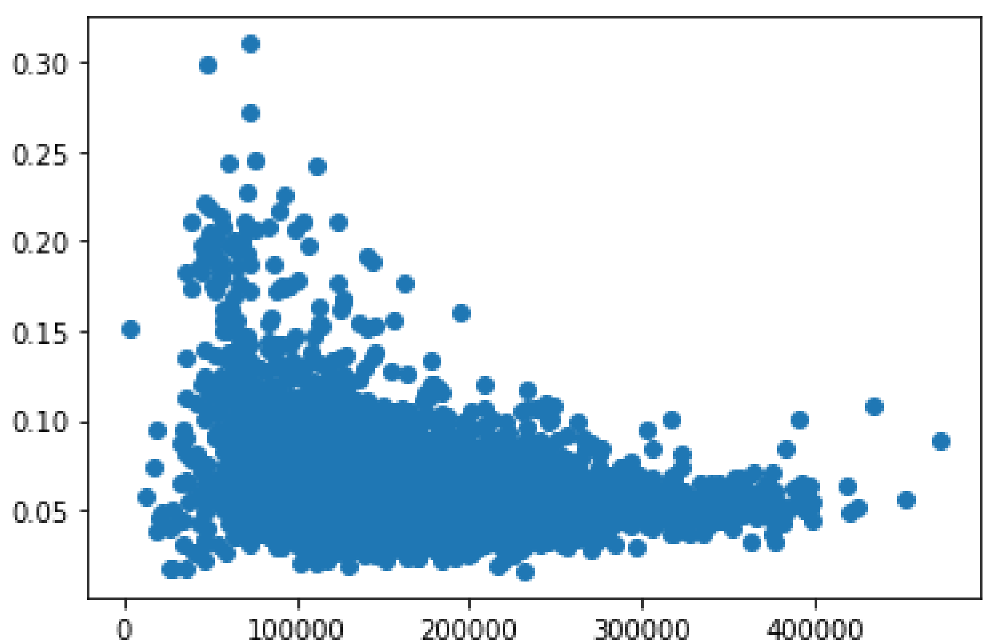
```
[13]: plt.scatter(data['Aggressiveness'], (data['Fuel Economy[mpg]']))
```

```
[13]: <matplotlib.collections.PathCollection at 0x2b41d9350750>
```




```
[14]: plt.scatter((data['Aggressiveness']), (1/data['Fuel Economy[mpg]']))
```

```
[14]: <matplotlib.collections.PathCollection at 0x2b41d9494cd0>
```



[15]:
```python
from sklearn.linear_model import LinearRegression
```

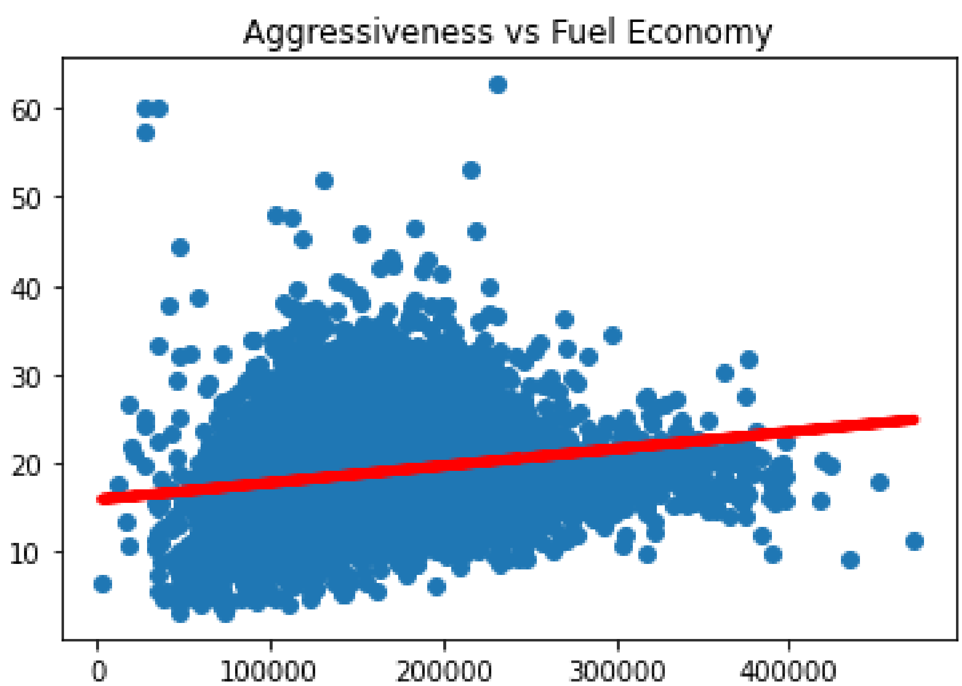
[23]:
```python
economy = LinearRegression().fit(np.array(data['Aggressiveness']).
    ↪reshape(-1,1), data['Fuel Economy[mpg]'])
```

[36]:
```python
pred = economy.predict(np.array(data['Aggressiveness']).reshape(-1,1))
squared_resid = (data['Fuel Economy[mpg]'] - pred)**2
variance = (data['Fuel Economy[mpg]']-data['Fuel Economy[mpg]'].mean())**2
r_squared = 1 - (squared_resid.sum()/variance.sum())
```

[37]:
```python
r_squared
```

[37]: 0.03349691796949994

[43]:
```python
plt.scatter(data['Aggressiveness'], (data['Fuel Economy[mpg]']))
plt.plot(data['Aggressiveness'], pred, color='red', linewidth=4.0)
plt.title('Aggressiveness vs Fuel Economy')
plt.show()
```

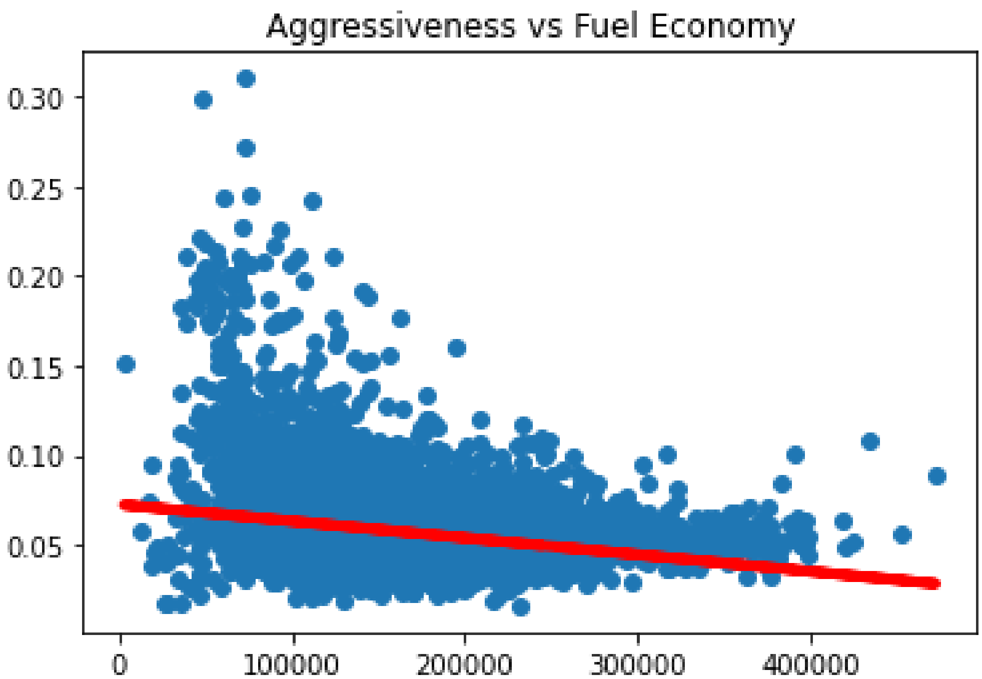


[48]:
```python
## Now for Consumption:
consumption = LinearRegression().fit(np.array(data['Aggressiveness']).
    ↪reshape(-1,1), (1/data['Fuel Economy[mpg]']))
```

```
pred = consumption.predict(np.array(data['Aggressiveness']).reshape(-1,1))
squared_resid = ((1/data['Fuel Economy[mpg]']) - pred)**2
variance = ((1/data['Fuel Economy[mpg]'])-(1/data['Fuel Economy[mpg]']).
 ↪mean())**2
r_squared = 1 - (squared_resid.sum()/variance.sum())
print(r_squared)
```

0.055417986635833416

[47]:
```
plt.scatter(data['Aggressiveness'], (1/data['Fuel Economy[mpg]']))
plt.plot(data['Aggressiveness'], pred, color='red', linewidth=4.0)
plt.title('Aggressiveness vs Fuel Economy')
plt.show()
```

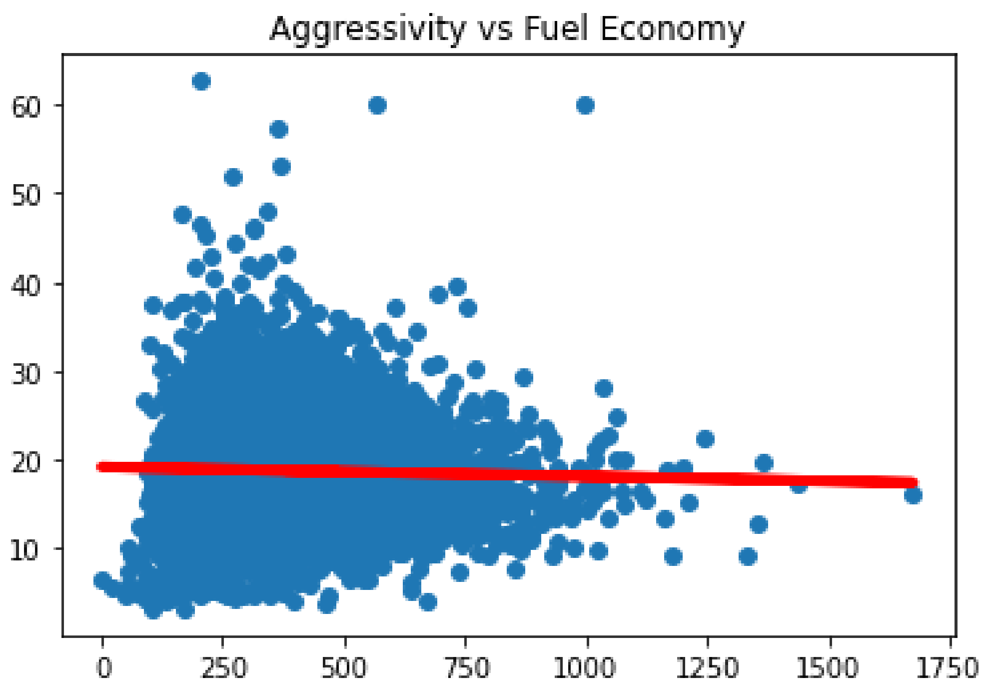


[ ]:
```
# Now for aggresivity:
```

[57]:
```
#economy
consumption = LinearRegression().fit(np.array(data['Aggressivity']).
 ↪reshape(-1,1), (data['Fuel Economy[mpg]']))
pred = consumption.predict(np.array(data['Aggressivity']).reshape(-1,1))
squared_resid = ((data['Fuel Economy[mpg]']) - pred)**2
variance = ((data['Fuel Economy[mpg]'])-(data['Fuel Economy[mpg]']).mean())**2
r_squared = 1 - (squared_resid.sum()/variance.sum())
print(r_squared)
```

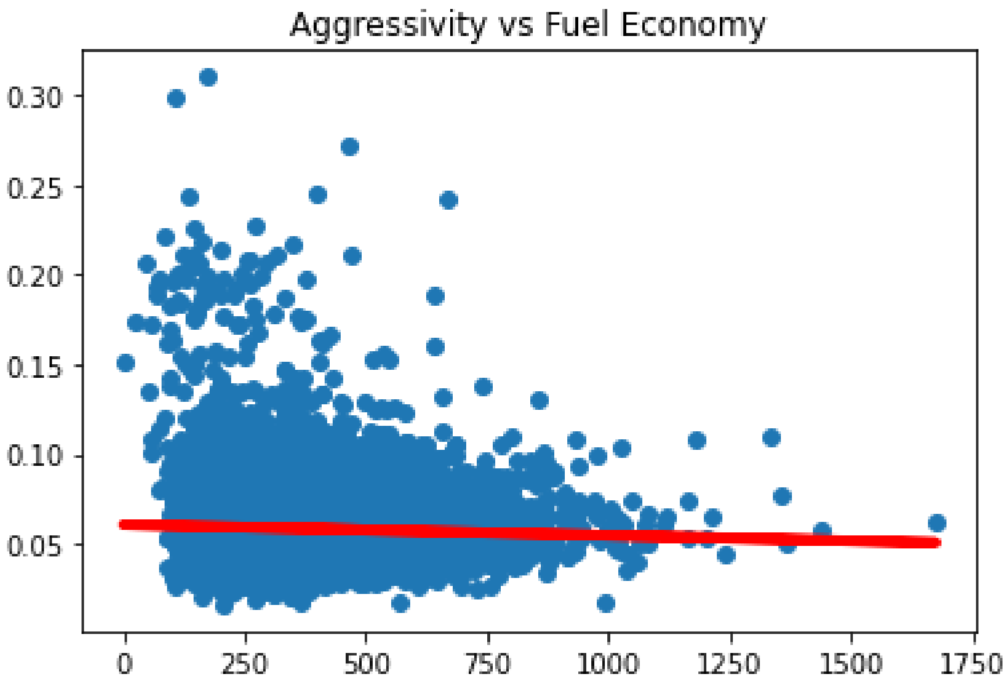
```
plt.scatter(data['Aggressivity'], (data['Fuel Economy[mpg]']))
plt.plot(data['Aggressivity'], pred, color='red', linewidth=4.0)
plt.title('Aggressivity vs Fuel Economy')
plt.show()
```

0.000736005666565176



[142]:
```
#consumption
consumption = LinearRegression().fit(np.array(data['Aggressivity']).
    ↪reshape(-1,1), (1/data['Fuel Economy[mpg]']))
pred = consumption.predict(np.array(data['Aggressivity']).reshape(-1,1))
squared_resid = ((1/data['Fuel Economy[mpg]']) - pred)**2
variance = ((1/data['Fuel Economy[mpg]'])-(1/data['Fuel Economy[mpg]']).
    ↪mean())**2
r_squared = 1 - (squared_resid.sum()/variance.sum())
print(r_squared)
plt.scatter(data['Aggressivity'], (1/data['Fuel Economy[mpg]']))
plt.plot(data['Aggressivity'], pred, color='red', linewidth=4.0)
plt.title('Aggressivity vs Fuel Economy')
plt.show()
```

0.0016599441161987416




```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

1.1 What about fitting a SUPER SIMPLE neural network?

```
[206]: import torch
       import torch.nn as nn
```

```
[211]: # lets fit a really stupid NN:
       # Aggressiveness -> 5 nodes -> 10 nodes -> 1 node (output)
       # class Squish(nn.Module):
       #     def __init__(self):
       #         return self
       #     def forward()

       # model = nn.Sequential(
       #     nn.Linear(1, 5),
       #     nn.ReLU(),
       #     nn.Linear(5,10),
       #     nn.ReLU(),
       #     nn.Linear(10,1)
```

```
# )

# model = nn.Sequential(
#     nn.Linear(1,10),
#     nn.ReLU(),
#     nn.Linear(10,20),
#     nn.ReLU(),
#     nn.Linear(20,40),
#     nn.ReLU(),
#     nn.Linear(40,20),
#     nn.ReLU(),
#     nn.Linear(20,10),
#     nn.ReLU(),
#     nn.Linear(10,5),
#     nn.ReLU(),
#     nn.Linear(5,1),
# )


# model = nn.Sequential(
#     nn.Linear(1,10),
#     nn.ReLU(),
#     nn.Linear(10,1),
# )

model = nn.Sequential(
    nn.Linear(1,32),
    nn.ReLU(),
    nn.Linear(32,64),
    nn.ReLU(),
    nn.Linear(64,1)
)
```

[212]:
```
X = torch.tensor(np.array(data['Aggressiveness']).reshape(-1,1), dtype=torch.
 ↪float32)
y = torch.tensor(np.array((1/data['Fuel Economy[mpg]'])).reshape(-1,1),␣
 ↪dtype=torch.float32)
```

[213]:
```
def train(X, y, epochs = 500):
    optimizer = torch.optim.Adagrad(model.parameters())
    for i in range(epochs):
        preds = model.forward(X)
        loss = ((y-preds)**2).sum()
        with torch.no_grad():
            optimizer.zero_grad()
            if i%100 == 0:
                print(loss.item())
```

```
            loss.backward()
            optimizer.step()
```

[210]:

```
train(X,y)
```

```
nan
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-210-7e0988a39aa5> in <module>
----> 1 train(X,y)

<ipython-input-209-5ce2b839807c> in train(X, y, epochs)
      8             if i%100 == 0:
      9                 print(loss.item())
---> 10             loss.backward()
     11             optimizer.step()
     12 

~/anaconda3/envs/nvfel/lib/python3.7/site-packages/torch/tensor.py in␣
↪backward(self, gradient, retain_graph, create_graph, inputs)
    243                 create_graph=create_graph,
    244                 inputs=inputs)
--> 245         torch.autograd.backward(self, gradient, retain_graph,␣
↪create_graph, inputs=inputs)
    246 
    247     def register_hook(self, hook):

~/anaconda3/envs/nvfel/lib/python3.7/site-packages/torch/autograd/__init__.py i␣
↪backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables,␣
↪inputs)
    145     Variable._execution_engine.run_backward(
    146         tensors, grad_tensors_, retain_graph, create_graph, inputs,
--> 147         allow_unreachable=True, accumulate_grad=True)  #␣
↪allow_unreachable flag
    148 
    149 

KeyboardInterrupt: 
```

[348]:

```
pred = model.forward(X).detach().numpy().reshape(-1)
squared_resid = ((1/data['Fuel Economy[mpg]']) - pred)**2
variance = ((1/data['Fuel Economy[mpg]'])-(1/data['Fuel Economy[mpg]']).
↪mean())**2
```

```python
r_squared = 1 - (squared_resid.sum()/variance.sum())
print(r_squared)
```

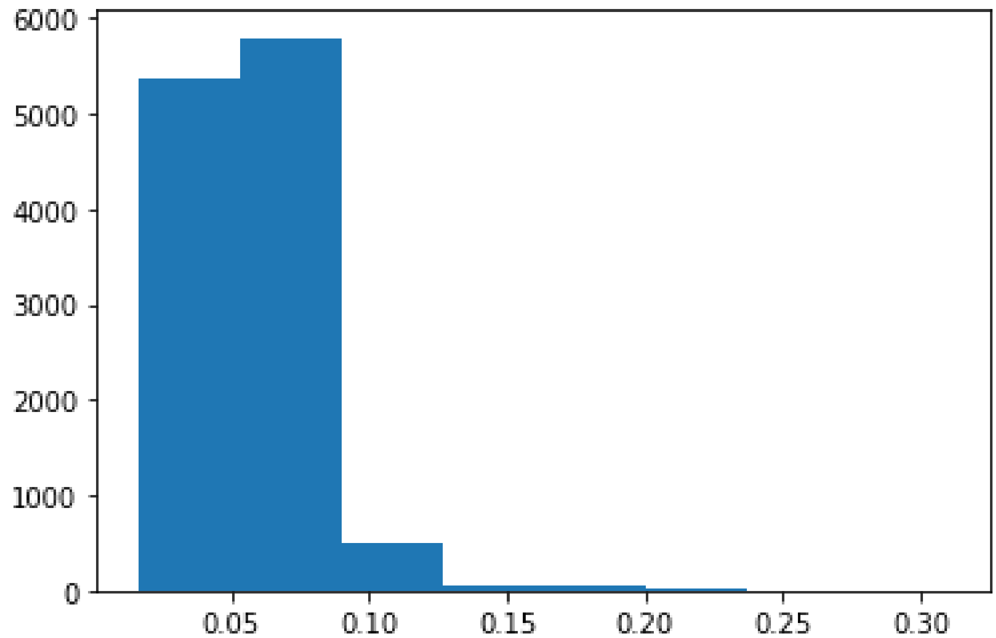
```
-25.066867691158762
```

[349]:
```python
for i in model.modules():
    if type(i) is nn.Linear:
        print(i.weight)
```

```
Parameter containing:
tensor([[-0.3123],
        [-0.0183],
        [-0.8189],
        [-0.1869],
        [ 0.0821],
        [-0.7355],
        [ 0.6740],
        [ 0.0160],
        [-0.1865],
        [-0.7160]], requires_grad=True)
Parameter containing:
tensor([[ 0.2757,  0.1795,  0.0310,  0.1598,  0.3081,  0.2170, -0.0380,  0.0204,
          0.0764,  0.1594]], requires_grad=True)
```

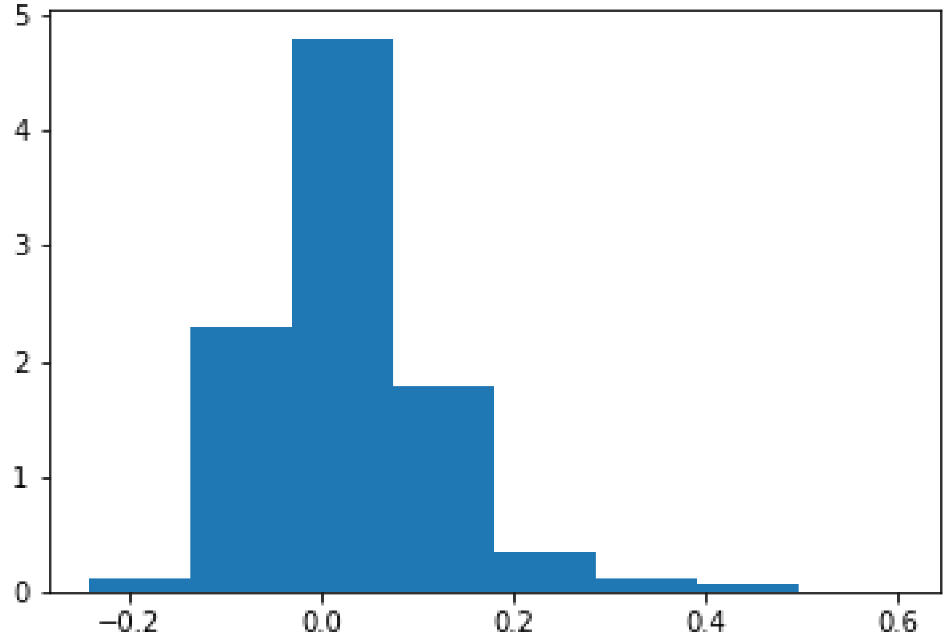
[350]:
```python
#histogram of real fuel consumption
plt.hist(y.reshape(-1).numpy(), bins=8)
```

[350]:
```
(array([5.375e+03, 5.797e+03, 5.010e+02, 6.400e+01, 4.300e+01, 1.800e+01,
        4.000e+00, 2.000e+00]),
 array([0.01594491, 0.05276915, 0.0895934 , 0.12641764, 0.16324186,
        0.2000661 , 0.23689035, 0.2737146 , 0.31053883], dtype=float32),
 <BarContainer object of 8 artists>)
```



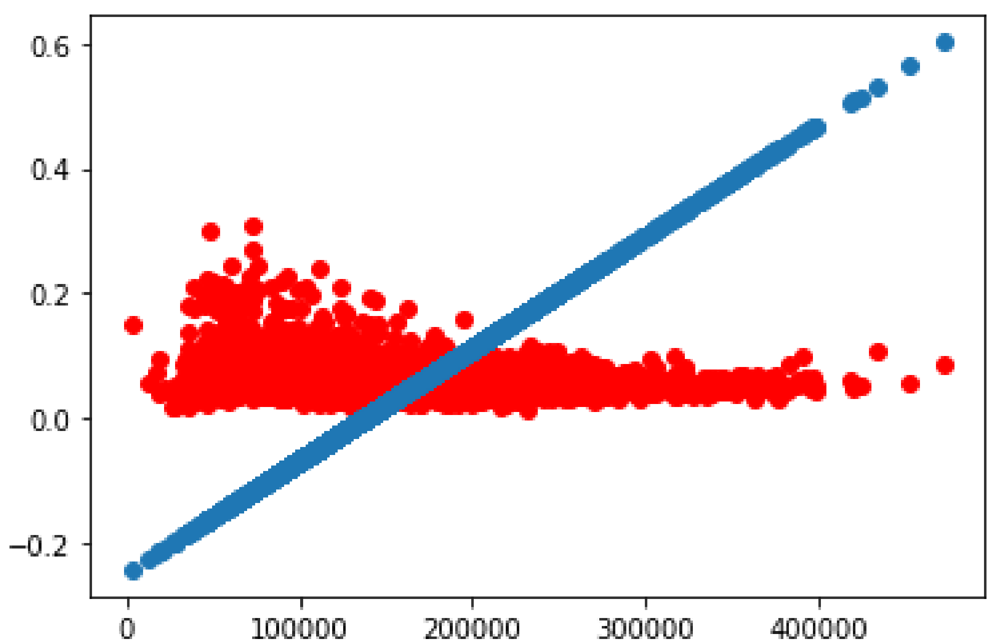
[351]:
```
plt.hist(pred,bins=8, density=True)
# plt.hist(y.reshape(-1))
```

[351]:
```
(array([0.10275641, 2.28873837, 4.80225639, 1.77094243, 0.33797224,
        0.11479817, 0.05378656, 0.00481671]),
 array([-0.24200058, -0.13647157, -0.03094256,  0.07458645,  0.18011546,
         0.28564447,  0.39117348,  0.4967025 ,  0.6022315 ], dtype=float32),
 <BarContainer object of 8 artists>)
```



[356]:

```
# plt.scatter(data['Aggressivity'], (1/data['Fuel Economy[mpg]']))
# plt.plot(data['Aggressivity'], pred, color='red', linewidth=4.0)
# plt.title('Aggressivity vs Fuel Economy')
plt.scatter(X.reshape(-1).numpy(), (y.numpy().reshape(-1)), color='red')
plt.scatter(X.reshape(-1).numpy(), pred)
plt.show()
```



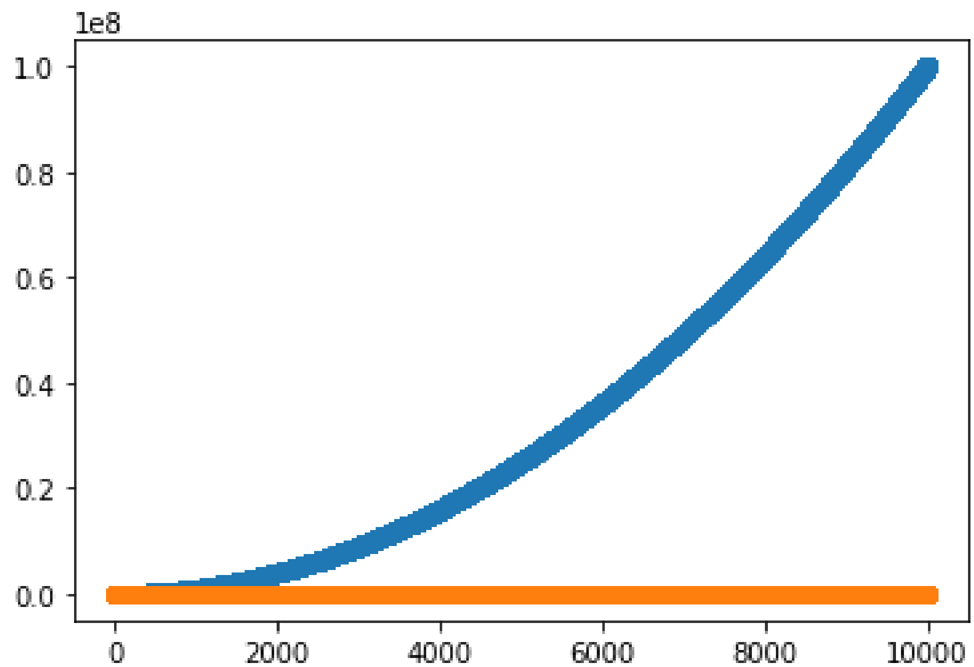
```
[214]: X_ = torch.tensor(np.linspace(0, 10000, 100001).reshape(-1,1), dtype=torch.
         ↪float32)
       y_ = X_**2
```

```
[216]: train(X_, y_, epochs=300)
```

```
2.0000267802045422e+20
1.9857836186174017e+20
1.9639740578125578e+20
```

```
[215]: pred = model.forward(X_)
       plt.scatter(X_, y_)
       plt.scatter(X_, pred.detach().numpy())
```

```
[215]: <matplotlib.collections.PathCollection at 0x2b76fc854310>
```




```
[ ]:
```

```
[ ]:
```

2 Fleet Generation with Normal Distributions:

2.1 Fleet Generation Qualms/Issues:

So here is the thing with fleet generation. We cant actually simulate any of the fuel economy data because that is scientifically unsound. It doesn't make sense for us to say: Here is the rpm/MAF etc across a trip and then get the fuel consumption from those numbers. So, we need to reframe the problem. Given a trip characteristics (vehicle weight, average speed, speed variance, displacement, and distance traveled) we will search for an equivalent trip in our Ann Arbor data and use that as a trip in our new dataset.

[61]:
```
data['score'] = 0
```

[79]:
```
data.sort_values(by=['Fuel Consumed[L]'])['Fuel Economy[mpg]']!=float('inf')
```

[79]:
```
8018      False
4087      False
11373     False
4085      False
4083      False
          …
17020      True
2751       True
6221       True
1272       True
10029      True
Name: Fuel Economy[mpg], Length: 17545, dtype: bool
```

[ ]:
```

```

[101]:
```
def search(data, trip_char:dict):
    data['score'] = 0
    for k,v in trip_char.items():
        #we'll use squared error divided by the mean value of the column for␣
 ↪our loss:
        # let T be target column and V be the value we want to closely match:
        # loss = sqrt(((T-V)/T.mean())**2)
        data['score'] += (((data[k] - v)/data[k].mean())**2)**(1/2)

    return data.sort_values(by=['score'], ascending=True).drop('score', axis=1)
```

[92]:
```
pd.concat([search(data,
       {
           'Distance[km]':2.22,
           'Displacement':2.4,
           'Weight':3000
       }
      ).iloc[:1],
```

```
search(data,
        {
            'Distance[km]':2.22,
            'Displacement':2.4,
            'Weight':3000
        }
       ).iloc[:1]
], axis=0)
```

[92]:

```
      Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1  TripId  VehId  Aggressivity  \
1667        1667          1667            1667    1667    562    263.493072
1667        1667          1667            1667    1667    562    263.493072

      Aggressiveness  Fuel Consumed[L]  Distance[km]  Weight  Displacement  \
1667   147058.005453          0.259039      2.221056  3000.0           2.4
1667   147058.005453          0.259039      2.221056  3000.0           2.4

      Fuel Economy[mpg]     score
1667          20.167774  0.000199
1667          20.167774  0.000199
```

[ ]:

```
def fleet_creation(data, fleet_char:dict):
    """
    data is where to perform searching
    fleet_char: dictionary with the following characteristics.
        size: []
    """
```

[82]:

```
np.random.normal(10, 3, size = 5)
```

[82]:

```
array([ 8.6203924 , 14.80248024, 10.46852257,  7.81959355,  8.48747206])
```

[93]:

```
np.random.randint(0, 10, 50)
```

[93]:

```
array([9, 7, 3, 3, 3, 7, 7, 7, 2, 9, 4, 7, 0, 2, 0, 3, 3, 6, 1, 5, 1, 7,
       2, 5, 3, 6, 6, 0, 3, 0, 5, 7, 9, 8, 4, 9, 8, 9, 9, 1, 9, 4, 1, 4,
       3, 9, 5, 2, 1, 0])
```

[120]:

```
def generate_fleet(data, fleet_dynamics = None):
    """
    fleet_dynamics is a dict of fleet characteristics.
    keys marked as (DIST) are distributions and are a tuple
    default_keys:
        size: int [1, ] the number of trips to simulate
        num_vehicles: int [1, size] the number of vehicles to create (these␣
 ↪vehicles will then be selected to make trips given size)
        Chars: A dictionary with the following key/values:
```

```
            NOT IN USE -- percent_AV: scalar [0,1] -> the number of vehicles␣
↪that are AVs (defaults at 1 for 100%. ONLY these vehicles will have changed␣
↪aggressivness values)
            NOT IN USE -- OAT: (DIST) Outside air temperature.
            NOT IN USE -- Average Speed: (DIST) Average Vehicle Speed measured␣
↪in KM/H
            NOT IN USE -- Variance Speed: (DIST) Variance of Vehicle Speed␣
↪measured in KM/H
            Vehicle Weight: (DIST) Weight of the vehicle in Kilograms
            Vehicle Displacement: (DIST) the displacement of the vehicle engine
            Distance: (DIST) the distance travelled over the trip

    """
    # first thing to do is generate our vehicles to perform search over:
    vehicles = {k:np.random.normal(v[0], v[1],␣
↪size=fleet_dynamics['num_vehicles']) for k,v in fleet_dynamics['Chars'].
↪items()}
#     print(vehicles)
    vehicles = [{k:v[i] for k,v in vehicles.items()} for i in␣
↪range(fleet_dynamics['num_vehicles'])]
#     print(vehicles)
    # vehicles is a list of dictionaries with k:v as search term and search␣
↪value.
    vehicle_mask = np.random.randint(0, fleet_dynamics['num_vehicles'],␣
↪fleet_dynamics['size'])

    ret_list = []
    for i in vehicle_mask:
        ret_list.append(search(data, vehicles[i]).iloc[:1])

    return pd.concat(ret_list)
```

[219]:
```
generate_fleet(data, {'size':10, 'num_vehicles':5,
                      'Chars':{
                               'Distance[km]':(15, 4),
                               'Displacement':(2, 1),
                               'Weight':(3000, 500)
                              }
                     })
```

[219]:
```
       Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1  TripId  VehId  Aggressivity  \
8985         8985          8985            8985    8985    203    409.677855
11618       11618         11618           11618   11618    246    360.161958
8985         8985          8985            8985    8985    203    409.677855
```

8985	8985	8985	8985	8985	203	409.677855
12230	12230	12230	12230	12230	185	328.878309
9514	9514	9514	9514	9514	266	414.561973
923	923	923	923	923	546	447.411057
8985	8985	8985	8985	8985	203	409.677855
9514	9514	9514	9514	9514	266	414.561973
9514	9514	9514	9514	9514	266	414.561973

	Aggressiveness	Fuel Consumed[L]	Distance[km]	Weight	Displacement \
8985	110360.758694	0.000000	12.976972	3500.0	3.3
11618	120154.080600	1.037098	6.999583	3000.0	2.7
8985	110360.758694	0.000000	12.976972	3500.0	3.3
8985	110360.758694	0.000000	12.976972	3500.0	3.3
12230	133100.121513	1.439362	13.624556	2500.0	1.5
9514	103186.894418	0.000000	16.662111	3500.0	2.4
923	203206.952471	1.477146	19.321667	2500.0	1.5
8985	110360.758694	0.000000	12.976972	3500.0	3.3
9514	103186.894418	0.000000	16.662111	3500.0	2.4
9514	103186.894418	0.000000	16.662111	3500.0	2.4

	Fuel Economy[mpg]
8985	inf
11618	15.875101
8985	inf
8985	inf
12230	22.264691
9514	inf
923	30.767012
8985	inf
9514	inf
9514	inf

2.2 GENERATION Pt 2:

List of all the variables we're going to have in our regression (at least somewhat comprehensive)

```
Air temperature,
Precipitation,
Weight,
Average Speed,
Speed Variance,
Displacement,
Distance[km],
Vehicle Type,
```

[197]:
```
def generate_trip_data(**kwargs):
    """
    kwargs:
```

```
        size = # of trips
        num_vehicles = # of vehicles
        prop_ICE = proportion of ICE vehicles
        prop_HEV = proportion of HEV
        prop_PHEV = prop of PHEV
        trips: DICT with trip characteristics as tuples
        cars: DICT with vehicle characteristics as tuples

    """
    size = kwargs['size']
    num_vehicles = kwargs['num_vehicles']

    #TODO: figure out how to deal with these
    # PRESENTLY Vehicle Type DEFAULTS TO ICE
#     prop_ICE = kwargs['prop_ICE']
#     prop_HEV = kwargs['prop_HEV']
#     prop_PHEV =kwargs['prop_PHEV']

    #DEFAULT TRIP AND CARS
    trips = {
        'Air Temp (units)':(0,0),
        'Precipitation (units)':(0,0),
        'Average Speed (units)':(0,0),
        'Speed Variance (units)':(0,0),
        'Distance (units)':(0,0)
            }

    cars = {
        'Weight (units)':(0,0),
        'Displacement (units)':(0,0),
            }
    #OVERWRITING DEAFAULTS

    for k,v in kwargs['trips'].items():
        trips[k] = v
    for k,v in kwargs['cars'].items():
        cars[k] = v

    car_keys = cars.keys()
    trip_keys = trips.keys()

    data = {k:[] for k in trips.keys()}
    for k in cars.keys():
        data[k] = []
    data['Vehicle Type'] = []
```

```
    vehicles = {k:np.random.normal(v[0], v[1], size=num_vehicles) for k,v in␣
↪cars.items()}
    vehicles = [{k:v[i] for k,v in vehicles.items()} for i in␣
↪range(num_vehicles)]
    vehicle_mask = np.random.randint(0, num_vehicles, size)

    trips = {k:np.random.normal(v[0], v[1], size) for k,v in trips.items()}
    trips = [{k:v[i] for k,v in trips.items()} for i in range(size)]

    # data has all the keys we just need to append for each value:
    for i, veh in enumerate(vehicle_mask):
        for k in trip_keys:
            data[k].append(trips[i][k])
        for k in car_keys:
            data[k].append(vehicles[veh][k])

        data['Vehicle Type'].append('ICE')

    return pd.DataFrame(data)
```

[218]:
```
args = {'size':10, 'num_vehicles':5,
        'trips':{'OAT':(72, 20), 'Distance':(10,1)},
        'cars':{'Displacement':(3,1), 'Weight':(3000,500)}
       }

gen = generate_trip_data(**args)
```

[199]:
```
gen
```

[199]:
```
   Air Temp  Precipitation  Average Speed  Speed Variance   Distance  \
0       0.0            0.0            0.0             0.0   8.859489
1       0.0            0.0            0.0             0.0   9.106707
2       0.0            0.0            0.0             0.0  10.023986
3       0.0            0.0            0.0             0.0   8.462334
4       0.0            0.0            0.0             0.0   9.429408
5       0.0            0.0            0.0             0.0  11.060453
6       0.0            0.0            0.0             0.0  11.295259
7       0.0            0.0            0.0             0.0   9.204781
8       0.0            0.0            0.0             0.0   9.646972
9       0.0            0.0            0.0             0.0  10.305263

          OAT       Weight  Displacement Vehicle Type
0   50.735414  2599.295624      2.550561          ICE
1  104.808715  3099.186723      4.196583          ICE
2   82.881598  2911.729838      3.638809          ICE
3   52.835531  3085.211283      3.292021          ICE
4   58.854848  3099.186723      4.196583          ICE
```

5	64.223717	2911.729838	3.638809	ICE
6	80.083409	3085.211283	3.292021	ICE
7	93.479671	2911.729838	3.638809	ICE
8	86.097323	3099.186723	4.196583	ICE
9	25.511017	3119.727099	4.028697	ICE

2.3 MANUAL FLEET CREATION:

[200]:
```
class ManualFleet():
    def __init__(self, col_names:list = ['Air Temp', 'Precipitation', 'Average␣
↪Speed', 'Speed Variance', 'Distance', 'Weight', 'Displacement', 'Vehicle␣
↪Type']):
        # set default data to nothing
        self.data = pd.DataFrame({l:[] for l in col_names})

    def reset(self):
        self.data = pd.DataFrame({l:[] for l in self.data.columns}).
↪reset_index(drop=True)

    def update(self, data:dict):
        """
        Data has:
        'Air Temp'
        'Precipitation'
        'Average Speed'
        'Speed Variance'
        'Distance'
        'Weight'
        'Displacement'
        'Vehicle Type'
        """
        self.data = pd.concat([self.data, pd.DataFrame(data)])
        return self.data
    def show(self):
        return self.data
```

[201]:
```
m = ManualFleet()
```

[202]:
```
m.show()
```

[202]:
```
Empty DataFrame
Columns: [Air Temp, Precipitation, Average Speed, Speed Variance, Distance,
Weight, Displacement, Vehicle Type]
Index: []
```

[203]:
```
m.update({'Air Temp':[10], 'Precipitation':[50], 'Average Speed':[3], 'Speed␣
↪Variance':[95], 'Distance':[0], 'Weight':[3932], 'Displacement':[8],␣
↪'Vehicle Type': 'ICE'})
```

[203]:
```
   Air Temp  Precipitation  Average Speed  Speed Variance  Distance  Weight  \
0      10.0           50.0            3.0            95.0       0.0  3932.0

   Displacement Vehicle Type
0           8.0          ICE
```

[204]:
```
m.show()
```

[204]:
```
   Air Temp  Precipitation  Average Speed  Speed Variance  Distance  Weight  \
0      10.0           50.0            3.0            95.0       0.0  3932.0

   Displacement Vehicle Type
0           8.0          ICE
```

[192]:
```
m.reset()
```

[193]:
```
m.show()
```

[193]:
```
Empty DataFrame
Columns: [Air Temp, Precipitation, Average Speed, Speed Variance, Distance,
Weight, Displacement, Vehicle Type]
Index: []
```

[ ]:
```

```