Volvo Truck Analytics

Ioannis Batsios, William Downs, Wahab Ehsan, James Polk, and Christopher Thacker

General Overview

- Christopher: Truck 2 GPS Speed Corrections
- Wahab: Time for Ideal Temperature Depending on Outside Temperature
- Ioannis: External Temperature Effects on Engine Components
- James: Time Series Analysis on CPU and Related Components
- Bill: APU Predictions by Regression

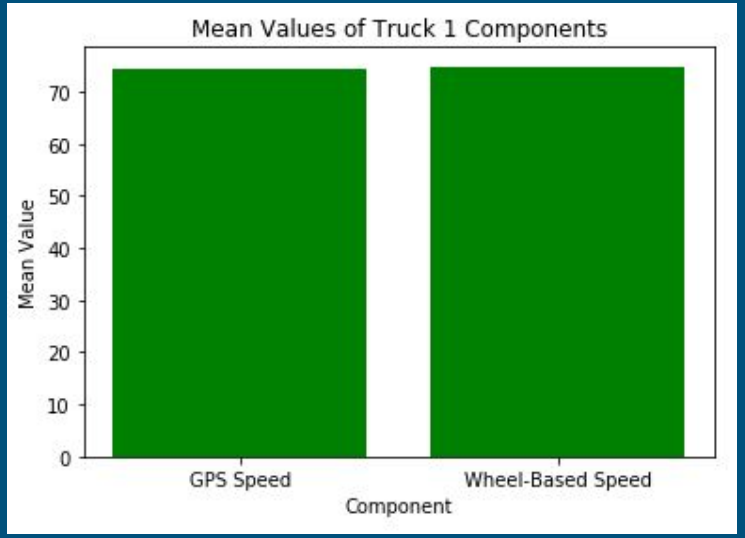
“Correcting” Truck 2’s GPS Speed Values

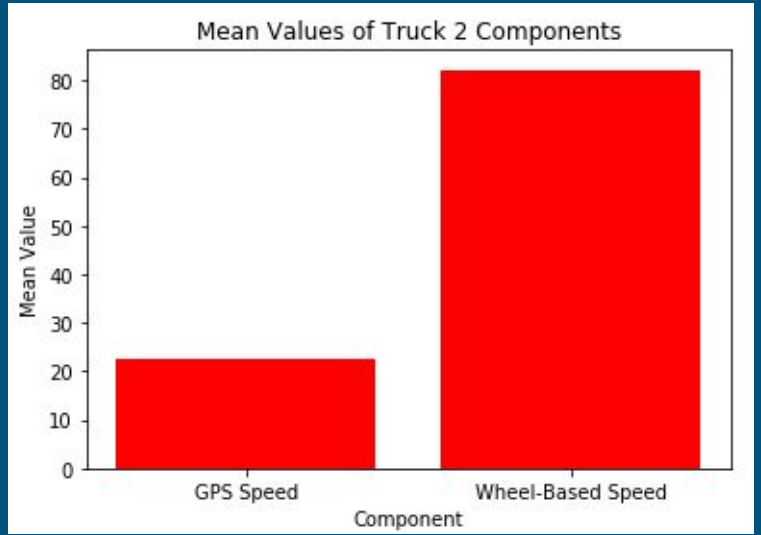
- Truck 2 has faulty GPS Speed data.
  - The component was clearly reading much lower than its Wheel-Based Speed counterpart.
  - Its faulty values are proven by the consistency in Truck 1’s data with the same components.
  - Misconfigured?
  - Different units?
  - Broken component?
- Goal:
  - Use the data and trends of Truck 1 to help predict, or correct, GPS Speed values for Truck 2.
  - Note: this was an analytical task, NOT a machine learning task.

The Data at Hand

Recall the data for both trucks.

- Like Truck 1's data, Truck 2's values should have been relatively identical.



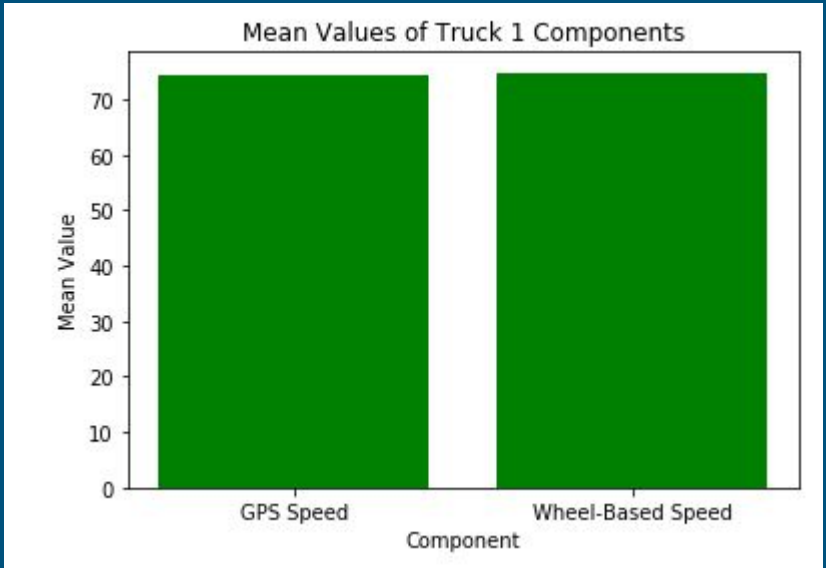


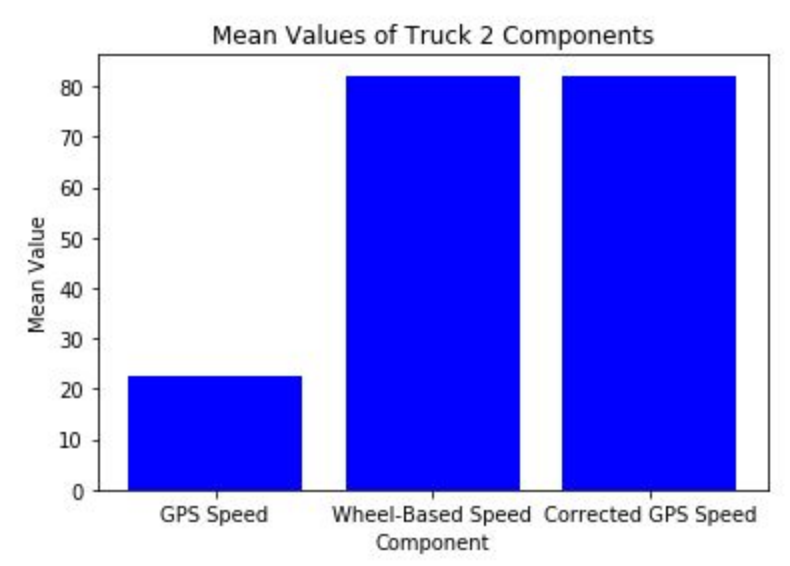
The Analytical Solution

Since Truck 1 had “better” data than Truck 2:

- Utilize Truck 1’s data to “predict” or “correct” Truck 2’s GPS Speed.
  - Given any Wheel-Based Speed value for Truck 2.
- Implementation:
  - Calculate mean difference of GPS Speed and Wheel-Based Speed for Truck 1.
    - -0.28539807628549124 (GPS - Wheel).
    - Indicates that GPS is reading slightly lower than Wheel-Based.
  - Iterate through Truck 2’s Wheel-Based Speed data.
    - Add the difference of means to each one and store the result in a new column.

Resulting Data






```
GPS Speed Mean: 74.5450583567317
Wheel-Based Speed Mean: 74.83045643301719
```

```
GPS Speed Mean: 22.685516192007974
Wheel-Based Speed Mean: 82.13312230404554
Corrected GPS Speed Mean: 81.84772422776032
```

Time to Get to the Ideal Temperature

- Time for Engine Oil Temperature to get to ‘Ideal Temperature’ based on the Outside Temperature.
- Ideal Temperature as Average temperature of oil.
- Outside Temperature as Average Temperature for the day.
- Supervised approach with Linear Regression.

Problems and Possible Solutions

- Time from start of truck to ‘ideal temperature’.
- Split data into when trucks started and when finished ‘session’.
- Truck 2 data was majority already ‘started and running’ data.
- Not all data available was from start of truck ‘session’.
- There were 45 ‘sessions’ but only 7 valid ones with enough temperature difference between start to ‘ideal’ and long enough ‘session’.

Problems and Possible Solutions (cont.)

- Made function to find time between rows.
- Found average oil temperature for each ‘session’ for ideal temperature.
- Filtered and ignored data with sessions less than 300 seconds and temperature difference of less than 30 degrees C.
- Was able to find average outside temperature using divide by day function.

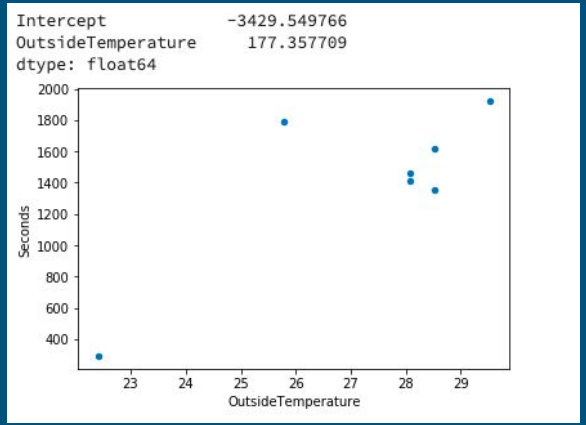
```
Time (DateTime)
08/05/2019    29.521704
08/06/2019    28.527456
08/07/2019    28.066670
08/08/2019    25.790046
08/09/2019    27.893517
08/10/2019    19.213840
08/12/2019    22.412655
Name: Outside Air Temperature (C), dtype: float64
```

Findings

- Using the function, Seconds took till ideal temp ~ Outside Temperature, I was able to find out the following:

```
lm = smf.ols(formula='Seconds ~ OutsideTemperature', data=df_data).fit()
```

- Interesting correlation between the variables.
- Not as expected, Took longer when warmer Temperature.

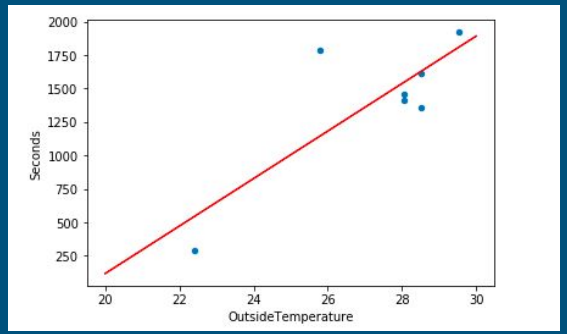


Prediction using Linear Regression

- Using statsmodels, was able to figure out the coefficients.
- Seconds = (177.3577 * Outside Temperature) - 3429.54977
- Using 20 degrees, 25 degrees, 30 degrees Celsius, was able to get the following predictions:

```
0     117.604418
1    1004.392964
2    1891.181511
dtype: float64
```

20° C
25° C
30° C



- Plot of Least Square Line =
- Maybe truck already uses system to check outside temperature and heats up oil if colder weather?

Problem?

Is there a way to determine whether an engine part could be degrading based on its temperature or that of the temperature of other parts?

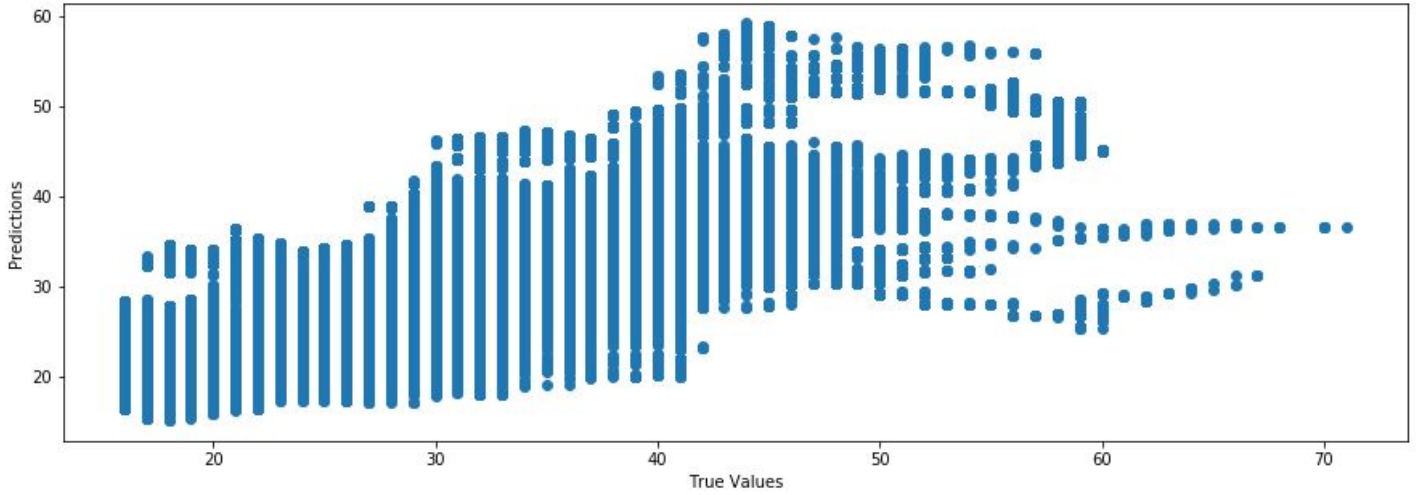
Solution:

Build a multiple linear regression model to determine what values are expected.

After making a model

```
model.score(X_test,y_test)
```

```
0.7341028591090475
```



Improving the model

Using Backward Elimination

P-values too small.

Out[91]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.735
Model:	OLS	**Adj. R-squared:**	0.735
Method:	Least Squares	**F-statistic:**	1.167e+06
Date:	Tue, 03 Dec 2019	**Prob (F-statistic):**	0.00
Time:	02:02:06	**Log-Likelihood:**	-4.8943e+06
No. Observations:	1686308	**AIC:**	9.789e+06
Df Residuals:	1686303	**BIC:**	9.789e+06
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-33.8231	0.132	-256.293	0.000	-34.082	-33.564
x1	1.1370	0.002	582.600	0.000	1.133	1.141
x2	-0.6087	0.001	-1001.456	0.000	-0.610	-0.607
x3	0.3967	0.001	272.600	0.000	0.394	0.400
x4	0.8608	0.001	1139.289	0.000	0.859	0.862

Omnibus:	347849.307	**Durbin-Watson:**	0.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1102257.968
Skew:	1.056	**Prob(JB):**	0.00
Kurtosis:	6.351	**Cond. No.**	6.23e+03

Time Series Analysis

Performed Time Series Analysis on CPU Load, Vehicle Weight, Driver Requested Torque, Outside Air Temperature, and Temperature of Air Entering Vehicle.

Problem:

Data is spotty with frequent gaps. Furthermore, data is sampled every 100 milliseconds, causing noise.

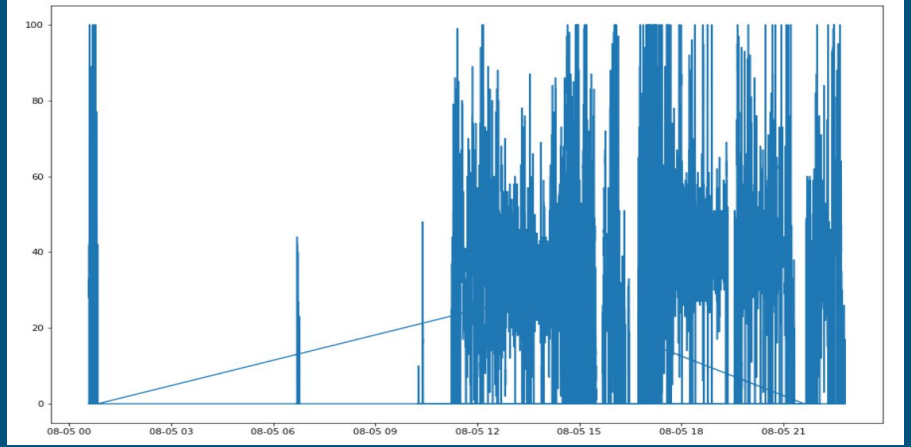
Attempted Solution

Resample data to every minute using mean.

Still frequent dropouts.

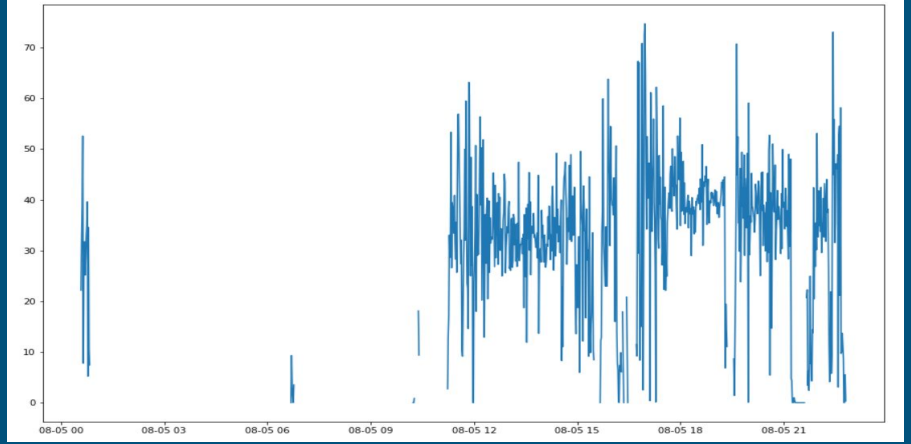
```
plt.plot(day1['Driver Requested Torque (%)'])
```

```
[<matplotlib.lines.Line2D at 0x2796037b308>]
```




```
minutes = day1.resample('1T').mean()
plt.plot(minutes['Driver Requested Torque (%)'])
```

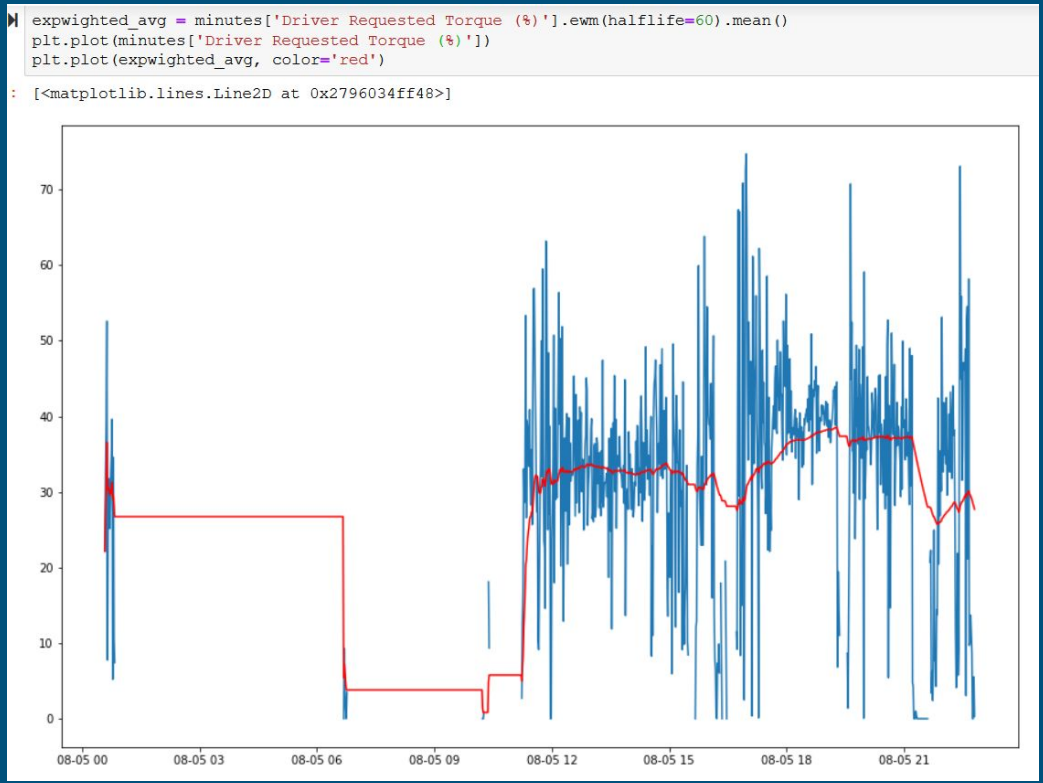
```
[<matplotlib.lines.Line2D at 0x27910eec788>]
```



Driver Requested Torque

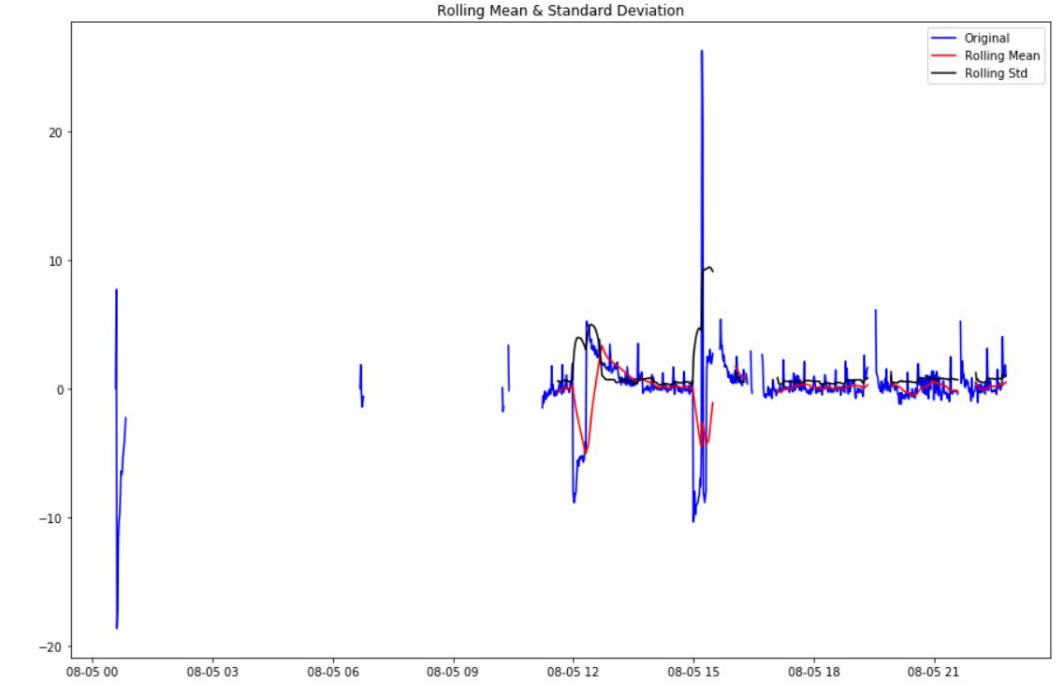
```
expwighted_avg = minutes['Driver Requested Torque (%)'].ewm(halflife=60).mean()
plt.plot(minutes['Driver Requested Torque (%)'])
plt.plot(expwighted_avg, color='red')
```

```
: [<matplotlib.lines.Line2D at 0x2796034ff48>]
```



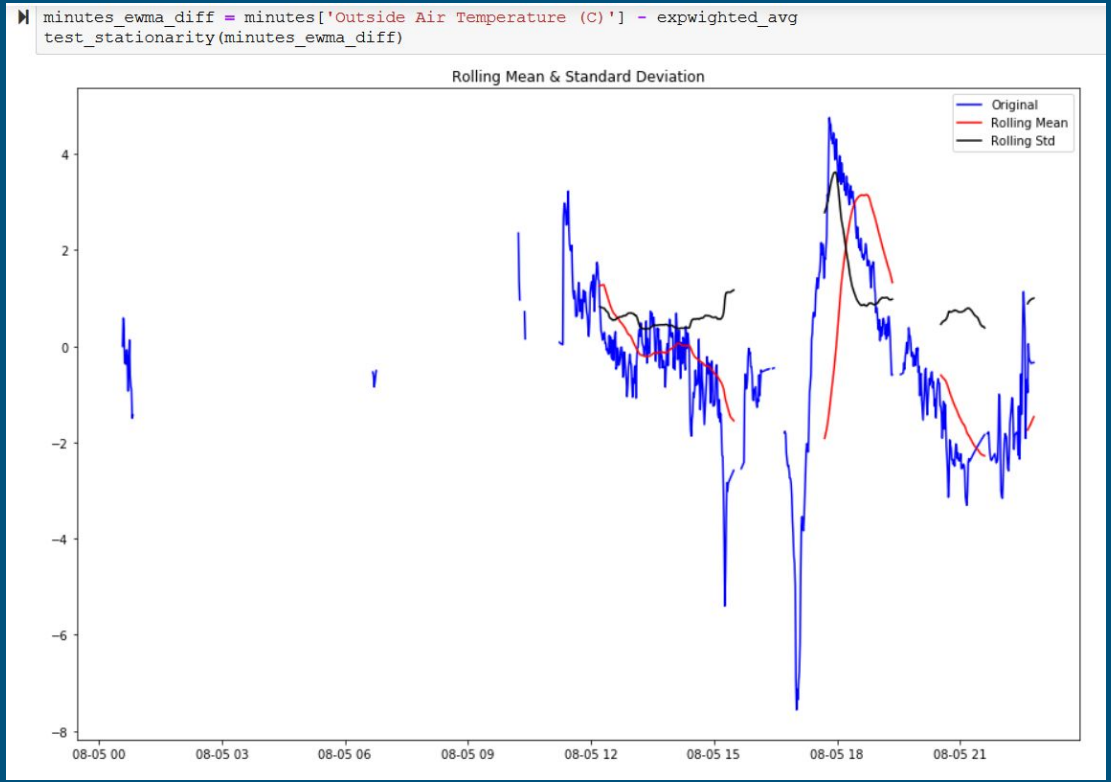
CPU Load

```
expwighted_avg = daysavg['CPU Load (%)'].ewm(halflife=24).mean()
daysavg_ewma_diff = daysavg['CPU Load (%)'] - expwighted_avg
test_stationarity(daysavg_ewma_diff)
```



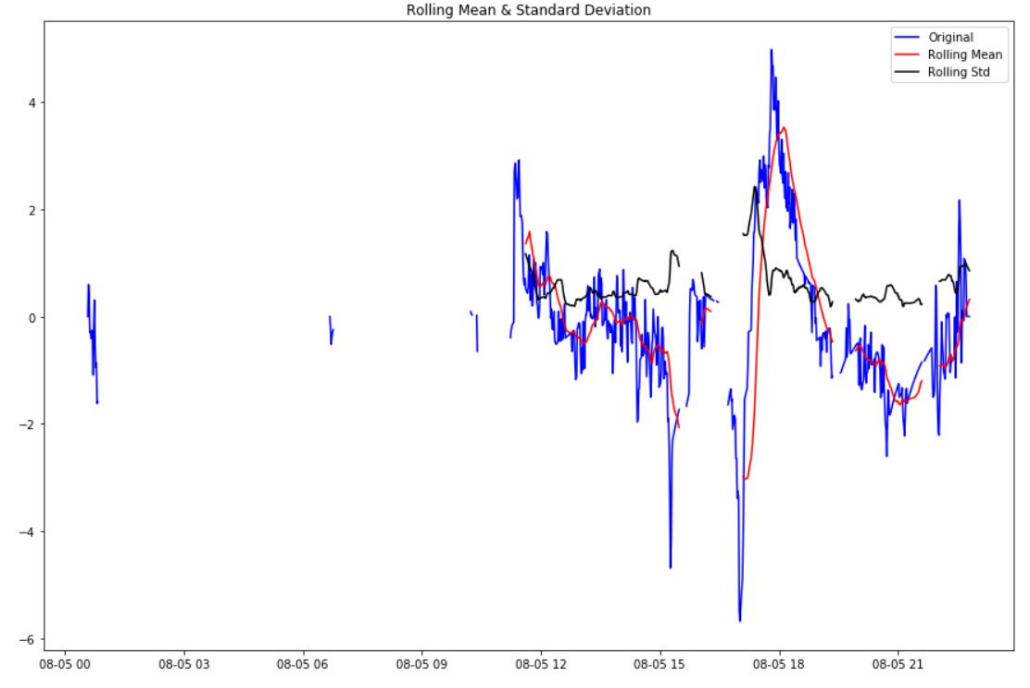
Outside Air Temperature

```
minutes_ewma_diff = minutes['Outside Air Temperature (C)'] - expwighted_avg
test_stationarity(minutes_ewma_diff)
```



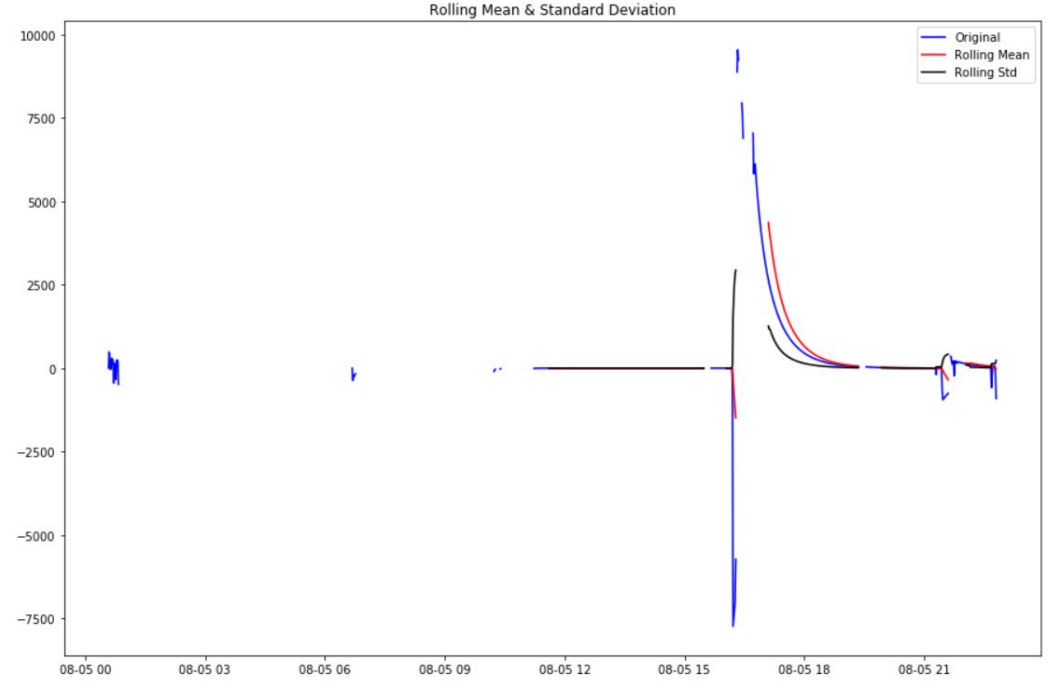
Temp. of Air Entering Vehicle

```
expwighted_avg = daysavg['Temperature of Air Entering Vehicle (C)'].ewm(halflife=24).mean()
daysavg_ewma_diff = daysavg['Temperature of Air Entering Vehicle (C)'] - expwighted_avg
test_stationarity(daysavg_ewma_diff)
```



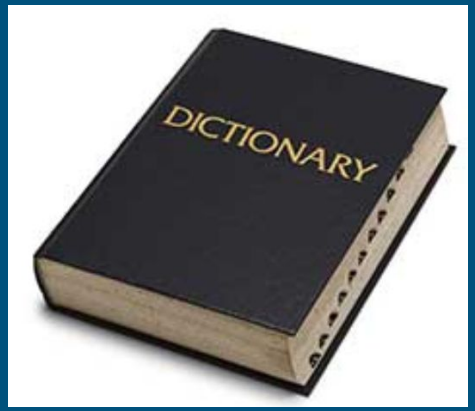
Vehicle Weight

```
expwighted_avg = daysavg['Vehicle Weight (kg)'].ewm(halflife=24).mean()
daysavg_ewma_diff = daysavg['Vehicle Weight (kg)'] - expwighted_avg
test_stationarity(daysavg_ewma_diff)
```



Bill Downs: APU prediction

+

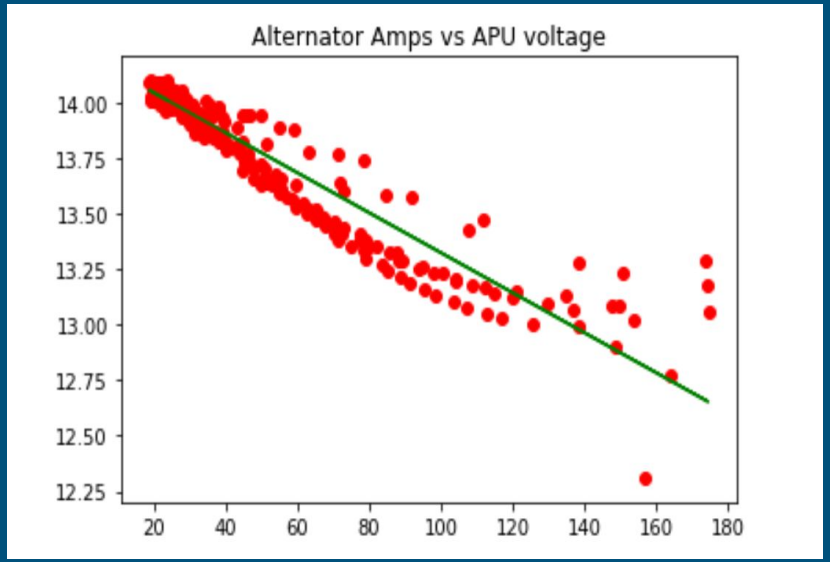


=

Maybe…..

Recall proj 2 APU

This data is related. We can use if for our model.



Idea - Binary classification
How- Logistic Regression

First I needed to add a binary column to my data.

Using my dataframe from last project cleaned the data removing any unwanted columns, which left me with the Time, APU, and alternator data.

The APU data was used to calculate a new column where if the voltage was above 14 then 1 else 0. The Volvo electrical engineers were fine with greater than 14 as charged value.

The data was resampled to every 300S for better insight of results. *The 10hz measurement wasnt good for my model.

	4649_Ch1_Alternator_250A	4649_Ch8_APU_BatteryVoltage
count	45.000000	45.000000
mean	47.888751	13.819850
std	38.458300	0.317695
min	18.789823	13.025014
25%	23.473271	13.684152
50%	28.164433	13.996532
75%	54.850180	14.037002
max	174.051498	14.093159

Classifier: Logistic regression

Why? The APU/ and Alternator data had very little outliers as theses are real time monitoring systems connected somehow to a smart switch. i.e...Lead acid batteries can explode if overcharged.

I resampled the data at 300 second and by 3000 to see the results, to maybe better understand my results. At 3000 seconds there were only 14 values in my testing set. So went with 300 which had more values at 68(20%). Which left 80% to my training. Tried a lot of different values here.

```
(68, 2) - Test(x) dataframe shape
(271, 2) - Training(x) shape
(68,) - Testing(y) shape
(271,) - Testing(y) shape
```

Scores. Confusion matrix. Precision Recall.

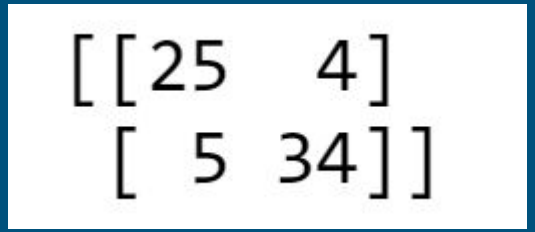
Training: 0.94
Testing: 0.86

```
1  #get some score values for 'acc
2  logReg.score(X_train,y_train)
]: 0.948339483394834

1  logReg.score(X_test,y_test)#thi
]: 0.8676470588235294
```

I liked these values because with the 10Hz data there was a lot of variation from 1 on both testing and training, to low training and high testing....

C.Matrix



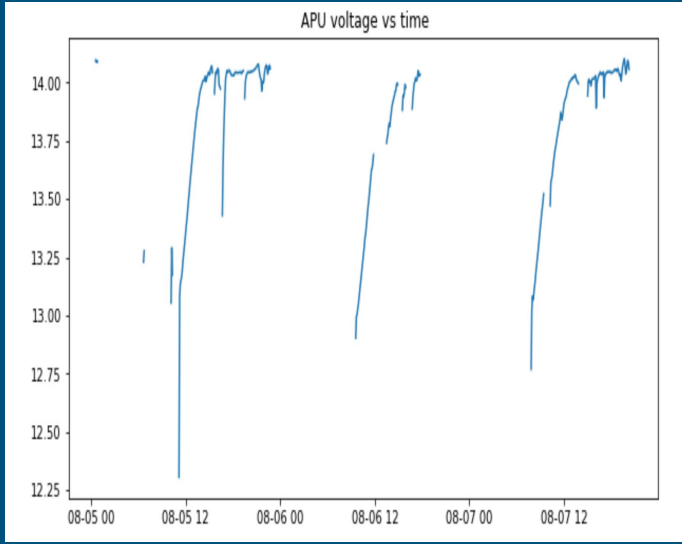
P&R

	precision	recall	f1-score	support
0	0.86	0.83	0.85	30
1	0.87	0.89	0.88	38
micro avg	0.87	0.87	0.87	68
macro avg	0.87	0.86	0.87	68
weighted avg	0.87	0.87	0.87	68

Prediction

```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0])
```

But is it believable?? Maybe. More data is probably needed.



Time	4649_Ch8_	4649_Ch1_Alternator_250A	
8/7/2019 11:15	13.7934	41.34075	array([[9.99982618e-01, 1.73824566e-05],
8/7/2019 19:30	14.08159	21.25699	[8.69124474e-03, 9.91308755e-01],
8/7/2019 16:10	13.99505	27.67186	[5.67938696e-01, 4.32061304e-01],
8/5/2019 11:50	13.28952	88.28844	[1.00000000e+00, 2.24537236e-21],
8/7/2019 18:40	14.04888	21.67571	[1.22176306e-02, 9.87782369e-01],
8/7/2019 20:10	14.08686	21.38521	[9.55600452e-03, 9.90443995e-01],
8/5/2019 16:45	13.6376	71.85345	[1.00000000e+00, 9.13247721e-16],
8/5/2019 22:35	14.06674	24.41085	[9.18741704e-02, 9.08125830e-01],
8/6/2019 9:35	12.90422	148.66	[1.00000000e+00, 9.80310780e-42],
8/5/2019 11:40	13.23158	100.4002	[1.00000000e+00, 1.87058693e-25],
8/7/2019 13:25	14.03414	24.96136	[1.36447599e-01, 8.63552401e-01],
8/5/2019 16:40	13.43022	107.6826	[1.00000000e+00, 7.64291541e-28],
8/6/2019 10:00	13.07478	106.9722	[1.00000000e+00, 1.05713032e-27],
8/6/2019 17:15	14.01223	23.58174	[5.23006501e-02, 9.47699350e-01],
8/6/2019 14:25	13.93646	29.94169	[8.87343640e-01, 1.12656360e-01],
8/7/2019 10:40	13.65863	55.28275	[1.00000000e+00, 3.35511392e-10],
8/5/2019 22:20	14.07108	21.82358	[1.34879714e-02, 9.86512029e-01],
8/5/2019 17:45	14.03608	27.73642	[5.73807170e-01, 4.26192830e-01],
8/6/2019 14:00	13.84889	39.91026	[9.99945646e-01, 5.43542615e-05],
8/7/2019 15:40	14.01707	27.38171	[5.08846606e-01, 4.91153394e-01],
8/7/2019 11:35	13.87242	35.76111	[9.98641269e-01, 1.35873109e-03],
8/7/2019 13:10	14.02427	25.28241	[1.69261536e-01, 8.30738464e-01],
8/5/2019 17:50	14.03196	26.74513	[3.85618378e-01, 6.14381622e-01],
8/5/2019 13:05	13.76529	44.09613	[9.99997968e-01, 2.03235196e-06],
8/5/2019 13:40	13.92353	30.31995	[9.14061711e-01, 8.59382885e-02],
8/5/2019 17:25	14.04727	27.22046	[4.72964583e-01, 5.27035417e-01],
8/5/2019 0:45	14.09451	24.04218	[6.95704936e-02, 9.30429506e-01],
8/7/2019 17:30	14.04748	20.92801	[6.90001308e-03, 9.93099987e-01],
8/7/2019 15:05	14.00622	34.63013	[9.96465576e-01, 3.53442357e-03],
8/7/2019 15:15	14.00354	28.96895	[7.80756646e-01, 2.19243354e-01],
8/7/2019 7:55	13.02187	153.8548	[1.00000000e+00, 1.90908913e-43],
8/6/2019 17:35	14.03098	23.04861	[3.48672123e-02, 9.65132788e-01],
8/7/2019 17:35	14.04485	21.78971	[1.33607444e-02, 9.86639256e-01],
8/6/2019 15:30	13.88244	59.31233	[1.00000000e+00, 1.71870705e-11],
8/5/2019 21:10	14.07798	23.99426	[6.78693974e-02, 9.32130603e-01],
8/7/2019 15:25	13.98732	26.63547	[3.72316476e-01, 6.27683524e-01],
8/5/2019 16:25	13.97393	24.64454	[1.13867840e-01, 8.86132160e-01],
8/7/2019 19:25	14.06612	19.87486	[3.03460431e-03, 9.96965396e-01],
8/5/2019 16:20	13.98586	21.51933	[1.12777054e-02, 9.88722295e-01],
8/7/2019 15:55	14.02369	25.08682	[1.49111214e-01, 8.50888786e-01],
8/5/2019 19:00	14.04617	24.31895	[8.71479881e-02, 9.12852012e-01],
8/5/2019 19:40	14.02389	23.80275	[6.10158271e-02, 9.38984173e-01],
8/5/2019 14:40	14.01355	22.51167	[2.35569586e-02, 9.76443041e-01],
8/5/2019 14:25	14.01467	24.21654	[8.25507516e-02, 9.17449248e-01],
8/5/2019 12:25	13.51681	63.31943	[1.00000000e+00, 6.17394284e-13],
8/7/2019 11:50	13.86059	31.53672	[9.65906829e-01, 3.40931709e-02],
8/7/2019 11:30	13.84261	38.42224	[9.99829123e-01, 1.70876717e-04],
8/7/2019 15:10	14.01411	28.85791	[7.64521341e-01, 2.35478659e-01],
8/5/2019 18:15	14.04241	24.08706	[7.40739809e-02, 9.25926019e-01],
8/5/2019 14:15	14.00966	24.38401	[9.38145695e-02, 9.06185431e-01],

MLTestdata

APU binary classification conclusion

Low volume data, as i sampled may not be enough. But questions are there now to look into.

Driver habits?

APUs performance for a particular driver?

Fleet driver-break policies?