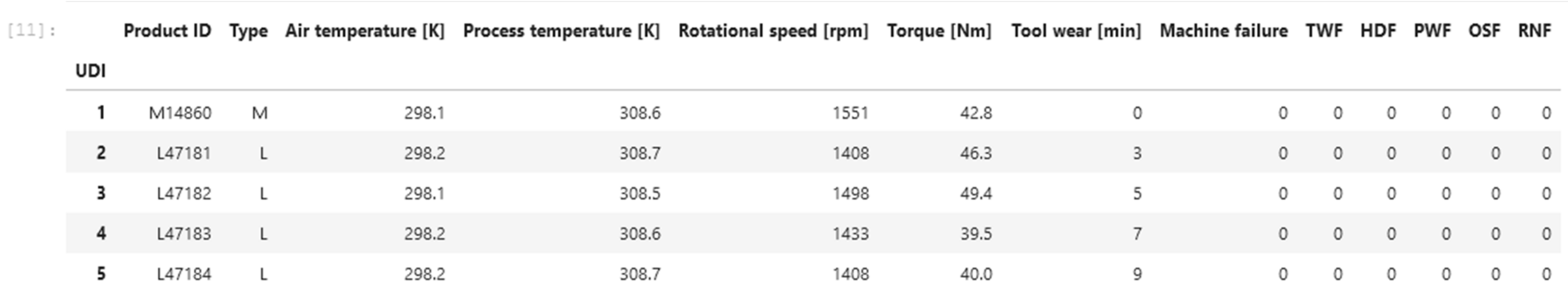
Reliability Engineering

By

D. Kelley

About This Dataset and the Assumptions made:

- This Data Set is right censored. This has been neglected due to the lack of information required.
- 3 quality indices for on machine (L,M, H). Treated as 3 different machines
- 3 features (Air temp, Torque, Tool Wear). The others were removed due to lack of linear independence
- Tool wear is just a measure of time in minutes
- There are 5 failure modes :
    - TWF – Tool Wear Failure
    - HDF-Heat D Failure ,
    - PWF-Power W Failure,
    - OSF-Overstrain Failure and
    - RNF-Random Failure –Not considered here
- Machine failure occurs if any failure mode occurs.
- The failure modes are Binary in nature. (0 for operational, and 1 for failure)
- The reliability calculations were performed

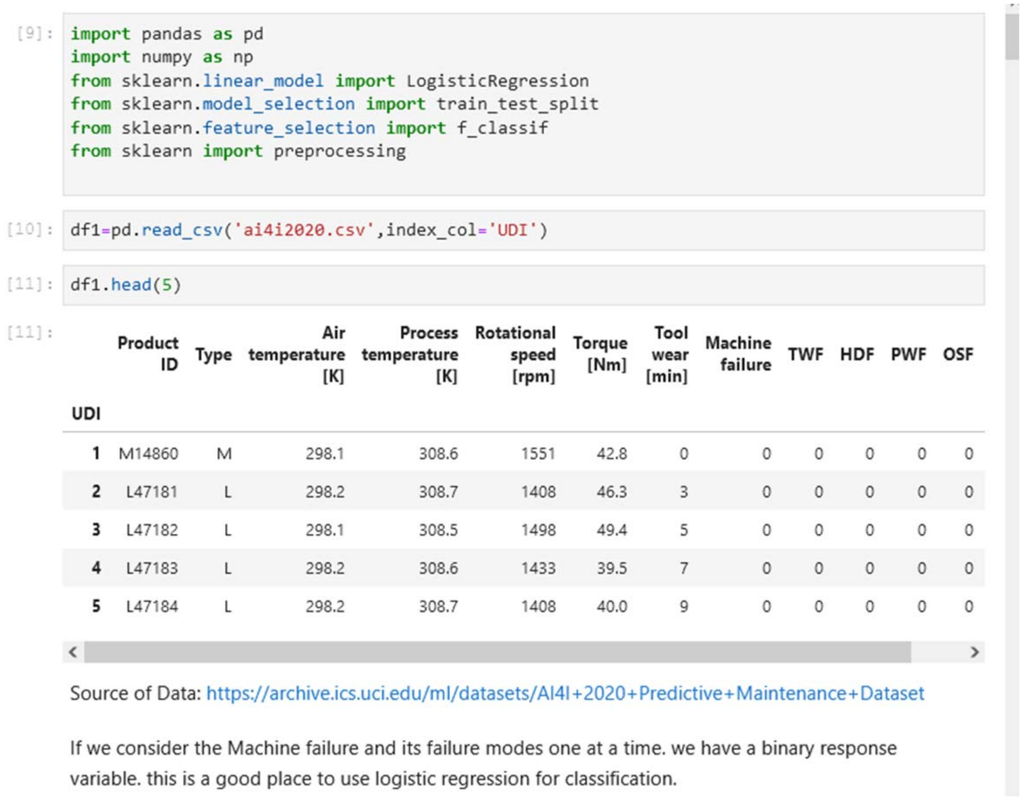


[11]:

	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
UDI													
1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0
4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0
5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0	0

Modules and Data Insertion

Shows the source of the data and modules needed



```
[9]: import pandas as pd
     import numpy as np
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split
     from sklearn.feature_selection import f_classif
     from sklearn import preprocessing
```

```
[10]: df1=pd.read_csv('ai4i2020.csv',index_col='UDI')
```

```
[11]: df1.head(5)
```

[11]:

UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF
1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0
2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0
3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0
4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0
5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0

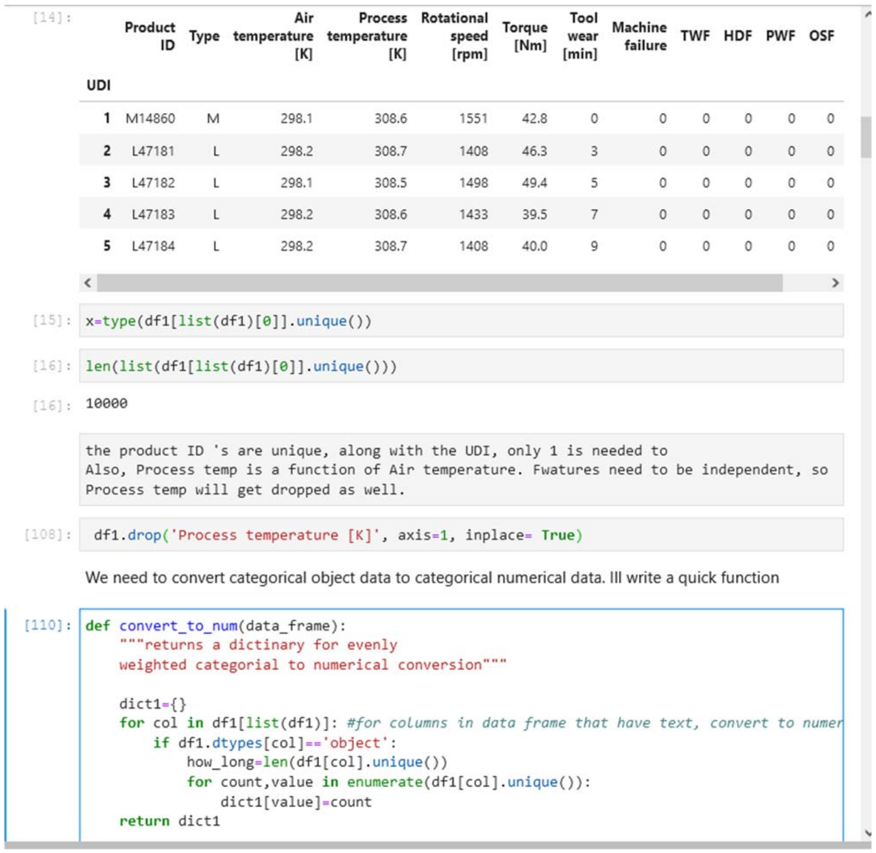
Source of Data: https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset

If we consider the Machine failure and its failure modes one at a time. we have a binary response variable. this is a good place to use logistic regression for classification.

Data Cleanup

Here unneeded information is dropped, and a function is created to generate a dictionary.

This dictionary is used to define a replacement of strings to numerical values



[14]:

UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF
1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0
2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0
3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0
4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0
5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0

[15]:
```python
x=type(df1[list(df1)[0]].unique())
```

[16]:
```python
len(list(df1[list(df1)[0]].unique()))
```

[16]:
```
10000
```

```
the product ID 's are unique, along with the UDI, only 1 is needed to
Also, Process temp is a function of Air temperature. Fwatures need to be independent, so
Process temp will get dropped as well.
```

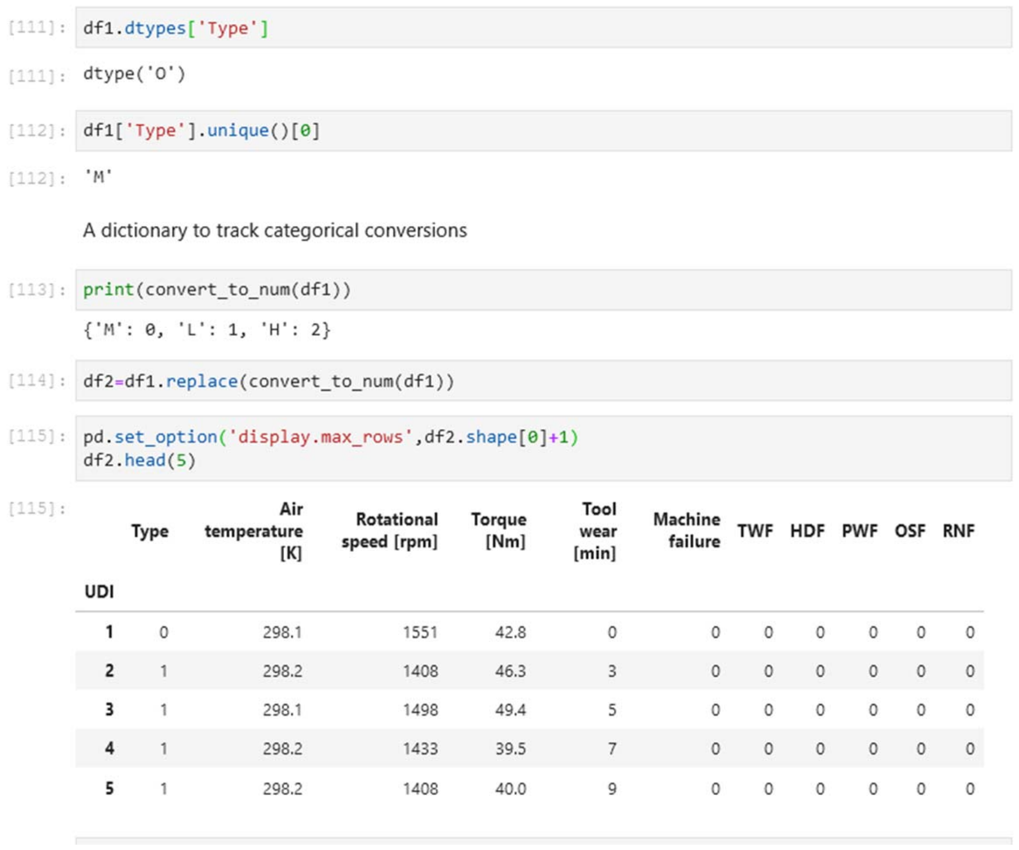
[108]:
```python
 df1.drop('Process temperature [K]', axis=1, inplace= True)
```

We need to convert categorical object data to categorical numerical data. Ill write a quick function

[110]:
```python
def convert_to_num(data_frame):
    """returns a dictionary for evenly
    weighted categorial to numerical conversion"""

    dict1={}
    for col in df1[list(df1)]: #for columns in data frame that have text, convert to numer
        if df1.dtypes[col]=='object':
            how_long=len(df1[col].unique())
            for count,value in enumerate(df1[col].unique()):
                dict1[value]=count
    return dict1
```

Minor Data Wrangling

The replace function is performed and the type of machine is now numerical. This allows the logistic function to operate.



```
[111]: df1.dtypes['Type']
[111]: dtype('O')
[112]: df1['Type'].unique()[0]
[112]: 'M'
```

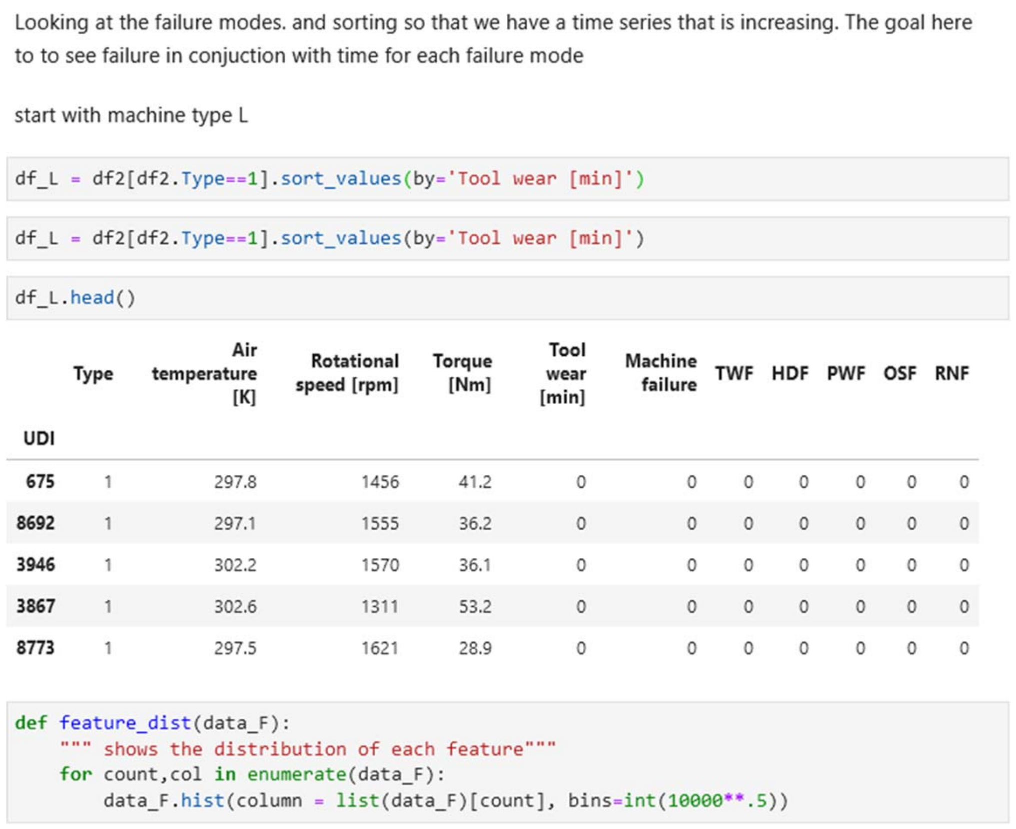
A dictionary to track categorical conversions

```
[113]: print(convert_to_num(df1))
{'M': 0, 'L': 1, 'H': 2}
[114]: df2=df1.replace(convert_to_num(df1))
[115]: pd.set_option('display.max_rows',df2.shape[0]+1)
       df2.head(5)
[115]:
```

	Type	Air temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
UDI											
1	0	298.1	1551	42.8	0	0	0	0	0	0	0
2	1	298.2	1408	46.3	3	0	0	0	0	0	0
3	1	298.1	1498	49.4	5	0	0	0	0	0	0
4	1	298.2	1433	39.5	7	0	0	0	0	0	0
5	1	298.2	1408	40.0	9	0	0	0	0	0	0

Organizing the Dataset

Working with Machine type L and increasing time



Looking at the failure modes. and sorting so that we have a time series that is increasing. The goal here to to see failure in conjuction with time for each failure mode

start with machine type L

```
df_L = df2[df2.Type==1].sort_values(by='Tool wear [min]')
```

```
df_L = df2[df2.Type==1].sort_values(by='Tool wear [min]')
```

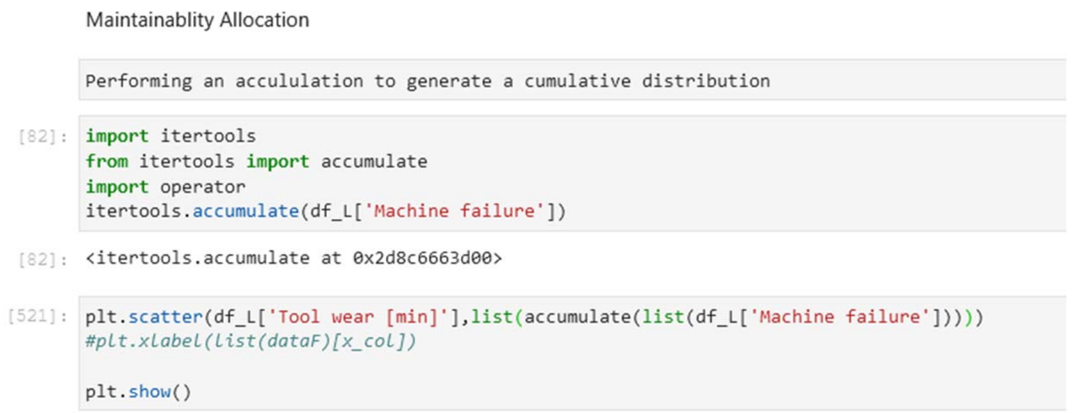
```
df_L.head()
```

	Type	Air temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
UDI											
675	1	297.8	1456	41.2	0	0	0	0	0	0	0
8692	1	297.1	1555	36.2	0	0	0	0	0	0	0
3946	1	302.2	1570	36.1	0	0	0	0	0	0	0
3867	1	302.6	1311	53.2	0	0	0	0	0	0	0
8773	1	297.5	1621	28.9	0	0	0	0	0	0	0

```
def feature_dist(data_F):
    """ shows the distribution of each feature"""
    for count,col in enumerate(data_F):
        data_F.hist(column = list(data_F)[count], bins=int(10000**.5))
```

Cumulative Distribution Function for machine L

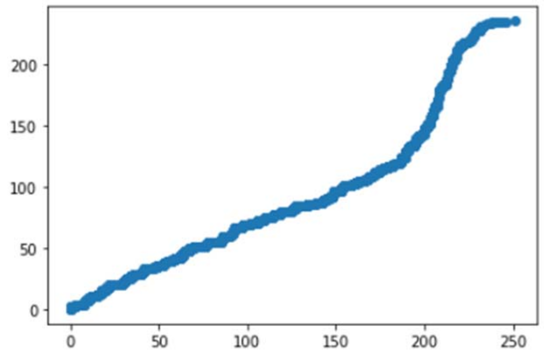
Time vs Machine Failures (ALL failure modes)



Maintainablity Allocation

```
Performing an acculation to generate a cumulative distribution
```

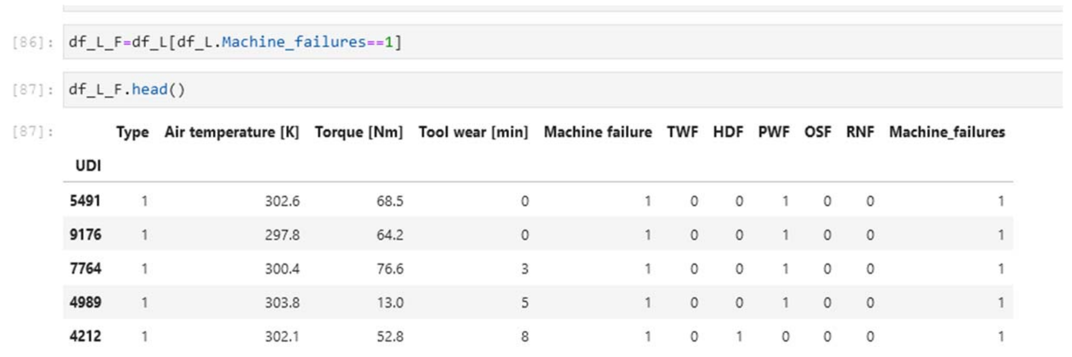
```
[82]: import itertools
      from itertools import accumulate
      import operator
      itertools.accumulate(df_L['Machine failure'])

[82]: <itertools.accumulate at 0x2d8c6663d00>

[521]: plt.scatter(df_L['Tool wear [min]'],list(accumulate(list(df_L['Machine failure']))))
       #plt.xlabel(list(dataF)[x_col])

       plt.show()
```



A Data Frame with strictly failures times.

Creation of df_L_F where we can focus on failures only.

This allows for a Time to Failure calculation later.



```
[86]: df_L_F=df_L[df_L.Machine_failures==1]
```

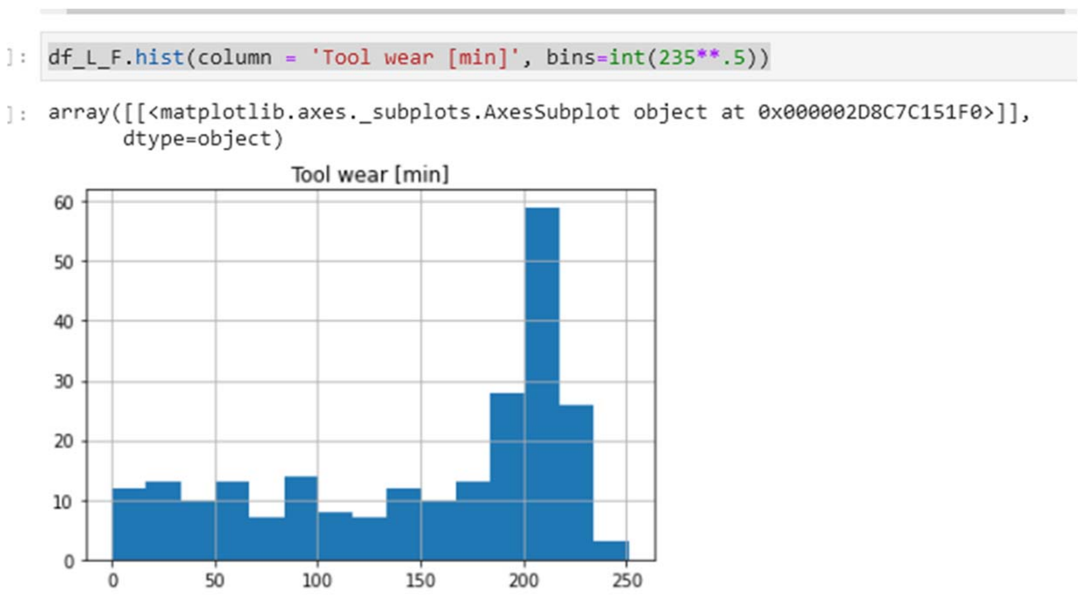
```
[87]: df_L_F.head()
```

[87]:

UDI	Type	Air temperature [K]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	Machine_failures
5491	1	302.6	68.5	0	1	0	0	1	0	0	1
9176	1	297.8	64.2	0	1	0	0	1	0	0	1
7764	1	300.4	76.6	3	1	0	0	1	0	0	1
4989	1	303.8	13.0	5	1	0	0	1	0	0	1
4212	1	302.1	52.8	8	1	0	1	0	0	0	1

Tool wear distribution

Tool wear in relation to machine failures and the Probability Mass Function shown with a Histogram



```
]: df_L_F.hist(column = 'Tool wear [min]', bins=int(235**.5))
```

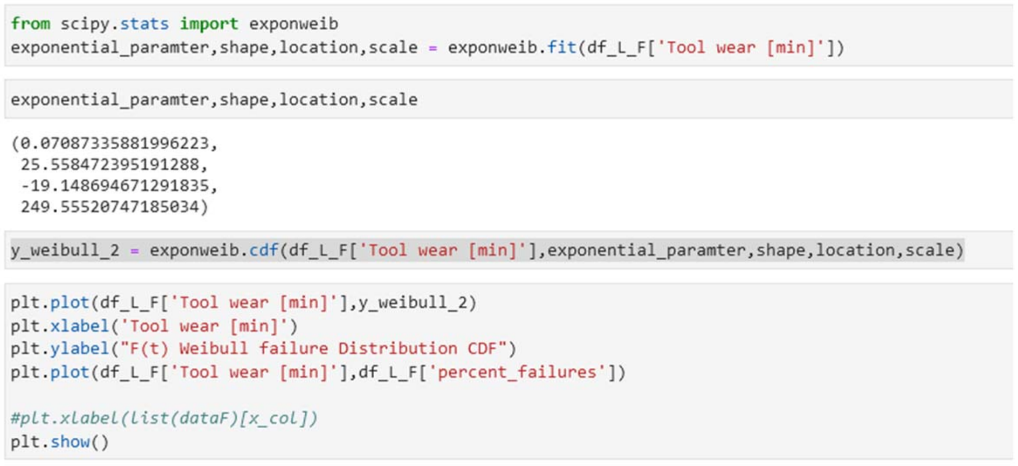
```
]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002D8C7C151F0>]],
         dtype=object)
```

Fitting a Weibull Distribution

Used SciPy to estimate Weibull parameters,

Then used those parameters to create a graph.

As shown the Blue is Weibull and the Orange is Tool Wear vs the CDF (percent failures from the Data)



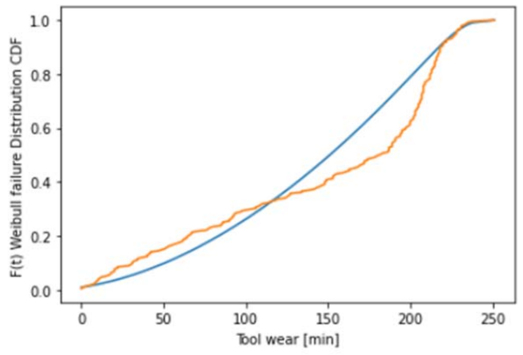
```
from scipy.stats import exponweib
exponential_paramter,shape,location,scale = exponweib.fit(df_L_F['Tool wear [min]'])
```

```
exponential_paramter,shape,location,scale
```

```
(0.07087335881996223,
 25.558472395191288,
 -19.148694671291835,
 249.55520747185034)
```

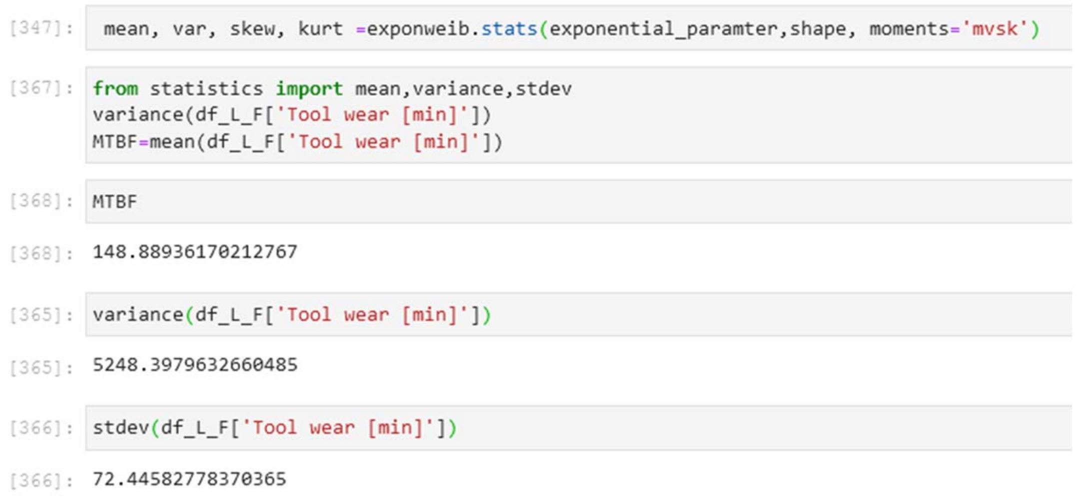
```
y_weibull_2 = exponweib.cdf(df_L_F['Tool wear [min]'],exponential_paramter,shape,location,scale)
```

```
plt.plot(df_L_F['Tool wear [min]'],y_weibull_2)
plt.xlabel('Tool wear [min]')
plt.ylabel("F(t) Weibull failure Distribution CDF")
plt.plot(df_L_F['Tool wear [min]'],df_L_F['percent_failures'])

#plt.xlabel(list(dataF)[x_col])
plt.show()
```



Reliability Stats

Using Scipy to get the statistics. Here MTBF is mean time between failures



```
[347]:  mean, var, skew, kurt =exponweib.stats(exponential_paramter,shape, moments='mvsk')

[367]: from statistics import mean,variance,stdev
       variance(df_L_F['Tool wear [min]'])
       MTBF=mean(df_L_F['Tool wear [min]'])

[368]: MTBF

[368]: 148.88936170212767

[365]: variance(df_L_F['Tool wear [min]'])

[365]: 5248.3979632660485

[366]: stdev(df_L_F['Tool wear [min]'])

[366]: 72.44582778370365
```

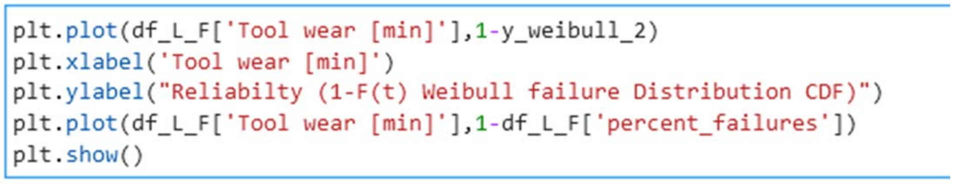
Reliability (survival function)

Torque has a relationship with the PWF failure mode. this will require further investigation in the future.

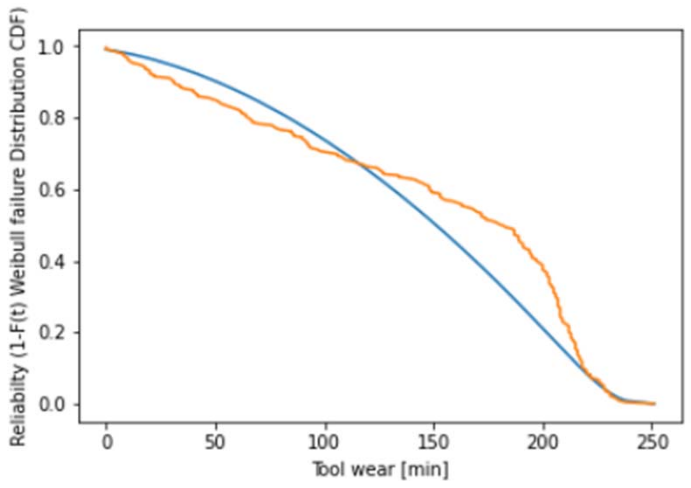
High torque has an effect on HDF failure and OSF

Tool wear has an effect on OSF and TWF

High air temp can cause an HDF failure



```
plt.plot(df_L_F['Tool wear [min]'],1-y_weibull_2)
plt.xlabel('Tool wear [min]')
plt.ylabel("Reliabilty (1-F(t) Weibull failure Distribution CDF)")
plt.plot(df_L_F['Tool wear [min]'],1-df_L_F['percent_failures'])
plt.show()
```



Final Conclusions

The reliability seems to be 0 near the 255-minute mark.
The biggest indicator of failure is Time.
To improve performance design needs to be adjusted to lengthen time

Survivability can be improved by

1) changing design parameters such that the temperature does not go as high. A way to do this could be improved cooling.
2) Limitations on Torque could be improved as to reduce the chance of Overstrain.
3) Perform more experiments and allow them to run until they fail. This removes censored data and possibly shows the reliability to be greater than the current calculations.

These changes could allow for longer tool wear out periods and better warranty for the machines

Works Cited

1) Klosterman, Stephan. Data Science Projects with Python, Packt Publishing, 2019, Online at O'Reilly Media.
2) Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
3) Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
4) Dovivh, Robert and Wartman, Bill Certified Reliability Engineer Primer, Indiana Council of quality, 2014
5) Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

Works Cited Continued

6) US government, NIST Statistical handbook https://www.itl.nist.gov/div898/handbook/apr/section1/apr162.htm , https://www.itl.nist.gov/div898/handbook/apr/section4/apr47.htm

7) Zaiontz, C. (2020) Real Statistics Using Excel. www.real-statistics.com , 11/27/2021