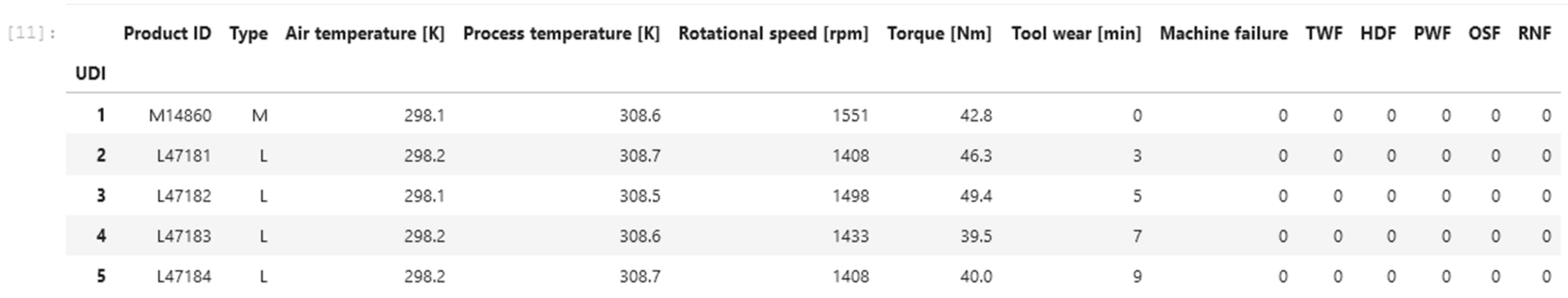
Preventative Maintenance
By
D. Kelley

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

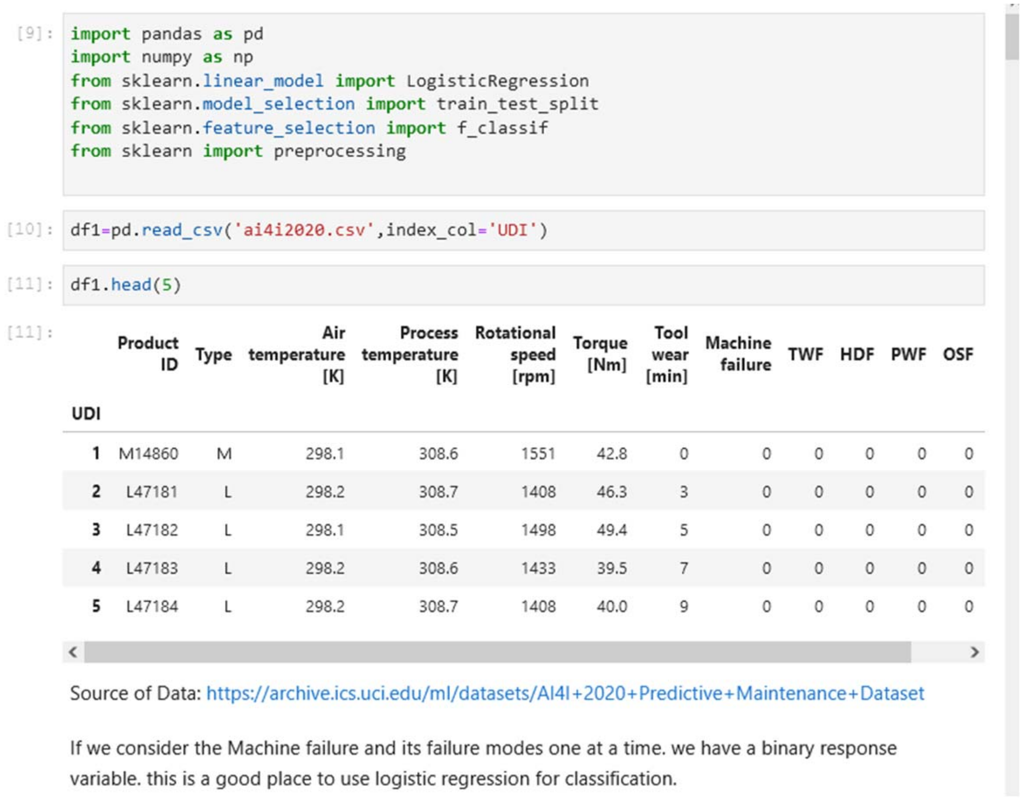


[11]:

	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

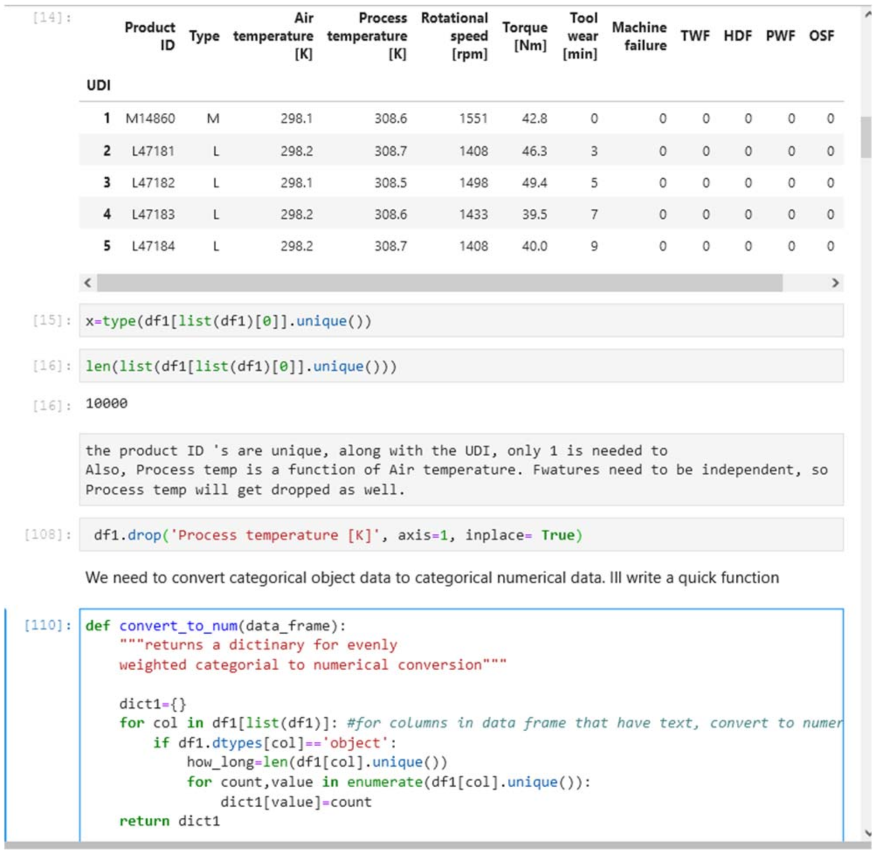
Source of Data: https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset

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

```
[16]: 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.
```

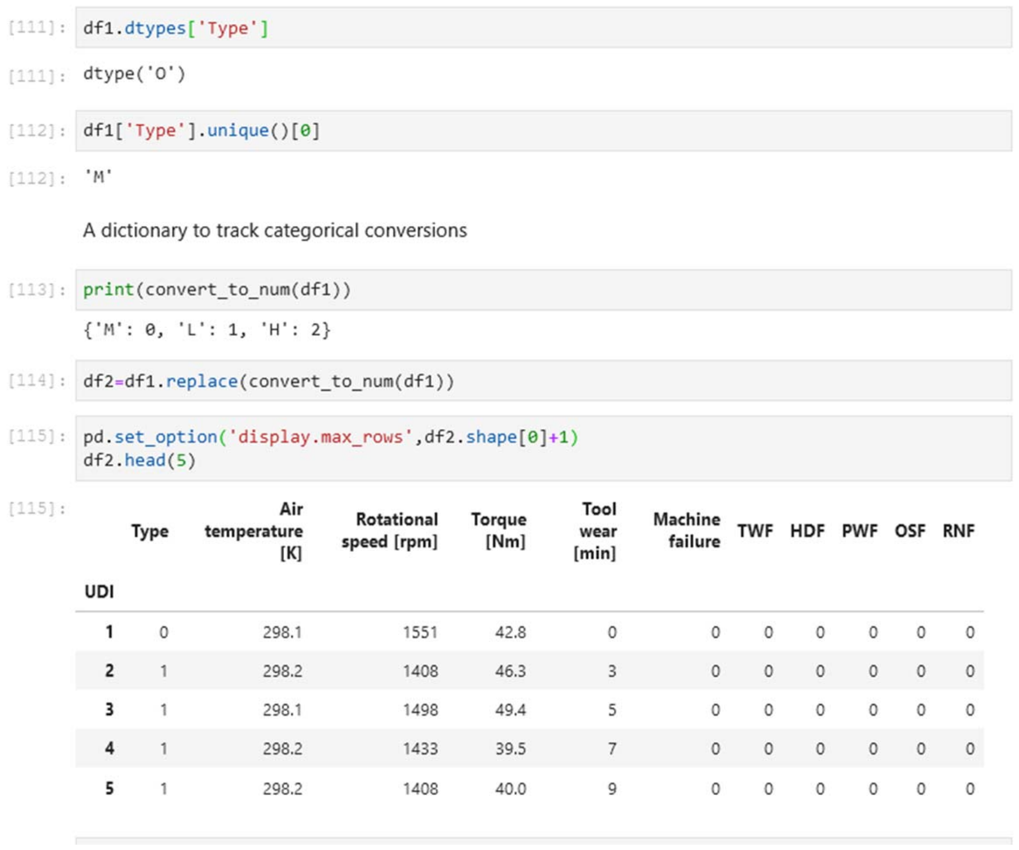
```
[108]: df1.drop('Process temperature [K]', axis=1, inplace= True)
```

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

```
[110]: 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'
```

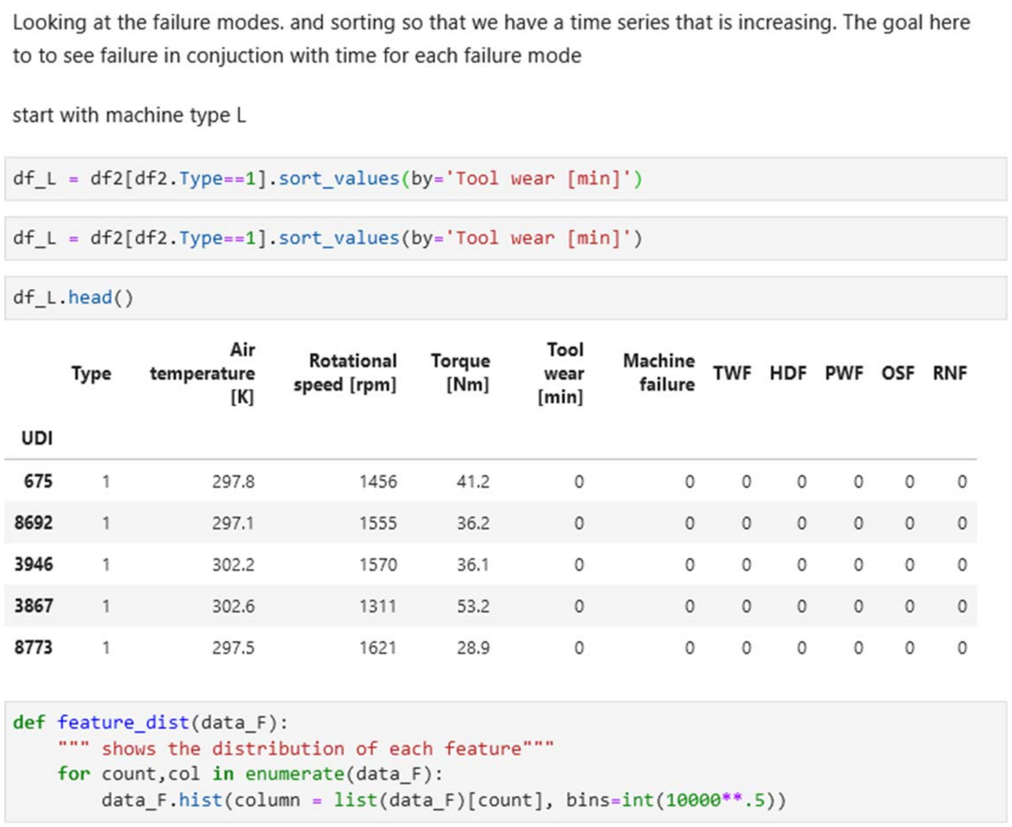
A dictionary to track categorical conversions

```
[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]')
```

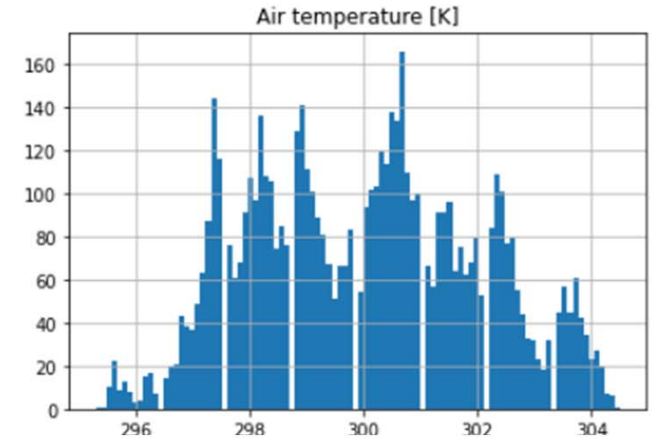
```
df_L.head()
```

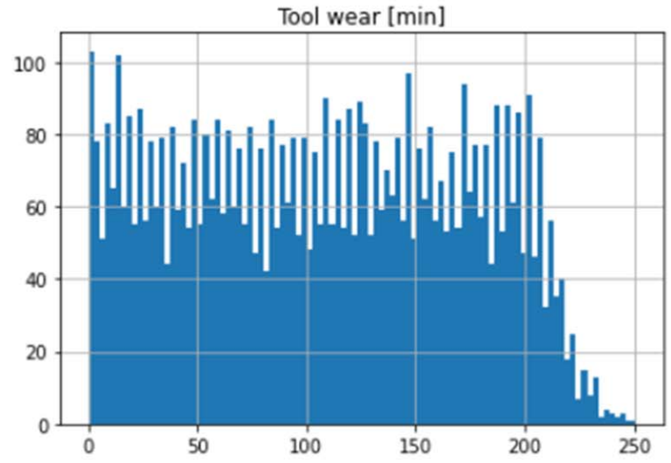
UDI	Type	Air temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
675	1	297.8	1456	41.2	0	0	0	0	0	0	0
8692	1	297.1	1555	36.2	0	0	0	0	0	0	0
3946	1	302.2	1570	36.1	0	0	0	0	0	0	0
3867	1	302.6	1311	53.2	0	0	0	0	0	0	0
8773	1	297.5	1621	28.9	0	0	0	0	0	0	0

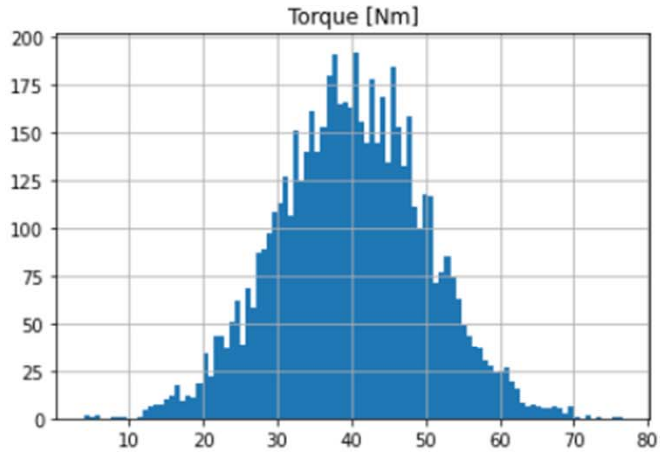
```
def feature_dist(data_F):
    """ shows the distribution of each feature"""
    for count,col in enumerate(data_F):
        data_F.hist(column = list(data_F)[count], bins=int(10000**.5))
```

```
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))
```

Except for tool wear, which appears to havde a uniform distrubution, the features seem to have a quasi-normal distribution.

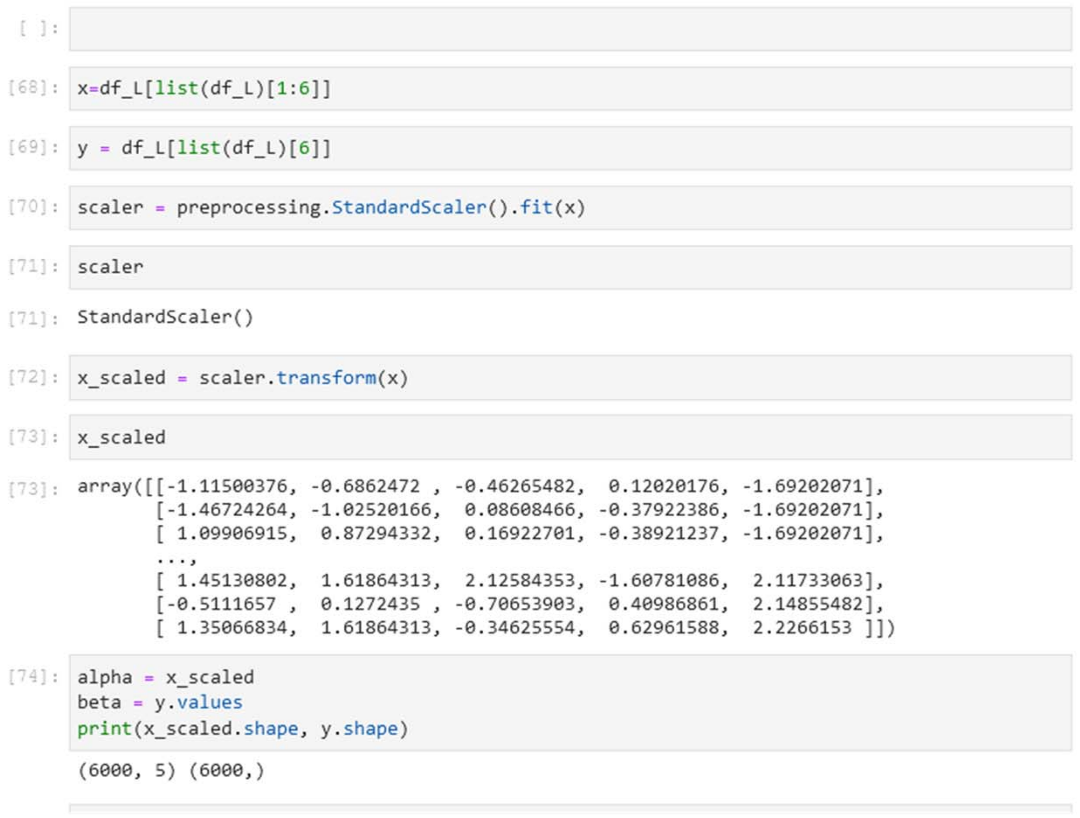






Normal Distribution

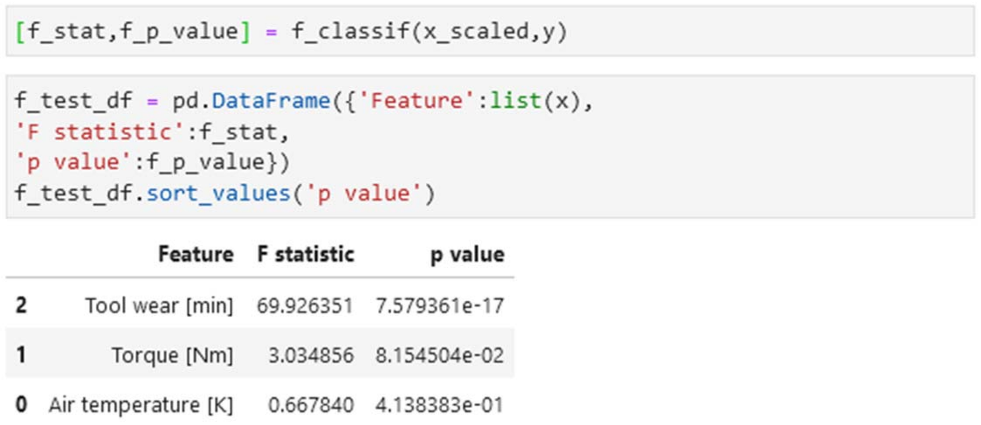
Preprocessing the data to ensure it has a normal distribution. Then running an ANOVA F Test.



```
[ ]:

[68]: x=df_L[list(df_L)[1:6]]

[69]: y = df_L[list(df_L)[6]]

[70]: scaler = preprocessing.StandardScaler().fit(x)

[71]: scaler

[71]: StandardScaler()

[72]: x_scaled = scaler.transform(x)

[73]: x_scaled

[73]: array([[-1.11500376, -0.6862472 , -0.46265482,  0.12020176, -1.69202071],
             [-1.46724264, -1.02520166,  0.08608466, -0.37922386, -1.69202071],
             [ 1.09906915,  0.87294332,  0.16922701, -0.38921237, -1.69202071],
             ...,
             [ 1.45130802,  1.61864313,  2.12584353, -1.60781086,  2.11733063],
             [-0.5111657 ,  0.1272435 , -0.70653903,  0.40986861,  2.14855482],
             [ 1.35066834,  1.61864313, -0.34625554,  0.62961588,  2.2266153 ]])

[74]: alpha = x_scaled
      beta = y.values
      print(x_scaled.shape, y.shape)

      (6000, 5) (6000,)
```

F Statistic

Results of the F test showing that Torque has the greatest effect on failures, regardless of Mode



```
[f_stat,f_p_value] = f_classif(x_scaled,y)
```

```
f_test_df = pd.DataFrame({'Feature':list(x),
'F statistic':f_stat,
'p value':f_p_value})
f_test_df.sort_values('p value')
```

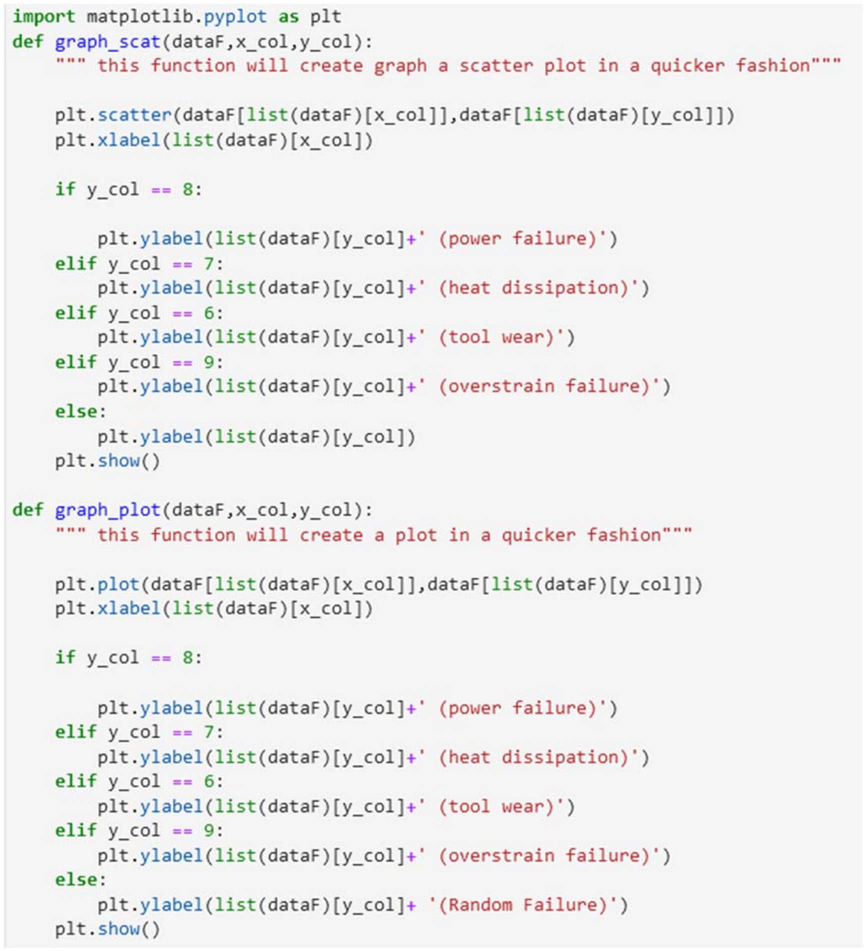
	Feature	F statistic	p value
2	Tool wear [min]	69.926351	7.579361e-17
1	Torque [Nm]	3.034856	8.154504e-02
0	Air temperature [K]	0.667840	4.138383e-01

Graph Customizing

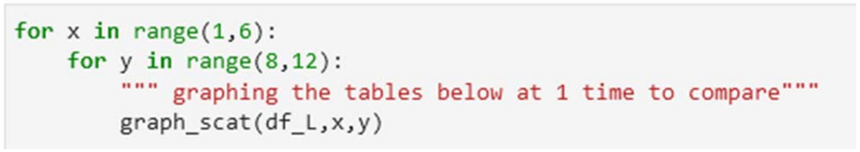
Here I am comparing features with failure modes one at a time. The functions take a dataframe, x column index and y column index.

Below the graphs that show a trend are shown.

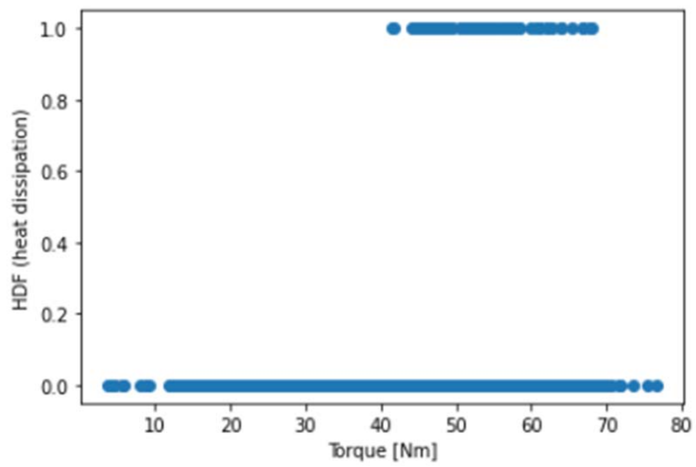
Recall that the Failure modes are binary (0 and 1)

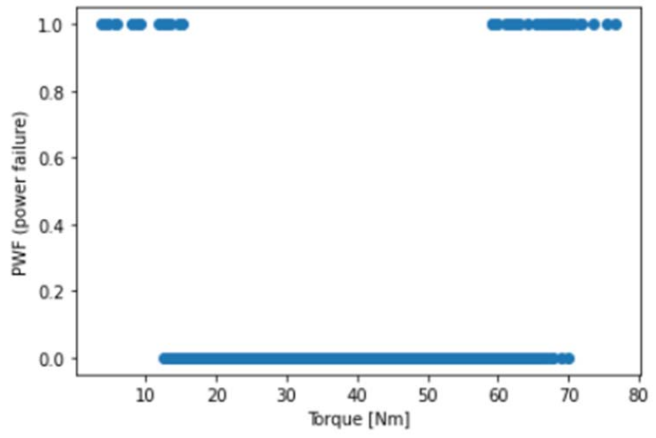


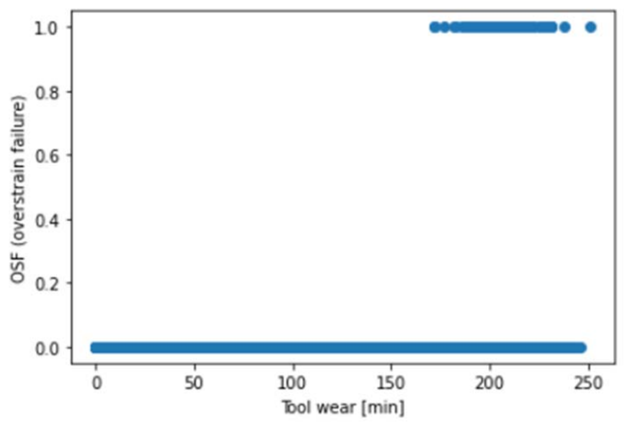
```python
import matplotlib.pyplot as plt
def graph_scat(dataF,x_col,y_col):
    """ this function will create graph a scatter plot in a quicker fashion"""

    plt.scatter(dataF[list(dataF)[x_col]],dataF[list(dataF)[y_col]])
    plt.xlabel(list(dataF)[x_col])

    if y_col == 8:

        plt.ylabel(list(dataF)[y_col]+' (power failure)')
    elif y_col == 7:
        plt.ylabel(list(dataF)[y_col]+' (heat dissipation)')
    elif y_col == 6:
        plt.ylabel(list(dataF)[y_col]+' (tool wear)')
    elif y_col == 9:
        plt.ylabel(list(dataF)[y_col]+' (overstrain failure)')
    else:
        plt.ylabel(list(dataF)[y_col])
    plt.show()

def graph_plot(dataF,x_col,y_col):
    """ this function will create a plot in a quicker fashion"""

    plt.plot(dataF[list(dataF)[x_col]],dataF[list(dataF)[y_col]])
    plt.xlabel(list(dataF)[x_col])

    if y_col == 8:

        plt.ylabel(list(dataF)[y_col]+' (power failure)')
    elif y_col == 7:
        plt.ylabel(list(dataF)[y_col]+' (heat dissipation)')
    elif y_col == 6:
        plt.ylabel(list(dataF)[y_col]+' (tool wear)')
    elif y_col == 9:
        plt.ylabel(list(dataF)[y_col]+' (overstrain failure)')
    else:
        plt.ylabel(list(dataF)[y_col]+ '(Random Failure)')
    plt.show()
```

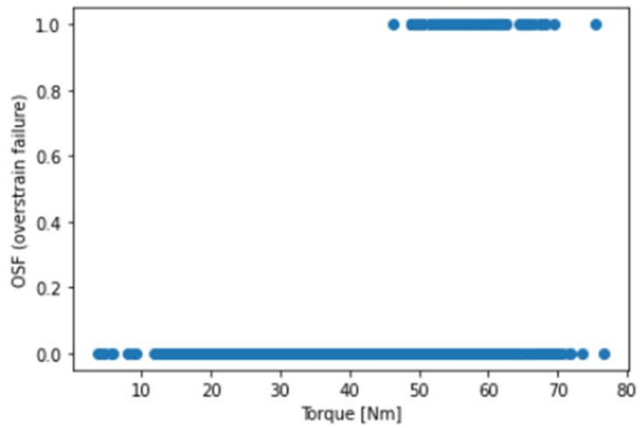


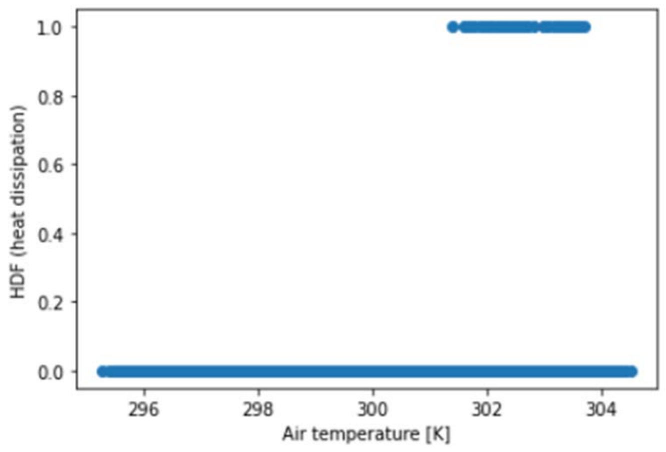
```python
for x in range(1,6):
    for y in range(8,12):
        """ graphing the tables below at 1 time to compare"""
        graph_scat(df_L,x,y)
```

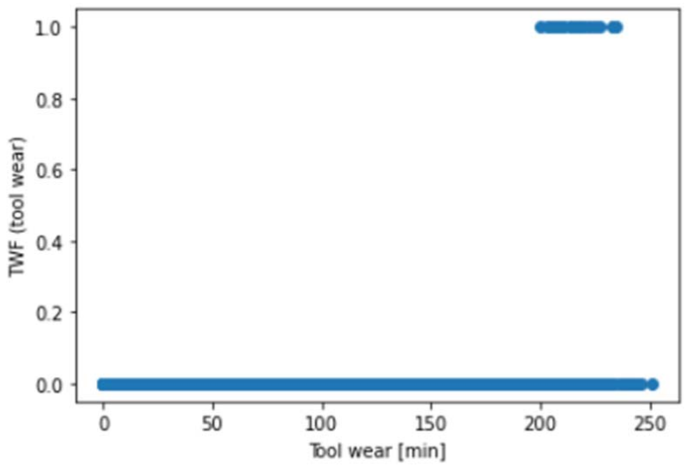

HDF (heat dissipation)
Torque [Nm]
1.0
0.8
0.6
0.4
0.2
0.0
10
20
30
40
50
60
70
80


PWF (power failure)
Torque [Nm]
1.0
0.8
0.6
0.4
0.2
0.0
10
20
30
40
50
60
70
80


OSF (overstrain failure)
Tool wear [min]
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250


OSF (overstrain failure)
Torque [Nm]
1.0
0.8
0.6
0.4
0.2
0.0
10
20
30
40
50
60
70
80


HDF (heat dissipation)
1.0
0.8
0.6
0.4
0.2
0.0
296
298
300
302
304
Air temperature [K]


TWF (tool wear)
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
Tool wear [min]

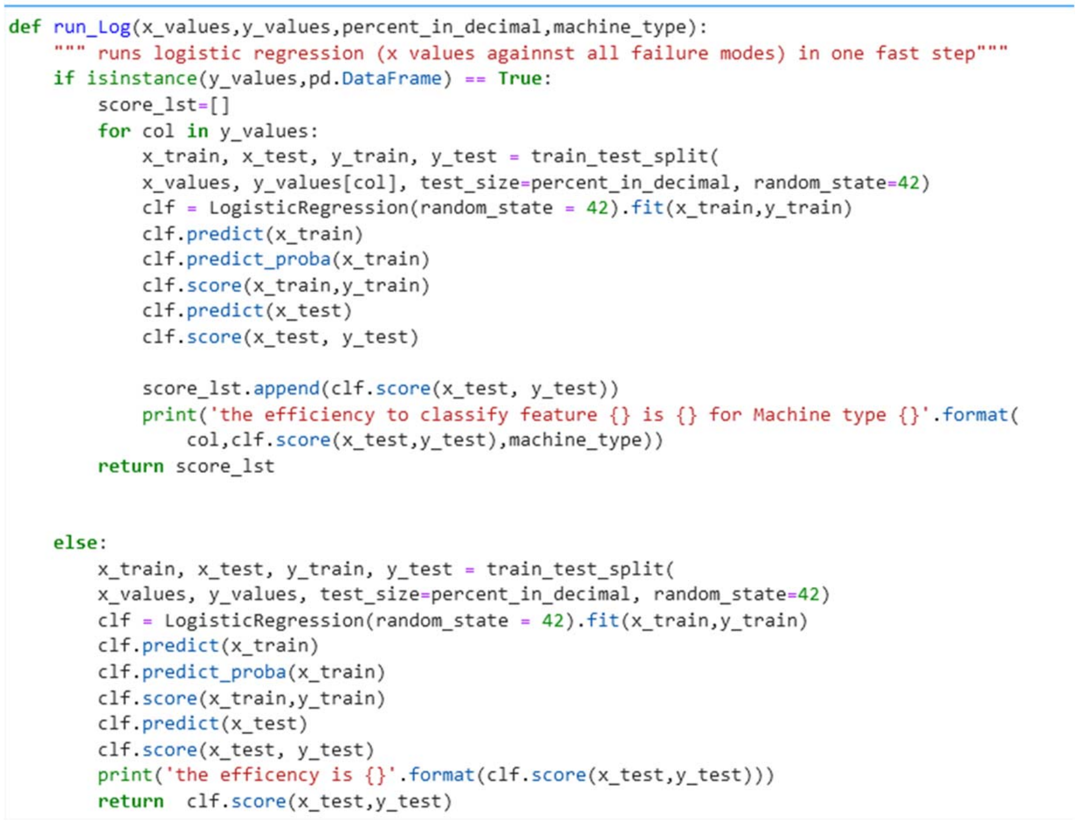
Graphical Conclusion:

More errors occur under the current conditions:

- High and Low Rotational speeds

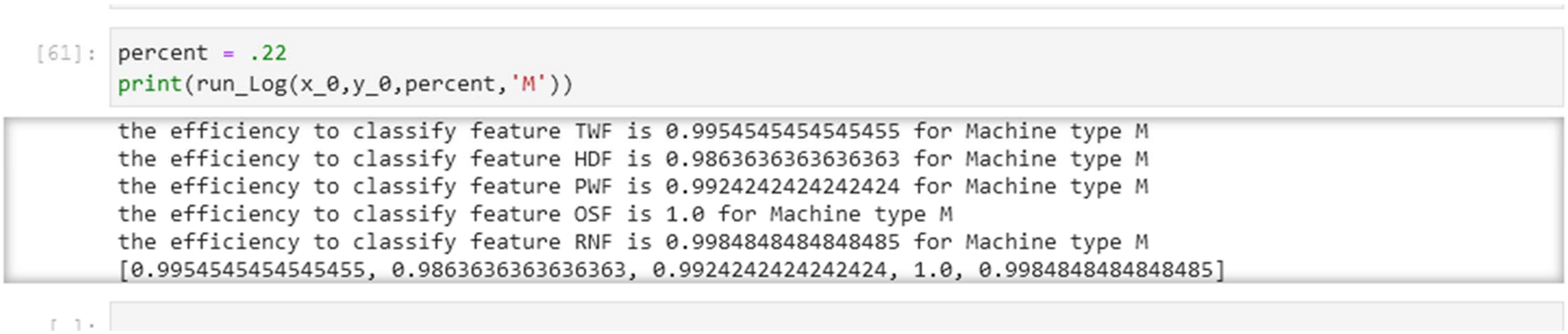
Simplify classification

Using a custom function to generate the classifiers.



```python
def run_Log(x_values,y_values,percent_in_decimal,machine_type):
    """ runs logistic regression (x values againnst all failure modes) in one fast step"""
    if isinstance(y_values,pd.DataFrame) == True:
        score_lst=[]
        for col in y_values:
            x_train, x_test, y_train, y_test = train_test_split(
            x_values, y_values[col], test_size=percent_in_decimal, random_state=42)
            clf = LogisticRegression(random_state = 42).fit(x_train,y_train)
            clf.predict(x_train)
            clf.predict_proba(x_train)
            clf.score(x_train,y_train)
            clf.predict(x_test)
            clf.score(x_test, y_test)

            score_lst.append(clf.score(x_test, y_test))
            print('the efficiency to classify feature {} is {} for Machine type {}'.format(
                col,clf.score(x_test,y_test),machine_type))
        return score_lst


    else:
        x_train, x_test, y_train, y_test = train_test_split(
        x_values, y_values, test_size=percent_in_decimal, random_state=42)
        clf = LogisticRegression(random_state = 42).fit(x_train,y_train)
        clf.predict(x_train)
        clf.predict_proba(x_train)
        clf.score(x_train,y_train)
        clf.predict(x_test)
        clf.score(x_test, y_test)
        print('the efficency is {}'.format(clf.score(x_test,y_test)))
        return  clf.score(x_test,y_test)
```

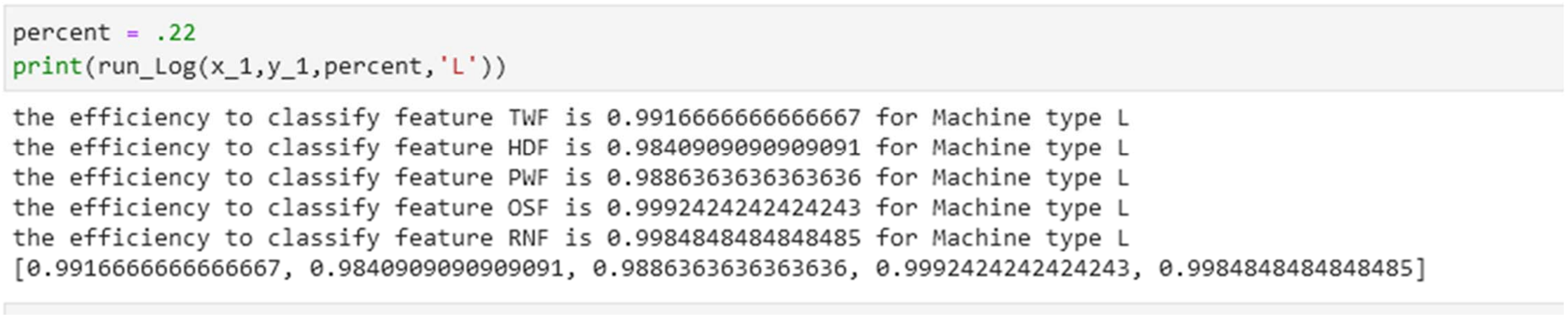
Results of the classifier



```
[61]: percent = .22
      print(run_Log(x_0,y_0,percent,'M'))
```

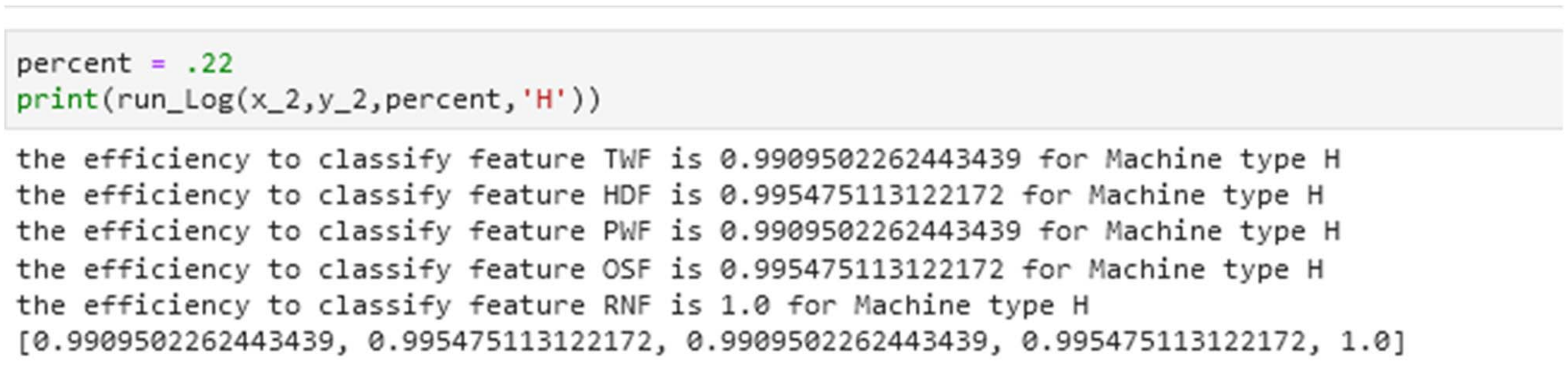
```
the efficiency to classify feature TWF is 0.9954545454545455 for Machine type M
the efficiency to classify feature HDF is 0.9863636363636363 for Machine type M
the efficiency to classify feature PWF is 0.9924242424242424 for Machine type M
the efficiency to classify feature OSF is 1.0 for Machine type M
the efficiency to classify feature RNF is 0.9984848484848485 for Machine type M
[0.9954545454545455, 0.9863636363636363, 0.9924242424242424, 1.0, 0.9984848484848485]
```

Results continued



```python
percent = .22
print(run_Log(x_1,y_1,percent,'L'))
```

```
the efficiency to classify feature TWF is 0.9916666666666667 for Machine type L
the efficiency to classify feature HDF is 0.9840909090909091 for Machine type L
the efficiency to classify feature PWF is 0.9886363636363636 for Machine type L
the efficiency to classify feature OSF is 0.9992424242424243 for Machine type L
the efficiency to classify feature RNF is 0.9984848484848485 for Machine type L
[0.9916666666666667, 0.9840909090909091, 0.9886363636363636, 0.9992424242424243, 0.9984848484848485]
```



```python
percent = .22
print(run_Log(x_2,y_2,percent,'H'))
```

```
the efficiency to classify feature TWF is 0.9909502262443439 for Machine type H
the efficiency to classify feature HDF is 0.995475113122172 for Machine type H
the efficiency to classify feature PWF is 0.9909502262443439 for Machine type H
the efficiency to classify feature OSF is 0.995475113122172 for Machine type H
the efficiency to classify feature RNF is 1.0 for Machine type H
[0.9909502262443439, 0.995475113122172, 0.9909502262443439, 0.995475113122172, 1.0]
```

Final Conclusions

The classifiers all work within 98-99 percent.

Those classifiers have the same results, regardless of the Type of the machine.

Quality can be improved by

1) changing design parameter 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.

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