Business Understanding

Downtime for heavy machinery costs a lot of money in the manufacturing industry, both in terms of idle time wasted due to maintenance work and in terms of repair costs. It would be a significant boost to the bottom line if firms could be proactive and undertake routine maintenance activities proactively, as well as predict concerns ahead of time using previous data. Instead, enterprises typically use IOT (Internet of Things) sensors to monitor and collect data from a variety of telemetric sensors. A predictive model can be constructed by combining telemetry data and failure reports to anticipate future heavy machinery fault occurrences.

Business Goal

The end goal is to create a proactive maintenance strategy that tries to predict future failures of various components in heavy machines. As mentioned earlier, it benefits the businesses by reducing operational costs, long term maintenance costs and maximizing production hours.

Importing Required python modules

In [15]:

```python
# Importing required modules

import pandas as pd
import numpy as np

from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# for measuring accuracy, precision, recall, f1 and auc scores
from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix


from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

# for model deployment
import joblib

import matplotlib.pyplot as plt
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')

from datetime import datetime as dt
```

Data Understanding

The following data sources were considered for building this Predictive Maintenance Model.

- **Telemetry**: Time series data consisting of various measurements like - Voltage, Rotation, Pressure and Vibration readings from various machines.
- **Machines**: Information about machines.
- **Failures**: Records of failed components.
- **Maintenance**: Maintenance historical records of machines involving component replacements due to regular maintenance activity or due to failures.
- **Errors**: Historical errors thrown by the machines.

Importing Data Sources

In [17]:

```python
# creating data file directory
import os
##cwd = os.getcwd()
os.chdir('C:/Users/14802/OneDrive/Desktop/DSC 680-PROJECTS/Projects/Week1/Cod

cwd = os.getcwd()
print(cwd)
```

```
C:\Users\14802\OneDrive\Desktop\DSC 680-PROJECTS\Projects\Week1\Code
```

In [18]:

```python
# creating data file directory
import os
cwd = os.getcwd()
print(cwd)

projdir = os.path.dirname(cwd)
datadir = os.path.join(projdir, 'Data')


# r=root, d=directories, f = files

print("\nThe directory contains below files : \n")
for r, d, f in os.walk(datadir):
    for file in f:
         print(file)
```

```
C:\Users\14802\OneDrive\Desktop\DSC 680-PROJECTS\Projects\Week1\Code

The directory contains below files :

errors.csv
failures.csv
machines.csv
maint.csv
telemetry.csv
```

In [19]:
```
# importing telemetry data

telemetryfile = os.path.join(datadir, 'telemetry.csv')

telemetry_df = pd.read_csv(telemetryfile)
telemetry_df.head()
```

Out[19]:

	datetime	machineID	volt	rotate	pressure	vibration
0	1/1/2015 6:00:00 AM	1	176.217853	418.504078	113.077935	45.087686
1	1/1/2015 7:00:00 AM	1	162.879223	402.747490	95.460525	43.413973
2	1/1/2015 8:00:00 AM	1	170.989902	527.349825	75.237905	34.178847
3	1/1/2015 9:00:00 AM	1	162.462833	346.149335	109.248561	41.122144
4	1/1/2015 10:00:00 AM	1	157.610021	435.376873	111.886648	25.990511

In [20]:
```
# importing machines data

machinesfile = os.path.join(datadir, 'machines.csv')

machines_df = pd.read_csv(machinesfile)
machines_df.head()
```

Out[20]:

	machineID	model	age
0	1	model3	18
1	2	model4	7
2	3	model3	8
3	4	model3	7
4	5	model3	2

In [21]:
```
# importing errors data

errorsfile = os.path.join(datadir, 'errors.csv')

errors_df = pd.read_csv(errorsfile)
errors_df.head()
```

Out[21]:

	datetime	machineID	errorID
0	1/3/2015 7:00:00 AM	1	error1
1	1/3/2015 8:00:00 PM	1	error3
2	1/4/2015 6:00:00 AM	1	error5
3	1/10/2015 3:00:00 PM	1	error4
4	1/22/2015 10:00:00 AM	1	error4

In [22]:

```python
# importing failures data

failuresfile = os.path.join(datadir, 'failures.csv')

failures_df = pd.read_csv(failuresfile)
failures_df.head()
```

Out[22]:

	datetime	machineID	failure
0	1/5/2015 6:00:00 AM	1	comp4
1	3/6/2015 6:00:00 AM	1	comp1
2	4/20/2015 6:00:00 AM	1	comp2
3	6/19/2015 6:00:00 AM	1	comp4
4	9/2/2015 6:00:00 AM	1	comp4

In [23]:

```python
failures_df.shape
```

Out[23]: (761, 3)

In [24]:

```python
# importing maintenance data

maintfile = os.path.join(datadir, 'maint.csv')

maint_df = pd.read_csv(maintfile)
maint_df.head()
```

Out[24]:

	datetime	machineID	comp
0	6/1/2014 6:00:00 AM	1	comp2
1	7/16/2014 6:00:00 AM	1	comp4
2	7/31/2014 6:00:00 AM	1	comp3
3	12/13/2014 6:00:00 AM	1	comp1
4	1/5/2015 6:00:00 AM	1	comp4

In [25]:
```python
# Defining fonts for plotting exploratory data analysis

titlefont = {'family': 'serif',
        'color':  'lightblue',
        'weight': 'bold',
        'size': 16,
        }

labelfont = {'family': 'serif',
        'color':  'black',
        'weight': 'normal',
        'size': 12,
        }
```

Exploratory Data Analysis

In [26]:
```python
telemetry_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 876100 entries, 0 to 876099
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   datetime   876100 non-null  object 
 1   machineID  876100 non-null  int64  
 2   volt       876100 non-null  float64
 3   rotate     876100 non-null  float64
 4   pressure   876100 non-null  float64
 5   vibration  876100 non-null  float64
dtypes: float64(4), int64(1), object(1)
memory usage: 40.1+ MB
```

In [27]:
```python
telemetry_df.describe()
```

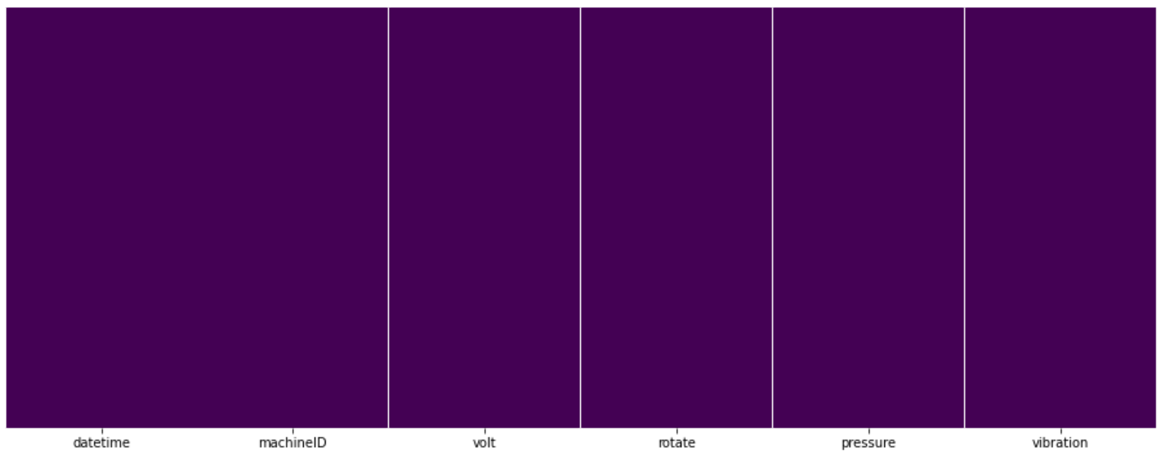
Out[27]:

	machineID	volt	rotate	pressure	vibration
count	876100.000000	876100.000000	876100.000000	876100.000000	876100.000000
mean	50.500000	170.777736	446.605119	100.858668	40.385007
std	28.866087	15.509114	52.673886	11.048679	5.370361
min	1.000000	97.333604	138.432075	51.237106	14.877054
25%	25.750000	160.304927	412.305714	93.498181	36.777299
50%	50.500000	170.607338	447.558150	100.425559	40.237247
75%	75.250000	181.004493	482.176600	107.555231	43.784938
max	100.000000	255.124717	695.020984	185.951998	76.791072

Checking for missing values

In [28]:
```
# checking heat map for missing values
plt.figure(figsize=(16, 6))
sns.heatmap(telemetry_df.isnull(),yticklabels=False,cbar=False,cmap='viridis'
```

Out[28]: <AxesSubplot:>



In [29]:
```
# Checking for blank values for each column of dataframe
telemetry_nullcols = telemetry_df.isnull().sum()
print(telemetry_nullcols)
```

```
datetime     0
machineID    0
volt         0
rotate       0
pressure     0
vibration    0
dtype: int64
```

In [30]:
```
# number of telemetry records
print(len(telemetry_df))
print(telemetry_df.shape)
```

```
876100
(876100, 6)
```

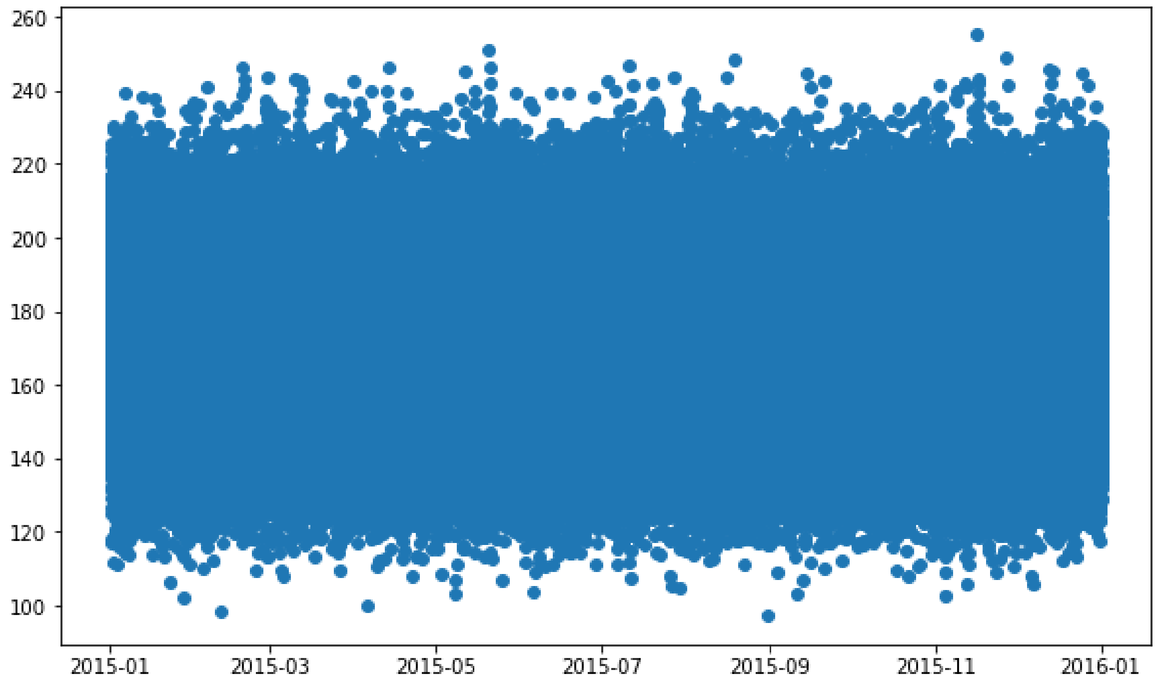
from datetime import datetime as dt dt.strftime(to_datetime['datetime'])

In [31]:
```
telemetry_df['DT'] = pd.to_datetime(telemetry_df['datetime'])
```

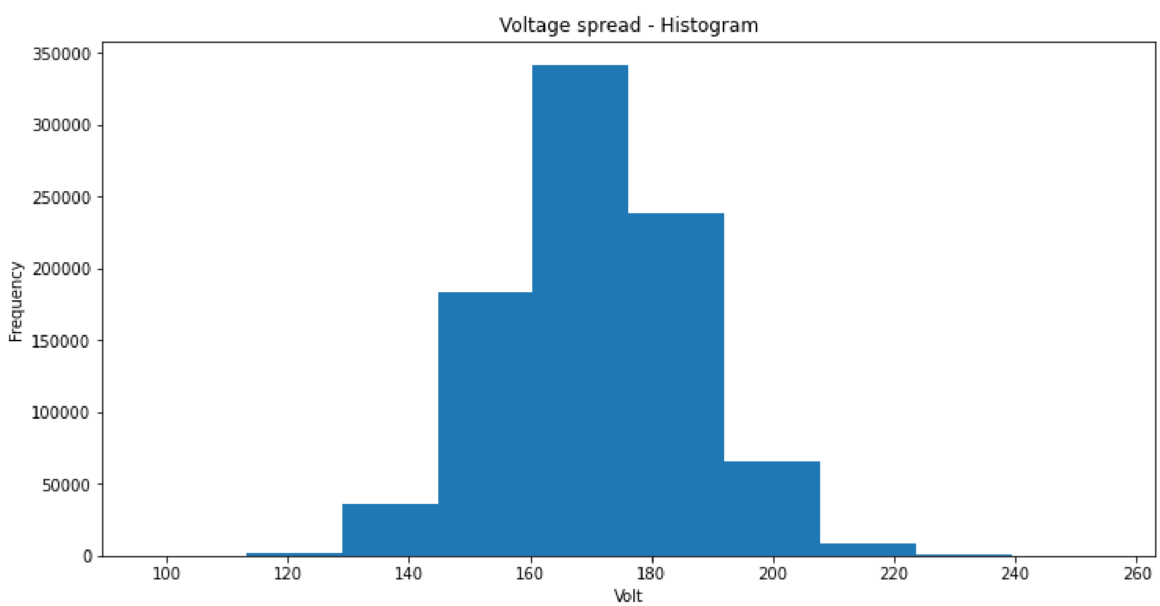
In [32]:
```
# date range of telemetric data
np.min(telemetry_df['DT']), np.max(telemetry_df['DT'])
```

Out[32]: (Timestamp('2015-01-01 06:00:00'), Timestamp('2016-01-01 06:00:00'))

In [33]:
```
plt.figure(figsize=(10, 6))
plt.plot_date(x=telemetry_df['DT'], y=telemetry_df['volt'])
plt.show()
```



In [34]:
```
plt.figure(figsize=(12, 6))
plt.hist(telemetry_df['volt'])
plt.xlabel('Volt')
plt.ylabel('Frequency')
plt.title('Voltage spread - Histogram')
plt.show()
```



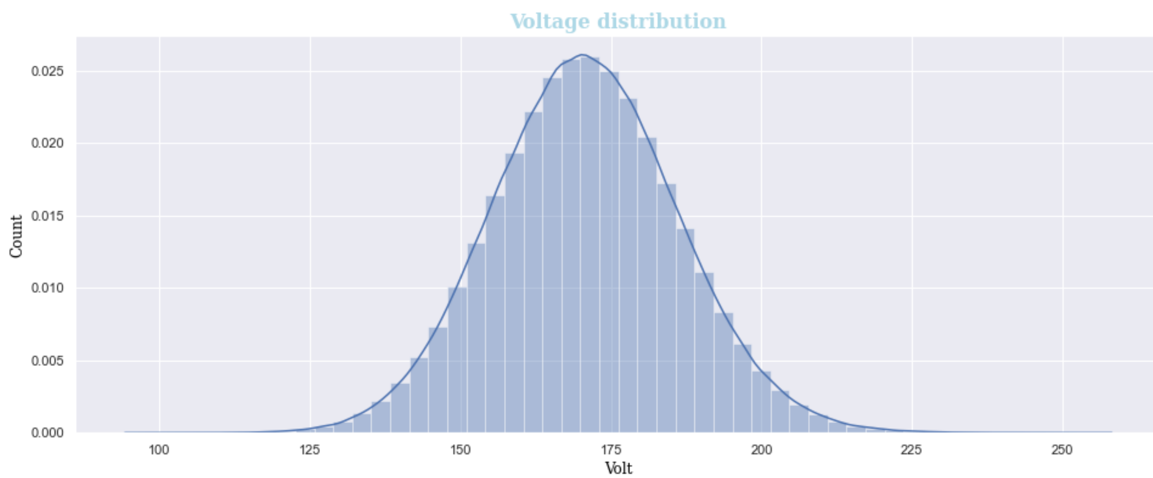
In [35]:

```
sns.set(style="darkgrid")
plt.figure(figsize=(16, 6))
sns.distplot(telemetry_df.volt, kde=True,color="b")

plt.xlabel("Volt", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Voltage distribution", fontdict=titlefont)
```

```
C:\Users\14802\anaconda3\lib\site-packages\seaborn\distributions.py:2551: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[35]:

```
Text(0.5, 1.0, 'Voltage distribution')
```

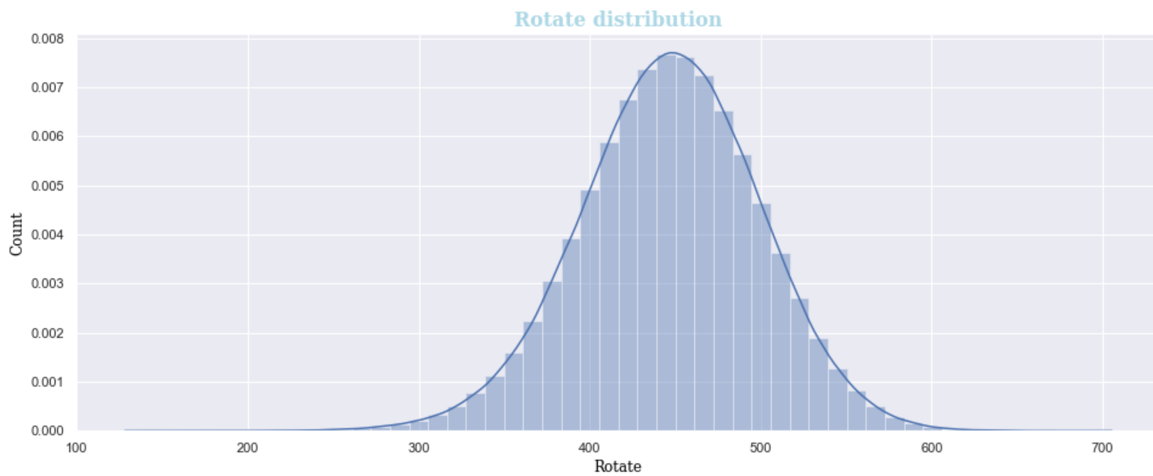


In [36]:

```
sns.set(style="darkgrid")
plt.figure(figsize=(16, 6))
sns.distplot(telemetry_df.rotate, kde=True,color="b")

plt.xlabel("Rotate", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Rotate distribution", fontdict=titlefont)
```

```
C:\Users\14802\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[36]: Text(0.5, 1.0, 'Rotate distribution')

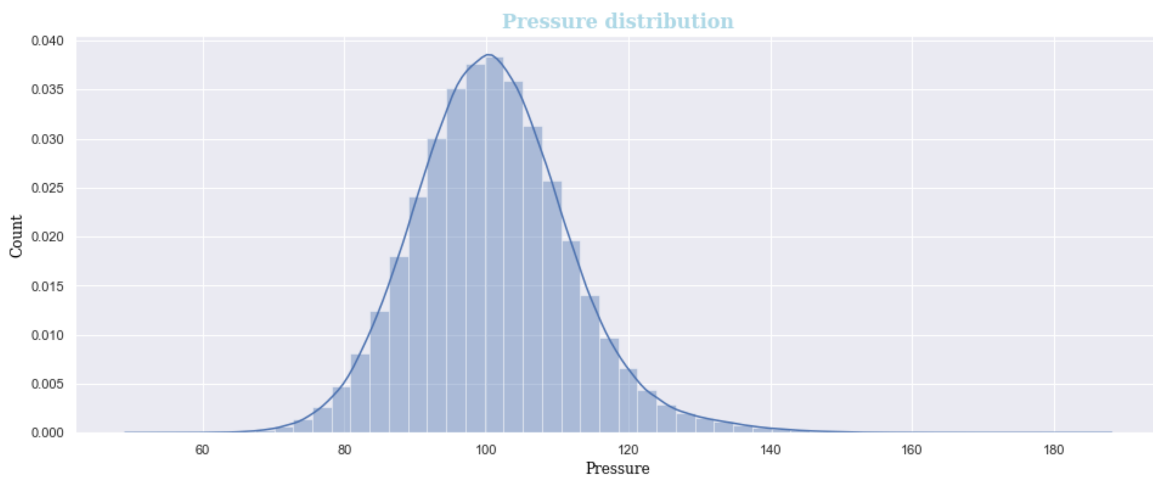


In [37]:

```
sns.set(style="darkgrid")
plt.figure(figsize=(16, 6))
sns.distplot(telemetry_df.pressure, kde=True,color="b")

plt.xlabel("Pressure", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Pressure distribution", fontdict=titlefont)
```

```
C:\Users\14802\anaconda3\lib\site-packages\seaborn\distributions.py:2551: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[37]: Text(0.5, 1.0, 'Pressure distribution')



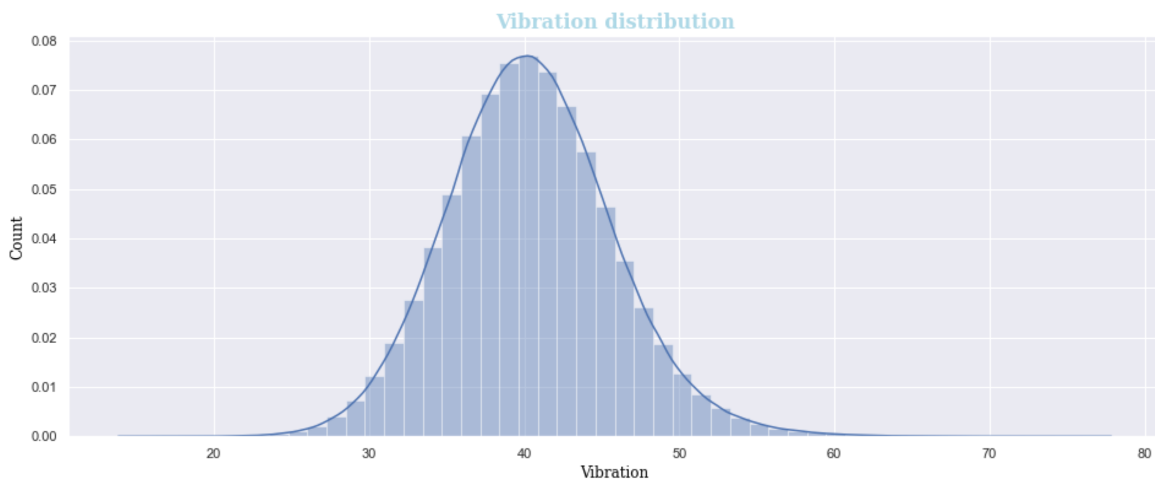
In [38]:

```
sns.set(style="darkgrid")
plt.figure(figsize=(16, 6))
sns.distplot(telemetry_df.vibration, kde=True,color="b")

plt.xlabel("Vibration", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Vibration distribution", fontdict=titlefont)
```

```
C:\Users\14802\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[38]:

```
Text(0.5, 1.0, 'Vibration distribution')
```



In [39]:

```python
# count of records by month

telemetry_month_df = telemetry_df[['machineID', 'DT']]
telemetry_month_df['month'] = pd.DatetimeIndex(telemetry_df['DT']).month
telemetry_month_df['yeat'] = pd.DatetimeIndex(telemetry_df['DT']).year
telemetry_month_df.head(5)
```

```
<ipython-input-39-6b6cef4bbe5f>:4: 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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  telemetry_month_df['month'] = pd.DatetimeIndex(telemetry_df['DT']).month
<ipython-input-39-6b6cef4bbe5f>:5: 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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  telemetry_month_df['yeat'] = pd.DatetimeIndex(telemetry_df['DT']).year
```

Out[39]:

	machineID	DT	month	yeat
0	1	2015-01-01 06:00:00	1	2015
1	1	2015-01-01 07:00:00	1	2015
2	1	2015-01-01 08:00:00	1	2015
3	1	2015-01-01 09:00:00	1	2015
4	1	2015-01-01 10:00:00	1	2015

In [40]:

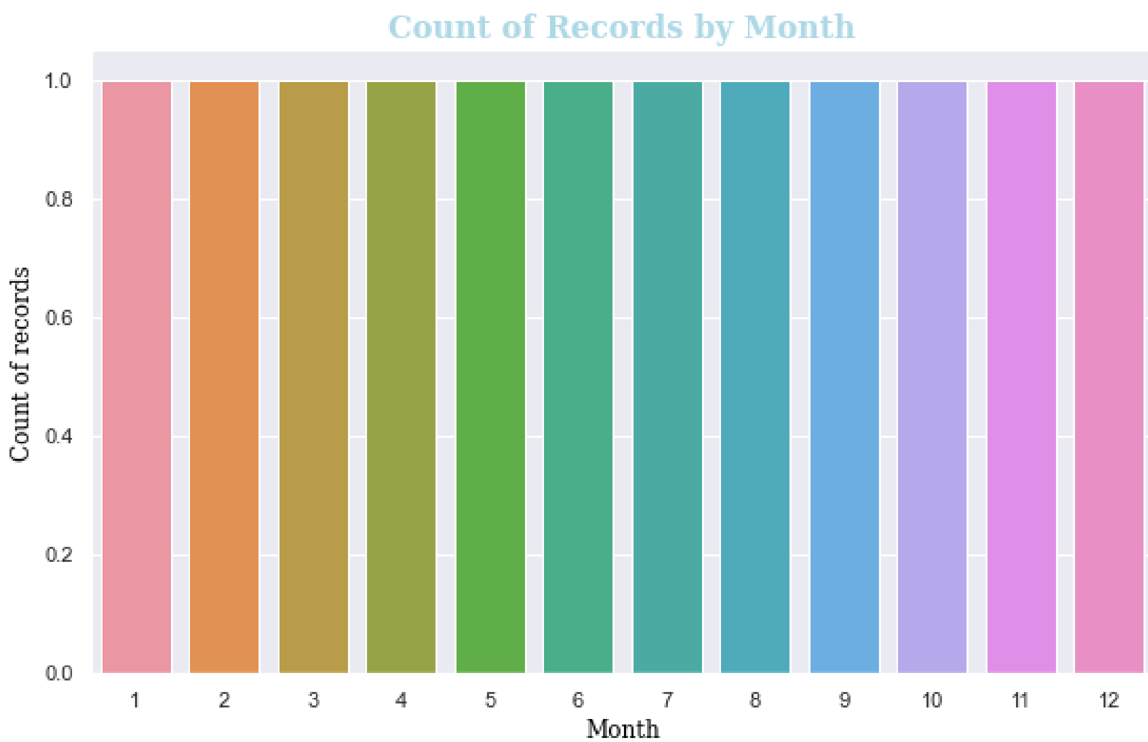
```python
telemetry_month_df2 = telemetry_month_df.groupby(['month'])['machineID'].cour
telemetry_month_df2.head()
```

Out[40]:

	month	machineID
0	1	74500
1	2	67200
2	3	74400
3	4	72000
4	5	74400

In [41]:
```python
# Draw count plot
sns.set(style="darkgrid")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="month", data=telemetry_month_df2)


plt.xlabel("Month", fontdict=labelfont)
plt.ylabel("Count of records", fontdict=labelfont)
plt.title("Count of Records by Month",fontdict=titlefont)
```

Out[41]:
```
Text(0.5, 1.0, 'Count of Records by Month')
```



In [42]:
```python
# print number of records and shape of Machines data frame
print(len(machines_df))
print(machines_df.shape)
```

```
100
(100, 3)
```

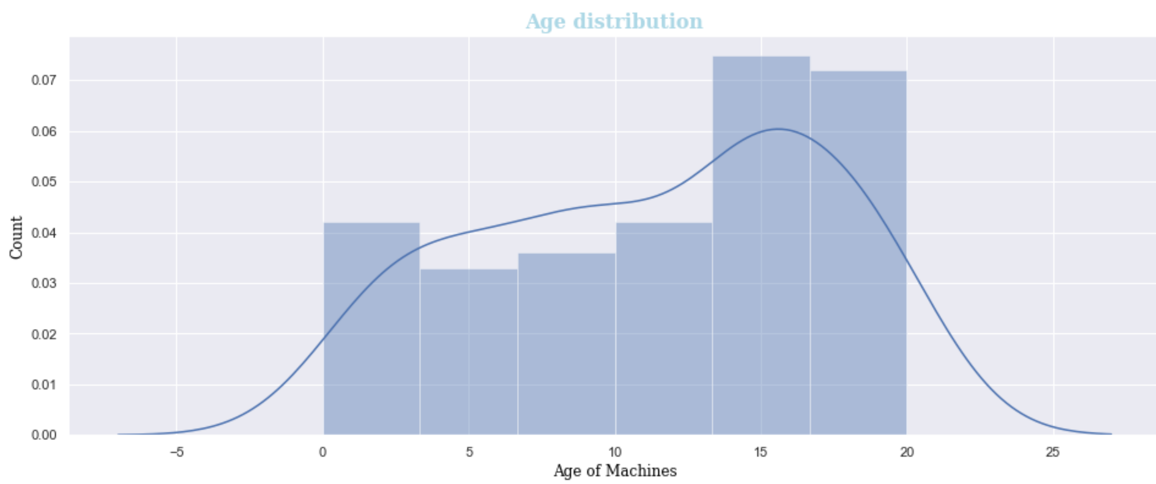
In [43]:
```python
# Checking for blank values for each column of dataframe
machines_nullcols = machines_df.isnull().sum()
print(machines_nullcols)
```

```
machineID    0
model        0
age          0
dtype: int64
```

In [44]:
```python
sns.set(style="darkgrid")
plt.figure(figsize=(16, 6))
sns.distplot(machines_df.age, kde=True,color="b")

plt.xlabel("Age of Machines", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Age distribution", fontdict=titlefont)
```

```
C:\Users\14802\anaconda3\lib\site-packages\seaborn\distributions.py:2551: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[44]: Text(0.5, 1.0, 'Age distribution')



In [45]:
```python
machines_df.columns
```

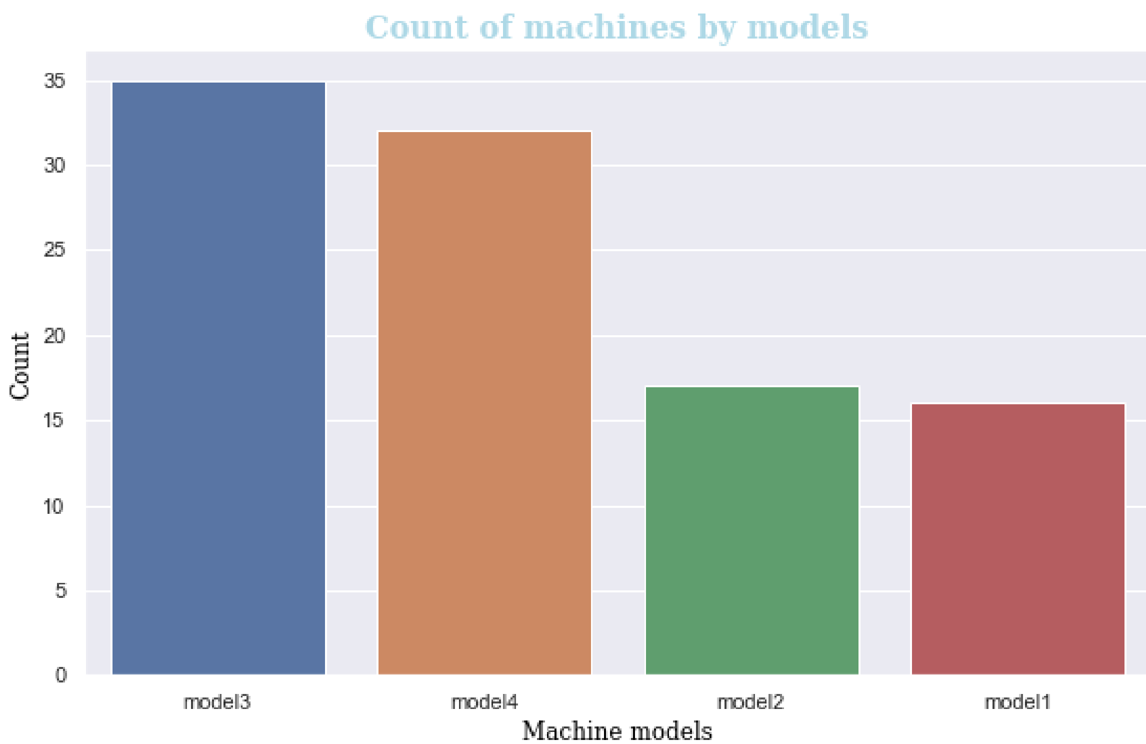
Out[45]: Index(['machineID', 'model', 'age'], dtype='object')

In [46]:

```
# Draw count plot
sns.set(style="darkgrid")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="model", data=machines_df)


plt.xlabel("Machine models", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Count of machines by models",fontdict=titlefont)
```

Out[46]:

```
Text(0.5, 1.0, 'Count of machines by models')
```



In [47]:

```
# print number of records and shape of Errors data frame
print(len(errors_df))
print(errors_df.shape)
```

```
3919
(3919, 3)
```

In [48]:
```python
# Checking for blank values for each column of dataframe
errors_nullcols = errors_df.isnull().sum()
print(errors_nullcols)
```

```
datetime     0
machineID    0
errorID      0
dtype: int64
```

In [49]:
```python
errors_df.columns
```

Out[49]:
```
Index(['datetime', 'machineID', 'errorID'], dtype='object')
```

In [50]:
```python
# Draw count plot
sns.set(style="darkgrid")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="errorID", data=errors_df)


plt.xlabel("Errors", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Count of Errors",fontdict=titlefont)
```

Out[50]:
```
Text(0.5, 1.0, 'Count of Errors')
```

In [51]:
```
# print number of records and shape of Failure data frame
print(len(failures_df))
print(failures_df.shape)
```

```
761
(761, 3)
```

In [52]:
```
# Checking for blank values for each column of dataframe
failures_nullcols = failures_df.isnull().sum()
print(failures_nullcols)
```

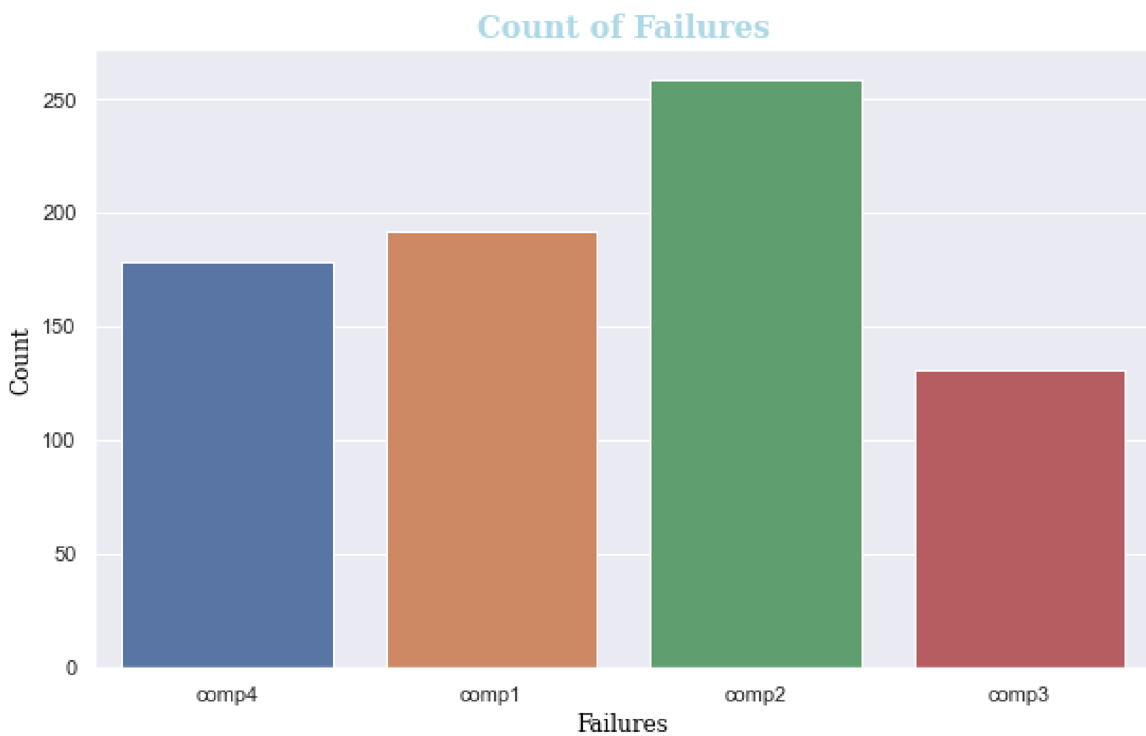
```
datetime     0
machineID    0
failure      0
dtype: int64
```

In [53]:
```
failures_df.columns
```

Out[53]:
```
Index(['datetime', 'machineID', 'failure'], dtype='object')
```

In [54]:
```
# Draw count plot
sns.set(style="darkgrid")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="failure", data=failures_df)


plt.xlabel("Failures", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Count of Failures",fontdict=titlefont)
```

Out[54]:
```
Text(0.5, 1.0, 'Count of Failures')
```



In [55]:
```
# print number of records and shape of Maintenance data frame
print(len(maint_df))
print(maint_df.shape)
```

```
3286
(3286, 3)
```

In [56]:
```
# Checking for blank values for each column of dataframe
maint_nullcols = maint_df.isnull().sum()
print(maint_nullcols)
```

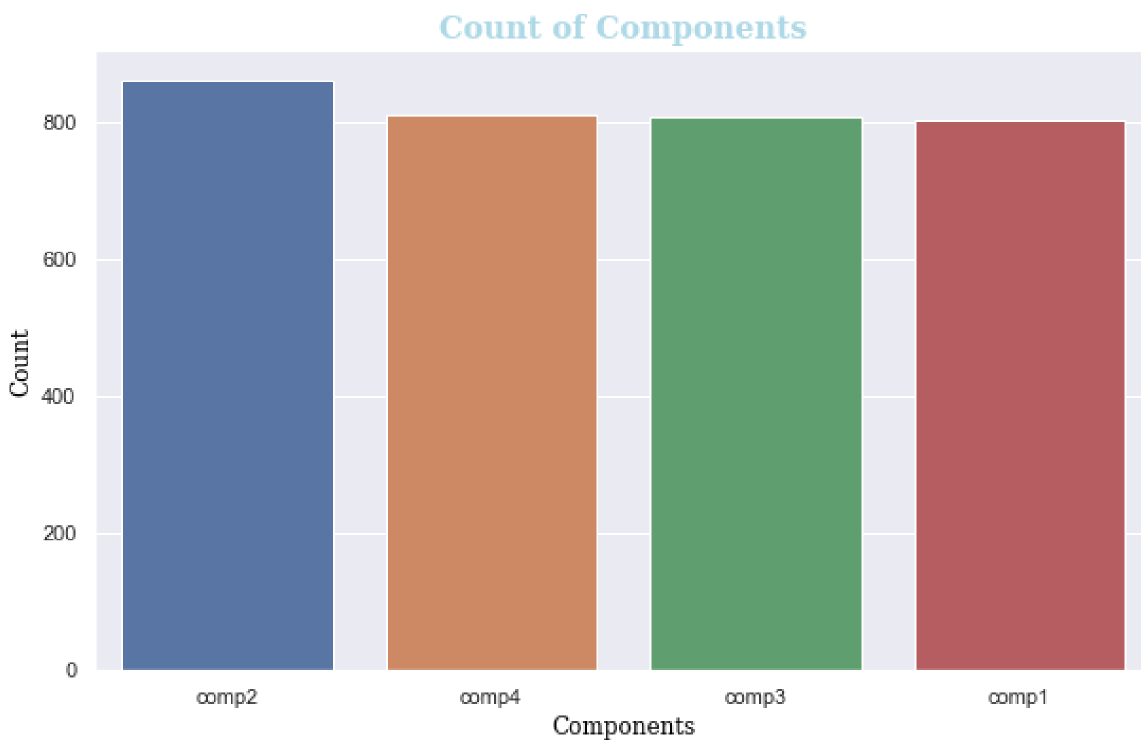
```
datetime     0
machineID    0
comp         0
dtype: int64
```

In [57]:
```
maint_df.columns
```

Out[57]:
```
Index(['datetime', 'machineID', 'comp'], dtype='object')
```

In [58]:
```
# Draw count plot
sns.set(style="darkgrid")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="comp", data=maint_df)


plt.xlabel("Components", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Count of Components",fontdict=titlefont)
```

Out[58]:
```
Text(0.5, 1.0, 'Count of Components')
```



In [ ]:

Data Preparation

Feature Engineering

In [59]:

```python
# converting all date time fields into Date Time Format

errors_df['DT'] = pd.to_datetime(errors_df['datetime'])
failures_df['DT'] = pd.to_datetime(failures_df['datetime'])
maint_df['DT'] = pd.to_datetime(maint_df['datetime'])
```

In [60]:

```python
telemetry_df.columns
```

Out[60]:

```
Index(['datetime', 'machineID', 'volt', 'rotate', 'pressure', 'vibration',
       'DT'],
      dtype='object')
```

for dt in telemetry_df['DT']: print(dt.strftime("%X"))

In [61]:

```python
telemetry_df['time'] = [dt.strftime("%H") for dt in telemetry_df['DT']]
telemetry_df.head(5)
```

Out[61]:

	datetime	machineID	volt	rotate	pressure	vibration	DT	time
0	1/1/2015 6:00:00 AM	1	176.217853	418.504078	113.077935	45.087686	2015-01-01 06:00:00	06
1	1/1/2015 7:00:00 AM	1	162.879223	402.747490	95.460525	43.413973	2015-01-01 07:00:00	07
2	1/1/2015 8:00:00 AM	1	170.989902	527.349825	75.237905	34.178847	2015-01-01 08:00:00	08
3	1/1/2015 9:00:00 AM	1	162.462833	346.149335	109.248561	41.122144	2015-01-01 09:00:00	09
4	1/1/2015 10:00:00 AM	1	157.610021	435.376873	111.886648	25.990511	2015-01-01 10:00:00	10

In [62]:

```python
for time in telemetry_df['time'][0:5]:
        print(type(int(time)))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
```

In [63]:

```
telemetry_df['hrbucket'] = [(int(time) // 3) for time in telemetry_df['time']
telemetry_df.head(5)
```

Out[63]:

	datetime	machineID	volt	rotate	pressure	vibration	DT	time	hrbuck
0	1/1/2015 6:00:00 AM	1	176.217853	418.504078	113.077935	45.087686	2015-01-01 06:00:00	06	
1	1/1/2015 7:00:00 AM	1	162.879223	402.747490	95.460525	43.413973	2015-01-01 07:00:00	07	
2	1/1/2015 8:00:00 AM	1	170.989902	527.349825	75.237905	34.178847	2015-01-01 08:00:00	08	
3	1/1/2015 9:00:00 AM	1	162.462833	346.149335	109.248561	41.122144	2015-01-01 09:00:00	09	
4	1/1/2015 10:00:00 AM	1	157.610021	435.376873	111.886648	25.990511	2015-01-01 10:00:00	10	

In [64]:

```
#telemetry_df['weekday'] = [dt.weekday() for dt in telemetry_df['DT']]
telemetry_df['weekday'] = [dt.strftime("%A") for dt in telemetry_df['DT']]
telemetry_df['date'] = [dt.strftime("%x") for dt in telemetry_df['DT']]
telemetry_df.head(5)
```

Out[64]:

	datetime	machineID	volt	rotate	pressure	vibration	DT	time	hrbuck
0	1/1/2015 6:00:00 AM	1	176.217853	418.504078	113.077935	45.087686	2015-01-01 06:00:00	06	
1	1/1/2015 7:00:00 AM	1	162.879223	402.747490	95.460525	43.413973	2015-01-01 07:00:00	07	
2	1/1/2015 8:00:00 AM	1	170.989902	527.349825	75.237905	34.178847	2015-01-01 08:00:00	08	
3	1/1/2015 9:00:00 AM	1	162.462833	346.149335	109.248561	41.122144	2015-01-01 09:00:00	09	
4	1/1/2015 10:00:00 AM	1	157.610021	435.376873	111.886648	25.990511	2015-01-01 10:00:00	10	

In [65]:

```python
# grouping data by machineID, date and hrbucket calculating mean of metrics

telemetry_df2 = telemetry_df.groupby(['machineID','date', 'hrbucket']).aggreg

telemetry_df2.head()
```

Out[65]:

machineID	date	hrbucket	DT	volt	rotate	pressure	vibration
1	**01/01/15**	**2**	2015-01-01 08:00:00	6.721032	67.849599	18.934956	5.874970
		3	2015-01-01 11:00:00	7.596570	50.120452	8.555032	7.662229
		4	2015-01-01 14:00:00	10.124584	55.084734	5.909721	5.169304
		5	2015-01-01 17:00:00	4.673269	42.047278	4.554047	2.106108
		6	2015-01-01 20:00:00	14.752132	47.048609	4.244158	2.207884

In [66]:

```python
# grouping data by machineID, date and hrbucket calculating mean of metrics

telemetry_df2 = telemetry_df.groupby(['machineID','date', 'hrbucket']).aggreg




telemetry_df2.head()
```

Out[66]:

	machineID	date	hrbucket	DT	volt		rotate		pres
				max	mean	std	mean	std	mea
0	1	01/01/15	2	2015-01-01 08:00:00	170.028993	6.721032	449.533798	67.849599	94
1	1	01/01/15	3	2015-01-01 11:00:00	164.192565	7.596570	403.949857	50.120452	105
2	1	01/01/15	4	2015-01-01 14:00:00	168.134445	10.124584	435.781707	55.084734	107
3	1	01/01/15	5	2015-01-01 17:00:00	165.514453	4.673269	430.472823	42.047278	101
4	1	01/01/15	6	2015-01-01 20:00:00	168.809347	14.752132	437.111120	47.048609	90

In [67]:

```python
# grouping data by machineID, date and hrbucket calculating mean of metrics

telemetry_df2 = telemetry_df.groupby(['machineID','date', 'hrbucket']).aggreg




telemetry_df2.head()
```

Out[67]:

	machineID	date	hrbucket	DT	volt		rotate		pres
				max	mean	std	mean	std	mea
0	1	01/01/15	2	2015-01-01 08:00:00	170.028993	6.721032	449.533798	67.849599	94
1	1	01/01/15	3	2015-01-01 11:00:00	164.192565	7.596570	403.949857	50.120452	105
2	1	01/01/15	4	2015-01-01 14:00:00	168.134445	10.124584	435.781707	55.084734	107
3	1	01/01/15	5	2015-01-01 17:00:00	165.514453	4.673269	430.472823	42.047278	101
4	1	01/01/15	6	2015-01-01 20:00:00	168.809347	14.752132	437.111120	47.048609	90

In [68]:

```python
# grouping data by machineID, date and hrbucket calculating mean of metrics

telemetry_df2 = telemetry_df.groupby(['machineID','date', 'hrbucket'])\
                            .agg({ 'DT' : [('m_DT', 'max')],
                                   'volt' : [('m_volt','mean'), ('sd_volt','st
                                   'rotate' : [('m_rotate','mean'), ('sd_rotat
                                   'pressure' : [('m_pressure','mean'), ('sd_p
                                    'vibration' : [('m_vibration','mean'), ('s
                                    }).reset_index()

telemetry_df2.head()
```

Out[68]:

	machineID	date	hrbucket	DT	volt		rotate		pres
				m_DT	m_volt	sd_volt	m_rotate	sd_rotate	m_p
0	1	01/01/15	2	2015-01-01 08:00:00	170.028993	6.721032	449.533798	67.849599	94
1	1	01/01/15	3	2015-01-01 11:00:00	164.192565	7.596570	403.949857	50.120452	105
2	1	01/01/15	4	2015-01-01 14:00:00	168.134445	10.124584	435.781707	55.084734	107
3	1	01/01/15	5	2015-01-01 17:00:00	165.514453	4.673269	430.472823	42.047278	10
4	1	01/01/15	6	2015-01-01 20:00:00	168.809347	14.752132	437.111120	47.048609	9

In [69]:

```python
telemetry_df2.columns
```

Out[69]:

```
MultiIndex([('machineID',             ''),
            (     'date',             ''),
            ( 'hrbucket',             ''),
            (       'DT',         'm_DT'),
            (     'volt',       'm_volt'),
            (     'volt',      'sd_volt'),
            (   'rotate',     'm_rotate'),
            (   'rotate',    'sd_rotate'),
            ( 'pressure',   'm_pressure'),
            ( 'pressure',  'sd_pressure'),
            ('vibration',  'm_vibration'),
            ('vibration', 'sd_vibration')],
           )
```

In [70]:
```
telemetry_df2.columns = telemetry_df2.columns.get_level_values(0)
telemetry_df2.columns = ['machineID', 'date', 'hrbucket', 'DT', 'm_volt', 'sc
       'sd_rotate', 'm_pressure', 'sd_pressure', 'm_vibration', 'sd_vibratior
```

In [71]:
```
telemetry_df2.head()
```

Out[71]:

	machineID	date	hrbucket	DT	m_volt	sd_volt	m_rotate	sd_rotate	m_p
0	1	01/01/15	2	2015-01-01 08:00:00	170.028993	6.721032	449.533798	67.849599	94
1	1	01/01/15	3	2015-01-01 11:00:00	164.192565	7.596570	403.949857	50.120452	105
2	1	01/01/15	4	2015-01-01 14:00:00	168.134445	10.124584	435.781707	55.084734	107
3	1	01/01/15	5	2015-01-01 17:00:00	165.514453	4.673269	430.472823	42.047278	101
4	1	01/01/15	6	2015-01-01 20:00:00	168.809347	14.752132	437.111120	47.048609	90

Calculate mean and standard deviation of metrics for a rolling 24 hour windows

In [72]:
```
volt_mean_24 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='volt').rolling(window=24).mean().resam



print(type(volt_mean_24))
```

```
<class 'pandas.core.series.Series'>
```

In [73]:
```
volt_std_24 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='volt').rolling(window=24).std().resamp



print(type(volt_std_24))
```

```
<class 'pandas.core.series.Series'>
```

In [74]:

```
rotate_mean_24 = pd.pivot_table(telemetry_df,
                               index='DT',
                               columns='machineID',
                               values='rotate').rolling(window=24).mean().res

rotate_std_24 = pd.pivot_table(telemetry_df,
                               index='DT',
                               columns='machineID',
                               values='rotate').rolling(window=24).std().resa
```

In [75]:

```
pressure_mean_24 = pd.pivot_table(telemetry_df,
                               index='DT',
                               columns='machineID',
                               values='pressure').rolling(window=24).mean().r

pressure_std_24 = pd.pivot_table(telemetry_df,
                               index='DT',
                               columns='machineID',
                               values='pressure').rolling(window=24).std().re
```

In [76]:

```
vibration_mean_24 = pd.pivot_table(telemetry_df,
                               index='DT',
                               columns='machineID',
                               values='vibration').rolling(window=24).mean().

vibration_std_24 = pd.pivot_table(telemetry_df,
                               index='DT',
                               columns='machineID',
                               values='vibration').rolling(window=24).std().r
```

In [77]:
```
list_24hrs = [volt_mean_24, volt_std_24, rotate_mean_24 ,rotate_std_24 , pres
telemetry_24df = pd.concat(list_24hrs, axis=1 )
telemetry_24df.head(10)
```

Out[77]:

		0	1	2	3	4	5	6
machineID	**DT**							
1	**2015-01-01 09:00:00**	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-01 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-01 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-01 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-01 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-02 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-02 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
	2015-01-02 06:00:00	169.733809	11.233120	445.179865	48.717395	96.797113	10.079880	40.3
	2015-01-02 09:00:00	170.614862	12.519402	446.364859	48.385076	96.849785	10.171540	39.7
	2015-01-02 12:00:00	169.893965	13.370357	447.009407	42.432317	97.715600	9.471669	39.4

In [78]:
```
telemetry_24df.columns
```

Out[78]:
```
RangeIndex(start=0, stop=8, step=1)
```

In [79]:

```python
telemetry_24df.columns = ['volt_mean_24', 'volt_std_24', 'rotate_mean_24' ,'r
                          'pressure_std_24', 'vibration_mean_24', 'vibration_
telemetry_24df = telemetry_24df.reset_index()
telemetry_24df.head(10)
```

Out[79]:

	machineID	DT	volt_mean_24	volt_std_24	rotate_mean_24	rotate_std_24	pressure_m
0	1	2015-01-01 09:00:00	NaN	NaN	NaN	NaN	
1	1	2015-01-01 12:00:00	NaN	NaN	NaN	NaN	
2	1	2015-01-01 15:00:00	NaN	NaN	NaN	NaN	
3	1	2015-01-01 18:00:00	NaN	NaN	NaN	NaN	
4	1	2015-01-01 21:00:00	NaN	NaN	NaN	NaN	
5	1	2015-01-02 00:00:00	NaN	NaN	NaN	NaN	
6	1	2015-01-02 03:00:00	NaN	NaN	NaN	NaN	
7	1	2015-01-02 06:00:00	169.733809	11.233120	445.179865	48.717395	96
8	1	2015-01-02 09:00:00	170.614862	12.519402	446.364859	48.385076	96.
9	1	2015-01-02 12:00:00	169.893965	13.370357	447.009407	42.432317	97.

In [80]:

```
volt_mean_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='volt').rolling(window=3).mean().resamp



volt_std_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='volt').rolling(window=3).std().resampl



rotate_mean_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='rotate').rolling(window=3).mean().resa



rotate_std_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='rotate').rolling(window=3).std().resam



pressure_mean_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='pressure').rolling(window=3).mean().re



pressure_std_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='pressure').rolling(window=3).std().res



vibration_mean_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='vibration').rolling(window=3).mean().r



vibration_std_3 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
```

```
                                    values='vibration').rolling(window=3).std().re
```

In [81]:

```
list_3hrs = [volt_mean_3, volt_std_3, rotate_mean_3,rotate_std_3, pressure_me
             vibration_mean_3, vibration_std_3]
telemetry_3df = pd.concat(list_3hrs, axis=1 )
telemetry_3df.head(10)
```

Out[81]:

machineID	DT	0	1	2	3	4	5	6
1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.592122	18.934956	40.
	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.315664	17.106476	39.
	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.523125	9.176711	34.
	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.491224	4.843754	40.
	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.424693	8.931082	41.
	2015-01-02 00:00:00	178.513887	7.084050	440.859804	53.461091	90.827785	4.388335	44.
	2015-01-02 03:00:00	167.989524	14.910036	481.173775	33.823675	94.103572	2.705111	37.
	2015-01-02 06:00:00	165.002124	5.959072	456.438211	54.613403	87.673270	7.623486	41.
	2015-01-02 09:00:00	193.164359	10.459846	448.652619	46.294659	101.438946	11.281152	38.
	2015-01-02 12:00:00	159.676811	11.972868	430.571937	51.632304	100.242184	12.041639	37.

In [82]:

```
telemetry_3df.columns = ['volt_mean_3', 'volt_std_3', 'rotate_mean_3' ,'rotat
                         'pressure_std_3', 'vibration_mean_3', 'vibration_st
telemetry_3df = telemetry_3df.reset_index()
telemetry_3df.head(10)
```

Out[82]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean_
0	1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.59212
1	1	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.31566
2	1	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.52312
3	1	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.49122
4	1	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.42469
5	1	2015-01-02 00:00:00	178.513887	7.084050	440.859804	53.461091	90.82778
6	1	2015-01-02 03:00:00	167.989524	14.910036	481.173775	33.823675	94.10357
7	1	2015-01-02 06:00:00	165.002124	5.959072	456.438211	54.613403	87.67327
8	1	2015-01-02 09:00:00	193.164359	10.459846	448.652619	46.294659	101.43894
9	1	2015-01-02 12:00:00	159.676811	11.972868	430.571937	51.632304	100.24218

In [83]:
```python
# creating combined data set of summarized data
telemetry_summarydf = pd.merge(telemetry_3df,telemetry_24df, left_on = ['mach
                               right_on = ['machineID','DT'], how='left').fillna(0)

telemetry_summarydf.head(10)
```

Out[83]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean_
0	1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.59212
1	1	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.31566
2	1	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.52312
3	1	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.49122
4	1	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.42469
5	1	2015-01-02 00:00:00	178.513887	7.084050	440.859804	53.461091	90.82778
6	1	2015-01-02 03:00:00	167.989524	14.910036	481.173775	33.823675	94.10357
7	1	2015-01-02 06:00:00	165.002124	5.959072	456.438211	54.613403	87.67327
8	1	2015-01-02 09:00:00	193.164359	10.459846	448.652619	46.294659	101.43894
9	1	2015-01-02 12:00:00	159.676811	11.972868	430.571937	51.632304	100.24218

Creating summary data for errors_df

In [84]:
```python
errors_df.columns
```

Out[84]: Index(['datetime', 'machineID', 'errorID', 'DT'], dtype='object')

In [85]:
```python
# grouping data by machineID, date and hrbucket calculating mean of metrics

errorcounts_df = errors_df.groupby(['machineID','DT', 'errorID'])\
                            .agg({ 'errorID' : [('count','count')]
                                   }).reset_index()

errorcounts_df.head()
```

Out[85]:

	machineID	DT	errorID	
				count
0	1	2015-01-03 07:00:00	error1	1
1	1	2015-01-03 20:00:00	error3	1
2	1	2015-01-04 06:00:00	error5	1
3	1	2015-01-10 15:00:00	error4	1
4	1	2015-01-22 10:00:00	error4	1

In [86]:
```python
errorcounts_df.columns = errorcounts_df.columns.get_level_values(0)
errorcounts_df.columns = ['machineID', 'DT', 'errorID', 'errorcount']
```

In [87]:
```python
errorcounts_ctdf = pd.crosstab([errorcounts_df.machineID, errorcounts_df.DT],
errorcounts_ctdf = errorcounts_ctdf.sort_values('DT')
errorcounts_ctdf.head()
```

Out[87]:

errorID	machineID	DT	error1	error2	error3	error4	error5
2874	81	2015-01-01 06:00:00	1	0	0	0	0
836	24	2015-01-01 06:00:00	1	0	0	0	0
2579	73	2015-01-01 06:00:00	0	0	0	1	0
1497	43	2015-01-01 07:00:00	0	0	1	0	0
2683	76	2015-01-01 08:00:00	0	0	0	0	1

In [88]:
```python
print(len(errorcounts_ctdf))
```

```
3616
```

In [89]:
```
# joining error and telemetry data based on date time and machine ID
telemetry_dts = telemetry_df[['machineID','DT']]
telemetry_dts.head(5)
```

Out[89]:

	machineID	DT
0	1	2015-01-01 06:00:00
1	1	2015-01-01 07:00:00
2	1	2015-01-01 08:00:00
3	1	2015-01-01 09:00:00
4	1	2015-01-01 10:00:00

In [90]:
```
errorcounts_dtdf = pd.merge(telemetry_dts,errorcounts_ctdf, left_on = ['machi
                        right_on = ['machineID','DT'], how='left').fillna(0)

errorcounts_dtdf.head(5)
```

Out[90]:

	machineID	DT	error1	error2	error3	error4	error5
0	1	2015-01-01 06:00:00	0.0	0.0	0.0	0.0	0.0
1	1	2015-01-01 07:00:00	0.0	0.0	0.0	0.0	0.0
2	1	2015-01-01 08:00:00	0.0	0.0	0.0	0.0	0.0
3	1	2015-01-01 09:00:00	0.0	0.0	0.0	0.0	0.0
4	1	2015-01-01 10:00:00	0.0	0.0	0.0	0.0	0.0

In [91]:

```
# summarize the errors for every 3 hours to includes errors occured in the la

error1_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error1').rolling(window=24).sum().resa



error2_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error2').rolling(window=24).sum().resa



error3_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error3').rolling(window=24).sum().resa



error4_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error4').rolling(window=24).sum().resa



error5_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error5').rolling(window=24).sum().resa
```

In [92]:

```
list_counts = [error1_count, error2_count, error3_count, error4_count, error5
error_sum_df = pd.concat(list_counts, axis=1 )
error_sum_df.head(10)
```

Out[92]:

machineID	DT	0	1	2	3	4
1	2015-01-01 09:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-01 12:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-01 15:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-01 18:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-01 21:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-02 00:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-02 03:00:00	NaN	NaN	NaN	NaN	NaN
	2015-01-02 06:00:00	0.0	0.0	0.0	0.0	0.0
	2015-01-02 09:00:00	0.0	0.0	0.0	0.0	0.0
	2015-01-02 12:00:00	0.0	0.0	0.0	0.0	0.0

In [93]:

```
error_sum_df.columns = ['error1_count', 'error2_count', 'error3_count', 'errc
error_sum_df = error_sum_df.reset_index()
error_sum_df = error_sum_df.fillna(0)
error_sum_df.head(10)
```

Out[93]:

	machineID	DT	error1_count	error2_count	error3_count	error4_count	error5_count
0	1	2015-01-01 09:00:00	0.0	0.0	0.0	0.0	0.0
1	1	2015-01-01 12:00:00	0.0	0.0	0.0	0.0	0.0
2	1	2015-01-01 15:00:00	0.0	0.0	0.0	0.0	0.0
3	1	2015-01-01 18:00:00	0.0	0.0	0.0	0.0	0.0
4	1	2015-01-01 21:00:00	0.0	0.0	0.0	0.0	0.0
5	1	2015-01-02 00:00:00	0.0	0.0	0.0	0.0	0.0
6	1	2015-01-02 03:00:00	0.0	0.0	0.0	0.0	0.0
7	1	2015-01-02 06:00:00	0.0	0.0	0.0	0.0	0.0
8	1	2015-01-02 09:00:00	0.0	0.0	0.0	0.0	0.0
9	1	2015-01-02 12:00:00	0.0	0.0	0.0	0.0	0.0

Featuring Eningeering of Maintenance data

In [94]:

```
maint_df.columns
```

Out[94]:

```
Index(['datetime', 'machineID', 'comp', 'DT'], dtype='object')
```

In [95]:
```python
# grouping data by machineID, date and hrbucket calculating mean of metrics

maint_df2 = maint_df.groupby(['machineID','datetime', 'comp', 'DT'])\
                            .agg({'comp' : [('compcount','count')]
                                   }).reset_index()

maint_df2.head()
```

Out[95]:

	machineID	datetime	comp	DT	comp
					compcount
0	1	1/20/2015 6:00:00 AM	comp1	2015-01-20 06:00:00	1
1	1	1/20/2015 6:00:00 AM	comp3	2015-01-20 06:00:00	1
2	1	1/5/2015 6:00:00 AM	comp1	2015-01-05 06:00:00	1
3	1	1/5/2015 6:00:00 AM	comp4	2015-01-05 06:00:00	1
4	1	10/17/2015 6:00:00 AM	comp2	2015-10-17 06:00:00	1

In [96]:
```python
maint_df2.columns = maint_df2.columns.get_level_values(0)
maint_df2.columns = ['machineID', 'datetime', 'comp', 'DT', 'compcount']
maint_df2.columns
```

Out[96]: Index(['machineID', 'datetime', 'comp', 'DT', 'compcount'], dtype='object')

In [97]:
```python
maint_df3 = pd.crosstab([maint_df2.machineID, maint_df2.datetime, maint_df2.D
maint_df3.head()
```

Out[97]:

comp	machineID	datetime	DT	comp1	comp2	comp3	comp4
0	1	1/20/2015 6:00:00 AM	2015-01-20 06:00:00	1	0	1	0
1	1	1/5/2015 6:00:00 AM	2015-01-05 06:00:00	1	0	0	1
2	1	10/17/2015 6:00:00 AM	2015-10-17 06:00:00	0	1	0	1
3	1	10/2/2015 6:00:00 AM	2015-10-02 06:00:00	1	0	0	1
4	1	11/1/2015 6:00:00 AM	2015-11-01 06:00:00	0	1	0	1

In [98]:

```
telemetry_dts.head(5)
```

Out[98]:

	machineID	DT
0	1	2015-01-01 06:00:00
1	1	2015-01-01 07:00:00
2	1	2015-01-01 08:00:00
3	1	2015-01-01 09:00:00
4	1	2015-01-01 10:00:00

In [99]:

```python
# merging with telemetry datetimes

maintcounts_dtdf = pd.merge(telemetry_dts,maint_df3, left_on = ['machineID','
                        right_on = ['machineID','DT'], how='outer').fillna(0)

maintcounts_dtdf.head(15)
```

Out[99]:

	machineID	DT	datetime	comp1	comp2	comp3	comp4
876101	1	2014-06-01 06:00:00	6/1/2014 6:00:00 AM	0.0	1.0	0.0	0.0
876102	1	2014-07-16 06:00:00	7/16/2014 6:00:00 AM	0.0	0.0	0.0	1.0
876103	1	2014-07-31 06:00:00	7/31/2014 6:00:00 AM	0.0	0.0	1.0	0.0
876100	1	2014-12-13 06:00:00	12/13/2014 6:00:00 AM	1.0	0.0	0.0	0.0
0	1	2015-01-01 06:00:00	0	0.0	0.0	0.0	0.0
1	1	2015-01-01 07:00:00	0	0.0	0.0	0.0	0.0
2	1	2015-01-01 08:00:00	0	0.0	0.0	0.0	0.0
3	1	2015-01-01 09:00:00	0	0.0	0.0	0.0	0.0
4	1	2015-01-01 10:00:00	0	0.0	0.0	0.0	0.0
5	1	2015-01-01 11:00:00	0	0.0	0.0	0.0	0.0
6	1	2015-01-01 12:00:00	0	0.0	0.0	0.0	0.0
7	1	2015-01-01 13:00:00	0	0.0	0.0	0.0	0.0
8	1	2015-01-01 14:00:00	0	0.0	0.0	0.0	0.0
9	1	2015-01-01 15:00:00	0	0.0	0.0	0.0	0.0
10	1	2015-01-01 16:00:00	0	0.0	0.0	0.0	0.0

In [100]:

```python
maintcounts_dtdf_comp1 = maintcounts_dtdf[maintcounts_dtdf['comp1'] == 1.0].s
maintcounts_dtdf_comp1['comp1rank'] = maintcounts_dtdf_comp1.groupby('machine
maintcounts_dtdf_comp1['comp1prevrank'] = maintcounts_dtdf_comp1['comp1rank']
maintcounts_dtdf_comp1 = maintcounts_dtdf_comp1.drop(['comp2', 'comp3', 'comp
maintcounts_dtdf_comp1.head(5)
maintcounts_dtdf_comp1_df2 = maintcounts_dtdf_comp1
```

In [101]:

```python
maintcounts_dtdf_comp1_lpdf = pd.merge(maintcounts_dtdf_comp1, maintcounts_dt
                                       left_on = ['machineID', 'comp1rank'], r
                                       how = 'outer')

maintcounts_dtdf_comp1_lpdf.columns
```

Out[101]:

```
Index(['machineID', 'DT_x', 'datetime_x', 'comp1_x', 'comp1rank_x',
       'comp1prevrank_x', 'DT_y', 'datetime_y', 'comp1_y', 'comp1rank_y',
       'comp1prevrank_y'],
      dtype='object')
```

In [102]:

```python
maintcounts_dtdf_comp1_lpdf = maintcounts_dtdf_comp1_lpdf.drop([ 'comp1prevra
       'comp1prevrank_y'], axis = 1)
maintcounts_dtdf_comp1_lpdf.columns
```

Out[102]:

```
Index(['machineID', 'DT_x', 'datetime_x', 'comp1_x', 'comp1rank_x', 'DT_
y'], dtype='object')
```

In [103]:

```python
maintcounts_dtdf_comp1_lpdf.columns = ['machineID', 'DT', 'datetime_x', 'comp

maintcounts_dtdf_comp1_lpdf['comp1Lastreplaceddt'] = maintcounts_dtdf_comp1_l
                                                          .fillna(method =
maintcounts_dtdf_comp1_lpdf.head()
```

Out[103]:

	machineID	DT	datetime_x	comp1_x	comp1rank_x	comp1Lastreplaceddt
0	1	2014-12-13 06:00:00	12/13/2014 6:00:00 AM	1.0	1.0	2014-12-13 06:00:00
1	1	2015-01-05 06:00:00	1/5/2015 6:00:00 AM	1.0	2.0	2014-12-13 06:00:00
2	1	2015-01-20 06:00:00	1/20/2015 6:00:00 AM	1.0	3.0	2015-01-05 06:00:00
3	1	2015-03-06 06:00:00	3/6/2015 6:00:00 AM	1.0	4.0	2015-01-20 06:00:00
4	1	2015-03-21 06:00:00	3/21/2015 6:00:00 AM	1.0	5.0	2015-03-06 06:00:00

In [104]:

```
maintcounts_dtdf_comp2 = maintcounts_dtdf[maintcounts_dtdf['comp2'] == 1.0].s
maintcounts_dtdf_comp2['comp2rank'] = maintcounts_dtdf_comp2.groupby('machine
maintcounts_dtdf_comp2['comp2prevrank'] = maintcounts_dtdf_comp2['comp2rank']
maintcounts_dtdf_comp2 = maintcounts_dtdf_comp2.drop(['comp1', 'comp3', 'comp

maintcounts_dtdf_comp2_df2 = maintcounts_dtdf_comp2


maintcounts_dtdf_comp2_lpdf = pd.merge(maintcounts_dtdf_comp2, maintcounts_dt
                                       left_on = ['machineID', 'comp2rank'], r
                                       how = 'outer')

maintcounts_dtdf_comp2_lpdf = maintcounts_dtdf_comp2_lpdf.drop([ 'comp2prevra
       'comp2prevrank_y'], axis = 1)


maintcounts_dtdf_comp2_lpdf.columns = ['machineID', 'DT', 'datetime_x', 'comp

maintcounts_dtdf_comp2_lpdf['comp2Lastreplaceddt'] = maintcounts_dtdf_comp2_l
                                                            .fillna(method =
maintcounts_dtdf_comp2_lpdf.head(5)
```

Out[104]:

	machineID	DT	datetime_x	comp2_x	comp2rank_x	comp2Lastreplaceddt
0	1	2014-06-01 06:00:00	6/1/2014 6:00:00 AM	1.0	1.0	2014-06-01 06:00:00
1	1	2015-04-20 06:00:00	4/20/2015 6:00:00 AM	1.0	2.0	2014-06-01 06:00:00
2	1	2015-05-05 06:00:00	5/5/2015 6:00:00 AM	1.0	3.0	2015-04-20 06:00:00
3	1	2015-05-20 06:00:00	5/20/2015 6:00:00 AM	1.0	4.0	2015-05-05 06:00:00
4	1	2015-07-04 06:00:00	7/4/2015 6:00:00 AM	1.0	5.0	2015-05-20 06:00:00

In [105]:

```
maintcounts_dtdf_comp3 = maintcounts_dtdf[maintcounts_dtdf['comp3'] == 1.0].s
maintcounts_dtdf_comp3['comp3rank'] = maintcounts_dtdf_comp3.groupby('machine
maintcounts_dtdf_comp3['comp3prevrank'] = maintcounts_dtdf_comp3['comp3rank']
maintcounts_dtdf_comp3 = maintcounts_dtdf_comp3.drop(['comp1', 'comp2', 'comp

maintcounts_dtdf_comp3_df2 = maintcounts_dtdf_comp3


maintcounts_dtdf_comp3_lpdf = pd.merge(maintcounts_dtdf_comp3, maintcounts_dt
                                       left_on = ['machineID', 'comp3rank'], r
                                       how = 'outer')

maintcounts_dtdf_comp3_lpdf = maintcounts_dtdf_comp3_lpdf.drop([ 'comp3prevra
       'comp3prevrank_y'], axis = 1)


maintcounts_dtdf_comp3_lpdf.columns = ['machineID', 'DT', 'datetime_x', 'comp
maintcounts_dtdf_comp3_lpdf['comp3Lastreplaceddt'] = maintcounts_dtdf_comp3_l
                                                                     .fillna(method =
maintcounts_dtdf_comp3_lpdf.head(5)
```

Out[105]:

	machineID	DT	datetime_x	comp3_x	comp3rank_x	comp3Lastreplaceddt
0	1	2014-07-31 06:00:00	7/31/2014 6:00:00 AM	1.0	1.0	2014-07-31 06:00:00
1	1	2015-01-20 06:00:00	1/20/2015 6:00:00 AM	1.0	2.0	2014-07-31 06:00:00
2	1	2015-02-04 06:00:00	2/4/2015 6:00:00 AM	1.0	3.0	2015-01-20 06:00:00
3	1	2015-02-19 06:00:00	2/19/2015 6:00:00 AM	1.0	4.0	2015-02-04 06:00:00
4	1	2015-04-05 06:00:00	4/5/2015 6:00:00 AM	1.0	5.0	2015-02-19 06:00:00

In [106]:

```
maintcounts_dtdf_comp4 = maintcounts_dtdf[maintcounts_dtdf['comp4'] == 1.0].s
maintcounts_dtdf_comp4['comp4rank'] = maintcounts_dtdf_comp4.groupby('machine
maintcounts_dtdf_comp4['comp4prevrank'] = maintcounts_dtdf_comp4['comp4rank']
maintcounts_dtdf_comp4 = maintcounts_dtdf_comp4.drop(['comp1', 'comp2', 'comp

maintcounts_dtdf_comp4_df2 = maintcounts_dtdf_comp4


maintcounts_dtdf_comp4_lpdf = pd.merge(maintcounts_dtdf_comp4, maintcounts_dt
                                       left_on = ['machineID', 'comp4rank'], r
                                       how = 'outer')

maintcounts_dtdf_comp4_lpdf = maintcounts_dtdf_comp4_lpdf.drop([ 'comp4prevra
       'comp4prevrank_y'], axis = 1)


maintcounts_dtdf_comp4_lpdf.columns = ['machineID', 'DT', 'datetime_x', 'comp

maintcounts_dtdf_comp4_lpdf['comp4Lastreplaceddt'] = maintcounts_dtdf_comp4_l
                                                         .fillna(method = 
maintcounts_dtdf_comp4_lpdf.head(5)
```

Out[106]:

	machineID	DT	datetime_x	comp4_x	comp4rank_x	comp4Lastreplaceddt
0	1	2014-07-16 06:00:00	7/16/2014 6:00:00 AM	1.0	1.0	2014-07-16 06:00:00
1	1	2015-01-05 06:00:00	1/5/2015 6:00:00 AM	1.0	2.0	2014-07-16 06:00:00
2	1	2015-02-04 06:00:00	2/4/2015 6:00:00 AM	1.0	3.0	2015-01-05 06:00:00
3	1	2015-06-19 06:00:00	6/19/2015 6:00:00 AM	1.0	4.0	2015-02-04 06:00:00
4	1	2015-09-02 06:00:00	9/2/2015 6:00:00 AM	1.0	5.0	2015-06-19 06:00:00

In [107]:

```python
# merging all maintenance dataframes

maint_summarydf = pd.merge(maintcounts_dtdf,maintcounts_dtdf_comp1_lpdf, on =

maint_summarydf = pd.merge(maint_summarydf,maintcounts_dtdf_comp2_lpdf, on =(

maint_summarydf = pd.merge(maint_summarydf,maintcounts_dtdf_comp3_lpdf, on =(

maint_summarydf = pd.merge(maint_summarydf,maintcounts_dtdf_comp4_lpdf, on =(

maint_summarydf.head(15)
```

Out[107]:

	machineID	DT	datetime	comp1	comp2	comp3	comp4	datetime_x_x	comp1_
0	1	2014-06-01 06:00:00	6/1/2014 6:00:00 AM	0.0	1.0	0.0	0.0	NaN	Na
1	1	2014-07-16 06:00:00	7/16/2014 6:00:00 AM	0.0	0.0	0.0	1.0	NaN	Na
2	1	2014-07-31 06:00:00	7/31/2014 6:00:00 AM	0.0	0.0	1.0	0.0	NaN	Na
3	1	2014-12-13 06:00:00	12/13/2014 6:00:00 AM	1.0	0.0	0.0	0.0	12/13/2014 6:00:00 AM	[illegible]
4	1	2015-01-01 06:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
5	1	2015-01-01 07:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
6	1	2015-01-01 08:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
7	1	2015-01-01 09:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
8	1	2015-01-01 10:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
9	1	2015-01-01 11:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
10	1	2015-01-01 12:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
11	1	2015-01-01 13:00:00	0	0.0	0.0	0.0	0.0	NaN	Na

	machineID	DT	datetime	comp1	comp2	comp3	comp4	datetime_x_x	comp1_
12	1	2015-01-01 14:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
13	1	2015-01-01 15:00:00	0	0.0	0.0	0.0	0.0	NaN	Na
14	1	2015-01-01 16:00:00	0	0.0	0.0	0.0	0.0	NaN	Na

15 rows × 23 columns

In [108]:
```
maint_summarydf.columns
```

Out[108]:
```
Index(['machineID', 'DT', 'datetime', 'comp1', 'comp2', 'comp3', 'comp4',
       'datetime_x_x', 'comp1_x', 'comp1rank_x', 'comp1Lastreplaceddt',
       'datetime_x_y', 'comp2_x', 'comp2rank_x', 'comp2Lastreplaceddt',
       'datetime_x_x', 'comp3_x', 'comp3rank_x', 'comp3Lastreplaceddt',
       'datetime_x_y', 'comp4_x', 'comp4rank_x', 'comp4Lastreplaceddt'],
      dtype='object')
```

In [109]:
```
maint_summarydf = maint_summarydf.drop([   'datetime_x_x', 'comp1_x', 'comp1r
       'datetime_x_y', 'comp2_x', 'comp2rank_x',
       'datetime_x_x', 'comp3_x', 'comp3rank_x',
       'datetime_x_y', 'comp4_x', 'comp4rank_x'], axis = 1)
```

```
In [110]: maint_summarydf = maint_summarydf[maint_summarydf['DT'] > pd.to_datetime('201
          maint_summarydf.head(15)
```

Out[110]:

	machineID	DT	datetime	comp1	comp2	comp3	comp4	comp1Lastreplaceddt	com
4	1	2015-01-01 06:00:00	0	0.0	0.0	0.0	0.0	NaT	
5	1	2015-01-01 07:00:00	0	0.0	0.0	0.0	0.0	NaT	
6	1	2015-01-01 08:00:00	0	0.0	0.0	0.0	0.0	NaT	
7	1	2015-01-01 09:00:00	0	0.0	0.0	0.0	0.0	NaT	
8	1	2015-01-01 10:00:00	0	0.0	0.0	0.0	0.0	NaT	
9	1	2015-01-01 11:00:00	0	0.0	0.0	0.0	0.0	NaT	
10	1	2015-01-01 12:00:00	0	0.0	0.0	0.0	0.0	NaT	
11	1	2015-01-01 13:00:00	0	0.0	0.0	0.0	0.0	NaT	
12	1	2015-01-01 14:00:00	0	0.0	0.0	0.0	0.0	NaT	
13	1	2015-01-01 15:00:00	0	0.0	0.0	0.0	0.0	NaT	
14	1	2015-01-01 16:00:00	0	0.0	0.0	0.0	0.0	NaT	
15	1	2015-01-01 17:00:00	0	0.0	0.0	0.0	0.0	NaT	
16	1	2015-01-01 18:00:00	0	0.0	0.0	0.0	0.0	NaT	
17	1	2015-01-01 19:00:00	0	0.0	0.0	0.0	0.0	NaT	
18	1	2015-01-01 20:00:00	0	0.0	0.0	0.0	0.0	NaT	

In [111]:
```python
maint_summarydf = pd.merge(telemetry_dts,maint_summarydf, on = ['machineID',

# forward-fill the most-recent date of component change
maint_summarydf['comp1Lastreplaceddt'] = maint_summarydf['comp1Lastreplaceddt
maint_summarydf['comp2Lastreplaceddt'] = maint_summarydf['comp2Lastreplaceddt
maint_summarydf['comp3Lastreplaceddt'] = maint_summarydf['comp3Lastreplaceddt
maint_summarydf['comp4Lastreplaceddt'] = maint_summarydf['comp4Lastreplaceddt

maint_summarydf[maint_summarydf['comp1Lastreplaceddt'] != ''].head(150)
```

Out[111]:

	machineID	DT	datetime	comp1	comp2	comp3	comp4	comp1Lastreplaceddt	co
0	1	2015-01-01 06:00:00	0	0.0	0.0	0.0	0.0	NaT	
1	1	2015-01-01 07:00:00	0	0.0	0.0	0.0	0.0	NaT	
2	1	2015-01-01 08:00:00	0	0.0	0.0	0.0	0.0	NaT	
3	1	2015-01-01 09:00:00	0	0.0	0.0	0.0	0.0	NaT	
4	1	2015-01-01 10:00:00	0	0.0	0.0	0.0	0.0	NaT	
...	...	...	...	...	...	...	...	...	
145	1	2015-01-07 07:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	
146	1	2015-01-07 08:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	
147	1	2015-01-07 09:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	
148	1	2015-01-07 10:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	
149	1	2015-01-07 11:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	

150 rows × 11 columns

back filling the first date

```
maint_summarydf['comp1Lastreplaceddt'] =
maint_summarydf['comp1Lastreplaceddt'].fillna(method='bfill')
maint_summarydf['comp2Lastreplaceddt'] =
maint_summarydf['comp2Lastreplaceddt'].fillna(method='bfill')
maint_summarydf['comp3Lastreplaceddt'] =
maint_summarydf['comp3Lastreplaceddt'].fillna(method='bfill')
maint_summarydf['comp4Lastreplaceddt'] =
maint_summarydf['comp4Lastreplaceddt'].fillna(method='bfill')
```

In [112]:

```
maint_summarydf['comp1_repgapdays'] = (maint_summarydf['DT'] - maint_summaryc

maint_summarydf['comp2_repgapdays'] = (maint_summarydf['DT'] - maint_summaryc
maint_summarydf['comp3_repgapdays'] = (maint_summarydf['DT'] - maint_summaryc
maint_summarydf['comp4_repgapdays'] = (maint_summarydf['DT'] - maint_summaryc

maint_summarydf.head(15)
```

Out[112]:

	machineID	DT	datetime	comp1	comp2	comp3	comp4	comp1Lastreplaceddt
0	1	2015-01-01 06:00:00	0	0.0	0.0	0.0	0.0	NaT
1	1	2015-01-01 07:00:00	0	0.0	0.0	0.0	0.0	NaT
2	1	2015-01-01 08:00:00	0	0.0	0.0	0.0	0.0	NaT
3	1	2015-01-01 09:00:00	0	0.0	0.0	0.0	0.0	NaT
4	1	2015-01-01 10:00:00	0	0.0	0.0	0.0	0.0	NaT
5	1	2015-01-01 11:00:00	0	0.0	0.0	0.0	0.0	NaT
6	1	2015-01-01 12:00:00	0	0.0	0.0	0.0	0.0	NaT
7	1	2015-01-01 13:00:00	0	0.0	0.0	0.0	0.0	NaT
8	1	2015-01-01 14:00:00	0	0.0	0.0	0.0	0.0	NaT
9	1	2015-01-01 15:00:00	0	0.0	0.0	0.0	0.0	NaT
10	1	2015-01-01 16:00:00	0	0.0	0.0	0.0	0.0	NaT
11	1	2015-01-01 17:00:00	0	0.0	0.0	0.0	0.0	NaT
12	1	2015-01-01 18:00:00	0	0.0	0.0	0.0	0.0	NaT
13	1	2015-01-01 19:00:00	0	0.0	0.0	0.0	0.0	NaT

	machineID	DT	datetime	comp1	comp2	comp3	comp4	comp1Lastreplaceddt
14	1	2015-01-01 20:00:00	0	0.0	0.0	0.0	0.0	NaT

In [113]:
```
maint_summarydf[maint_summarydf['comp1'] == 1.0 ].sort_values('DT').head(15)
```

Out[113]:

	machineID	DT	datetime	comp1	comp2	comp3	comp4	comp1Lastreplaceddt
289113	34	2015-01-01 06:00:00	1/1/2015 6:00:00 AM	1.0	1.0	0.0	0.0	2014-07-31 06:00:00
438050	51	2015-01-01 06:00:00	1/1/2015 6:00:00 AM	1.0	0.0	1.0	0.0	2014-09-14 06:00:00
113893	14	2015-01-01 06:00:00	1/1/2015 6:00:00 AM	1.0	0.0	0.0	0.0	2014-07-31 06:00:00
560704	65	2015-01-01 06:00:00	1/1/2015 6:00:00 AM	1.0	0.0	0.0	0.0	2014-09-14 06:00:00
630816	73	2015-01-02 06:00:00	1/2/2015 6:00:00 AM	1.0	0.0	1.0	0.0	2014-07-16 06:00:00
587011	68	2015-01-02 06:00:00	1/2/2015 6:00:00 AM	1.0	0.0	0.0	0.0	2014-09-14 06:00:00
700904	81	2015-01-02 06:00:00	1/2/2015 6:00:00 AM	1.0	0.0	1.0	0.0	2014-07-16 06:00:00
201527	24	2015-01-02 06:00:00	1/2/2015 6:00:00 AM	1.0	1.0	0.0	0.0	2014-09-14 06:00:00
762279	88	2015-01-04 06:00:00	1/4/2015 6:00:00 AM	1.0	0.0	0.0	0.0	2014-08-30 06:00:00
96	1	2015-01-05 06:00:00	1/5/2015 6:00:00 AM	1.0	0.0	0.0	1.0	2014-12-13 06:00:00
595844	69	2015-01-05 06:00:00	1/5/2015 6:00:00 AM	1.0	1.0	0.0	0.0	2014-11-28 06:00:00
552039	64	2015-01-05 06:00:00	1/5/2015 6:00:00 AM	1.0	0.0	0.0	1.0	2014-08-30 06:00:00
96515	12	2015-01-07 06:00:00	1/7/2015 6:00:00 AM	1.0	1.0	0.0	0.0	2014-11-13 06:00:00
648458	75	2015-01-07 06:00:00	1/7/2015 6:00:00 AM	1.0	0.0	0.0	0.0	2014-09-29 06:00:00
744853	86	2015-01-08 06:00:00	1/8/2015 6:00:00 AM	1.0	0.0	0.0	1.0	2014-12-28 06:00:00

In [114]:
```
maint_summarydf.columns
```

Out[114]:
```
Index(['machineID', 'DT', 'datetime', 'comp1', 'comp2', 'comp3', 'comp4',
       'comp1Lastreplaceddt', 'comp2Lastreplaceddt', 'comp3Lastreplaceddt',
       'comp4Lastreplaceddt', 'comp1_repgapdays', 'comp2_repgapdays',
       'comp3_repgapdays', 'comp4_repgapdays'],
      dtype='object')
```

In [115]:
```
maint_sum_df = maint_summarydf[['machineID', 'DT','comp1_repgapdays', 'comp2_
       'comp3_repgapdays', 'comp4_repgapdays']]
```

In [116]:
```
print(maint_sum_df.dtypes)
```

```
machineID                    int64
DT                  datetime64[ns]
comp1_repgapdays           float64
comp2_repgapdays           float64
comp3_repgapdays           float64
comp4_repgapdays           float64
dtype: object
```

Merging all datasets

In [117]:
```
Alldata = pd.DataFrame()

Alldata = pd.merge(telemetry_summarydf, error_sum_df, on = ['machineID', 'DT'

Alldata = pd.merge(Alldata, maint_sum_df, on = ['machineID', 'DT'], how = 'le
```

In [118]:

```python
# merging machine meta data

Alldata = pd.merge(Alldata, machines_df, on = ['machineID'], how = 'left')
Alldata.head(15)
```

Out[118]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean
0	1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.5921
1	1	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.3156
2	1	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.5231
3	1	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.4912
4	1	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.4246
5	1	2015-01-02 00:00:00	178.513887	7.084050	440.859804	53.461091	90.8277
6	1	2015-01-02 03:00:00	167.989524	14.910036	481.173775	33.823675	94.1035
7	1	2015-01-02 06:00:00	165.002124	5.959072	456.438211	54.613403	87.6732
8	1	2015-01-02 09:00:00	193.164359	10.459846	448.652619	46.294659	101.4389
9	1	2015-01-02 12:00:00	159.676811	11.972868	430.571937	51.632304	100.2421
10	1	2015-01-02 15:00:00	173.019460	12.377689	432.717201	17.368725	98.1268
11	1	2015-01-02 18:00:00	168.747581	14.479508	456.696379	62.025493	98.2974
12	1	2015-01-02 21:00:00	158.339642	11.343408	471.026837	40.271733	113.8168
13	1	2015-01-03 00:00:00	161.744699	21.532893	430.977304	16.196129	100.4871
14	1	2015-01-03 03:00:00	178.488928	13.001405	452.939230	44.300607	91.9648

15 rows × 29 columns

In [119]:

```
print(Alldata.dtypes)
```

```
machineID                       int64
DT                     datetime64[ns]
volt_mean_3                   float64
volt_std_3                    float64
rotate_mean_3                 float64
rotate_std_3                  float64
pressure_mean_3               float64
pressure_std_3                float64
vibration_mean_3              float64
vibration_std_3               float64
volt_mean_24                  float64
volt_std_24                   float64
rotate_mean_24                float64
rotate_std_24                 float64
pressure_mean_24              float64
pressure_std_24               float64
vibration_mean_24             float64
vibration_std_24              float64
error1_count                  float64
error2_count                  float64
error3_count                  float64
error4_count                  float64
error5_count                  float64
comp1_repgapdays              float64
comp2_repgapdays              float64
comp3_repgapdays              float64
comp4_repgapdays              float64
model                          object
age                             int64
dtype: object
```

Merging Failure data set

In [120]:

```python
# merging machine meta data

Alldata = pd.merge(Alldata, failures_df, on = ['machineID', 'DT'], how = 'lef

#Alldata.drop([], axis = 1)
Alldata.head(15)
```

Out[120]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_m
0	1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.5
1	1	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.3
2	1	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.5
3	1	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.4
4	1	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.4
5	1	2015-01-02 00:00:00	178.513887	7.084050	440.859804	53.461091	90.8
6	1	2015-01-02 03:00:00	167.989524	14.910036	481.173775	33.823675	94.1
7	1	2015-01-02 06:00:00	165.002124	5.959072	456.438211	54.613403	87.6
8	1	2015-01-02 09:00:00	193.164359	10.459846	448.652619	46.294659	101.4
9	1	2015-01-02 12:00:00	159.676811	11.972868	430.571937	51.632304	100.2
10	1	2015-01-02 15:00:00	173.019460	12.377689	432.717201	17.368725	98.1
11	1	2015-01-02 18:00:00	168.747581	14.479508	456.696379	62.025493	98.2
12	1	2015-01-02 21:00:00	158.339642	11.343408	471.026837	40.271733	113.8
13	1	2015-01-03 00:00:00	161.744699	21.532893	430.977304	16.196129	100.4

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_m
14	1	2015-01-03 03:00:00	178.488928	13.001405	452.939230	44.300607	91.9

15 rows × 31 columns

In [121]:

```
# back filling failure for last 24 hours
Alldata = Alldata.fillna(method = 'bfill', limit = 7)
Alldata = Alldata.fillna('none')
Alldata.head(5)
```

Out[121]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean_
0	1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.59212
1	1	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.31566
2	1	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.52312
3	1	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.49122
4	1	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.42469

5 rows × 31 columns

In [122]:

```python
# checking values for failure with comp2

Alldata[Alldata['failure'] == 'comp2'].head(5)
```

Out[122]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mea
864	1	2015-04-19 09:00:00	173.349101	15.556185	365.217244	39.847454	96.194
865	1	2015-04-19 12:00:00	169.871094	13.210808	409.578214	100.008800	101.059
866	1	2015-04-19 15:00:00	163.731593	12.711748	401.293490	43.833759	108.851
867	1	2015-04-19 18:00:00	188.938118	11.086738	342.800783	47.889011	97.269
868	1	2015-04-19 21:00:00	166.184120	8.689331	343.129904	116.932877	100.234

5 rows × 31 columns

In [123]:

```python
# checking values for failure with comp2

Alldata[Alldata['comp3_repgapdays'] == 'none'].head(5)


# dropping this records

Alldata.drop(Alldata[Alldata['comp3_repgapdays'] == 'none'].index, inplace =

Alldata[Alldata['comp3_repgapdays'] == 'none'].head(5)
```

Out[123]:

machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean_3	pre

0 rows × 31 columns

In [124]:

```
# checking values for failure with comp2

Alldata[Alldata['comp2_repgapdays'] == 'none'].head(5)


# dropping this records

Alldata.drop(Alldata[Alldata['comp2_repgapdays'] == 'none'].index, inplace =

Alldata[Alldata['comp2_repgapdays'] == 'none'].head(5)
```

Out[124]:

machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean_3	pre

0 rows × 31 columns

In [125]:

```
print(Alldata.dtypes)
```

```
machineID                    int64
DT                  datetime64[ns]
volt_mean_3                float64
volt_std_3                 float64
rotate_mean_3              float64
rotate_std_3               float64
pressure_mean_3            float64
pressure_std_3             float64
vibration_mean_3           float64
vibration_std_3            float64
volt_mean_24               float64
volt_std_24                float64
rotate_mean_24             float64
rotate_std_24              float64
pressure_mean_24           float64
pressure_std_24            float64
vibration_mean_24          float64
vibration_std_24           float64
error1_count               float64
error2_count               float64
error3_count               float64
error4_count               float64
error5_count               float64
comp1_repgapdays            object
comp2_repgapdays            object
comp3_repgapdays            object
comp4_repgapdays            object
model                       object
age                          int64
datetime                    object
failure                     object
dtype: object
```

In [126]:

```python
# converting object type to float
Alldata['comp1_repgapdays'] = Alldata['comp1_repgapdays'].fillna(0).astype(fl
Alldata['comp2_repgapdays'] = Alldata['comp2_repgapdays'].fillna(0).astype(fl
Alldata['comp3_repgapdays'] = Alldata['comp3_repgapdays'].fillna(0).astype(fl
Alldata['comp4_repgapdays'] = Alldata['comp4_repgapdays'].fillna(0).astype(fl

print(Alldata.dtypes)
```

```
machineID                      int64
DT                    datetime64[ns]
volt_mean_3                  float64
volt_std_3                   float64
rotate_mean_3                float64
rotate_std_3                 float64
pressure_mean_3              float64
pressure_std_3               float64
vibration_mean_3             float64
vibration_std_3              float64
volt_mean_24                 float64
volt_std_24                  float64
rotate_mean_24               float64
rotate_std_24                float64
pressure_mean_24             float64
pressure_std_24              float64
vibration_mean_24            float64
vibration_std_24             float64
error1_count                 float64
error2_count                 float64
error3_count                 float64
error4_count                 float64
error5_count                 float64
comp1_repgapdays             float64
comp2_repgapdays             float64
comp3_repgapdays             float64
comp4_repgapdays             float64
model                         object
age                            int64
datetime                      object
failure                       object
dtype: object
```

In [127]:

```
Alldata.groupby('failure').count()
```

Out[127]:

	machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_m
failure							
comp1	1528	1528	1528	1528	1528	1528	
comp2	2009	2009	2009	2009	2009	2009	
comp3	978	978	978	978	978	978	
comp4	1256	1256	1256	1256	1256	1256	
none	285506	285506	285506	285506	285506	285506	2

5 rows × 30 columns

Predictive Modelling

Generally for predictive modelling splitting the data randomly would suffifce, but for time series data, splitting data based on time is a better approach build, validate and test the model.

Splitting data into 3 samples for building and validation
pd.to_datetime('2015-07-31 01:00:00'), pd.to_datetime('2015-08-01 01:00:00')
pd.to_datetime('2015-08-31 01:00:00'), pd.to_datetime('2015-09-01 01:00:00')
pd.to_datetime('2015-09-30 01:00:00'), pd.to_datetime('2015-10-01 01:00:00')

In [128]:

```
splitlist = [[pd.to_datetime('2015-07-31 01:00:00'), pd.to_datetime('2015-08-
                [pd.to_datetime('2015-08-31 01:00:00'), pd.to_datetime('2015-
                [pd.to_datetime('2015-09-30 01:00:00'), pd.to_datetime('2015-


splitdf = pd.DataFrame(splitlist)
splitdf.columns = ['train_lastdate', 'test_firstdate']
splitdf
```

Out[128]:

	train_lastdate	test_firstdate
0	2015-07-31 01:00:00	2015-08-01 01:00:00
1	2015-08-31 01:00:00	2015-09-01 01:00:00
2	2015-09-30 01:00:00	2015-10-01 01:00:00

In [129]:

```
# remove unnecessary columns from Alldata
Alldata.columns
```

Out[129]:

```
Index(['machineID', 'DT', 'volt_mean_3', 'volt_std_3', 'rotate_mean_3',
       'rotate_std_3', 'pressure_mean_3', 'pressure_std_3', 'vibration_mean
_3',
       'vibration_std_3', 'volt_mean_24', 'volt_std_24', 'rotate_mean_24',
       'rotate_std_24', 'pressure_mean_24', 'pressure_std_24',
       'vibration_mean_24', 'vibration_std_24', 'error1_count', 'error2_cou
nt',
       'error3_count', 'error4_count', 'error5_count', 'comp1_repgapdays',
       'comp2_repgapdays', 'comp3_repgapdays', 'comp4_repgapdays', 'model',
       'age', 'datetime', 'failure'],
      dtype='object')
```

In [130]:

```
Alldata = Alldata.drop(['datetime'], axis = 1)
```

In [131]:

```
# Create random forest classifier object
randomforest = RandomForestClassifier(random_state=1,          # for consist
                                      n_estimators = 100,       # number
                                      oob_score=True,           # OOB Sco
                                      bootstrap=True,
                                      n_jobs=-1,                # for usi
                                      class_weight="balanced"   # for har
                                     )
```

Splitting data into test and train data sets based on the dates

In [132]:

```
trainsplit = Alldata[Alldata['DT'] < splitdf['train_lastdate'][2]]
    #print(split.columns)
X_train = pd.get_dummies(trainsplit.drop(['machineID', 'DT', 'failure'], axis
y_train = trainsplit['failure']

testsplit = Alldata[Alldata['DT'] > splitdf['test_firstdate'][2]]
X_test = pd.get_dummies(testsplit.drop(['machineID', 'DT', 'failure'], axis =
y_test = testsplit['failure']
```

In [133]:
```
Alldata.shape
Alldata.columns
print(Alldata.dtypes)
```

```
machineID                    int64
DT                  datetime64[ns]
volt_mean_3                float64
volt_std_3                 float64
rotate_mean_3              float64
rotate_std_3               float64
pressure_mean_3            float64
pressure_std_3             float64
vibration_mean_3           float64
vibration_std_3            float64
volt_mean_24               float64
volt_std_24                float64
rotate_mean_24             float64
rotate_std_24              float64
pressure_mean_24           float64
pressure_std_24            float64
vibration_mean_24          float64
vibration_std_24           float64
error1_count               float64
error2_count               float64
error3_count               float64
error4_count               float64
error5_count               float64
comp1_repgapdays           float64
comp2_repgapdays           float64
comp3_repgapdays           float64
comp4_repgapdays           float64
model                       object
age                          int64
failure                     object
dtype: object
```

In [134]:
```
X_train.shape
```

Out[134]:
```
(216569, 30)
```

Train and predict using the model, storing results for later

In [135]:

```python
# Train model
model = randomforest.fit(X_train, y_train)
#models.append(model)

#Predicting the target variable - class
y_predfailure = model.predict(X_test)
#y_predfailure_results.append(y_predfailure)

# Get predicted probabilities
y_prob_failure = model.predict_proba(X_test)[:,1]
#y_probfailure_results.append(y_prob_failure)
```

In [136]:

```python
# Calculate feature importances
impfeatures = model.feature_importances_
impfeatures
```

Out[136]:

```
array([0.04497048, 0.00121934, 0.02804014, 0.00131188, 0.05332834,
       0.00118739, 0.0288036 , 0.00159807, 0.09213923, 0.00487745,
       0.05200369, 0.0037463 , 0.08992738, 0.00510137, 0.07695131,
       0.00469673, 0.09917478, 0.0912101 , 0.07247695, 0.08575873,
       0.11000311, 0.00863172, 0.00563759, 0.00615114, 0.00682065,
       0.01187266, 0.0026542 , 0.00187385, 0.00472435, 0.00310749])
```

In [137]:

```
indices = np.argsort(impfeatures)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], impfeatures[indices[f]]
```

```
Feature ranking:
1. feature 20 (0.110003)
2. feature 16 (0.099175)
3. feature 8 (0.092139)
4. feature 17 (0.091210)
5. feature 12 (0.089927)
6. feature 19 (0.085759)
7. feature 14 (0.076951)
8. feature 18 (0.072477)
9. feature 4 (0.053328)
10. feature 10 (0.052004)
11. feature 0 (0.044970)
12. feature 6 (0.028804)
13. feature 2 (0.028040)
14. feature 25 (0.011873)
15. feature 21 (0.008632)
16. feature 24 (0.006821)
17. feature 23 (0.006151)
18. feature 22 (0.005638)
19. feature 13 (0.005101)
20. feature 9 (0.004877)
21. feature 28 (0.004724)
22. feature 15 (0.004697)
23. feature 11 (0.003746)
24. feature 29 (0.003107)
25. feature 26 (0.002654)
26. feature 27 (0.001874)
27. feature 7 (0.001598)
28. feature 3 (0.001312)
29. feature 1 (0.001219)
30. feature 5 (0.001187)
```

Converting feature importance metric into a data frame for easy read

In [138]:

```
impfeatdf = pd.DataFrame(X_train.columns, impfeatures).reset_index()
impfeatdf.rename(columns = {'index':'featureimportance',0: 'featurename'}, ir
impfeatdf = impfeatdf.sort_values('featureimportance',ascending=False).reset_
#impfeatdf.sort_values('impfeatures',ascending=False)
```

Plotting Cummulative Importance

In [139]:

```python
# Cumulative importances
impfeatdf = impfeatdf.drop(['index'], axis = 1)
impfeatdf['cum_imp'] = np.cumsum(impfeatdf.featureimportance)
impfeatdf
```

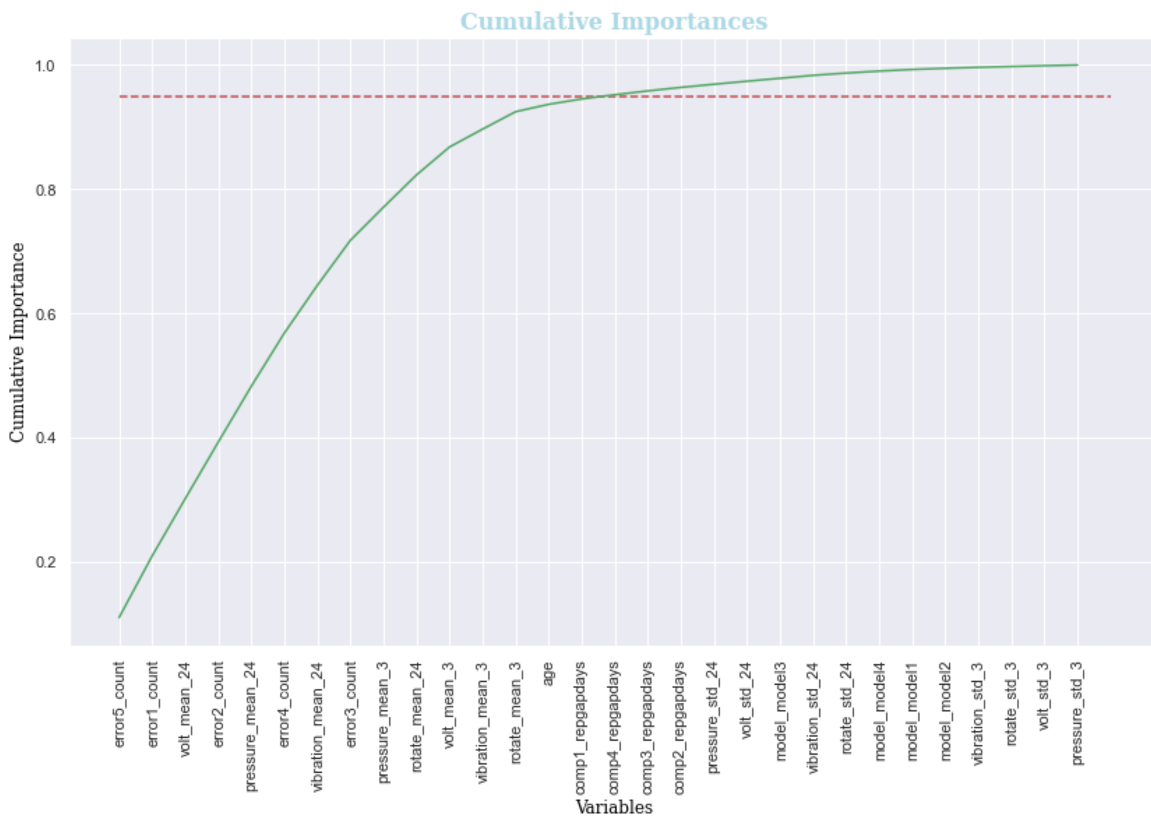
Out[139]:

	featureimportance	featurename	cum_imp
0	0.110003	error5_count	0.110003
1	0.099175	error1_count	0.209178
2	0.092139	volt_mean_24	0.301317
3	0.091210	error2_count	0.392527
4	0.089927	pressure_mean_24	0.482455
5	0.085759	error4_count	0.568213
6	0.076951	vibration_mean_24	0.645165
7	0.072477	error3_count	0.717642
8	0.053328	pressure_mean_3	0.770970
9	0.052004	rotate_mean_24	0.822974
10	0.044970	volt_mean_3	0.867944
11	0.028804	vibration_mean_3	0.896748
12	0.028040	rotate_mean_3	0.924788
13	0.011873	age	0.936660
14	0.008632	comp1_repgapdays	0.945292
15	0.006821	comp4_repgapdays	0.952113
16	0.006151	comp3_repgapdays	0.958264
17	0.005638	comp2_repgapdays	0.963902
18	0.005101	pressure_std_24	0.969003
19	0.004877	volt_std_24	0.973880
20	0.004724	model_model3	0.978605
21	0.004697	vibration_std_24	0.983301
22	0.003746	rotate_std_24	0.987048
23	0.003107	model_model4	0.990155
24	0.002654	model_model1	0.992809
25	0.001874	model_model2	0.994683
26	0.001598	vibration_std_3	0.996281
27	0.001312	rotate_std_3	0.997593
28	0.001219	volt_std_3	0.998813
29	0.001187	pressure_std_3	1.000000

In [140]:

```
xvalues = list(range(len(list(impfeatdf.featureimportance))))

plt.figure(figsize=(14, 8))

# Make a line graph
plt.plot(xvalues, impfeatdf.cum_imp, 'g-')

# Draw line at 95% of importance retained
plt.hlines(y = 0.95, xmin=0, xmax=len(impfeatdf.featureimportance), color = '

# Format x ticks and labels
plt.xticks(xvalues, impfeatdf.featurename, rotation = 'vertical')

# Axis labels and title
plt.xlabel('Variables', fontdict=labelfont)
plt.ylabel('Cumulative Importance', fontdict=labelfont)
plt.title('Cumulative Importances', fontdict=titlefont)
```

Out[140]: Text(0.5, 1.0, 'Cumulative Importances')



The following variables appear to be more important to the model as these variables could explain more than 90% of the variance in model:

error1_count
error5_count
error2_count
error4_count
volt_mean_24

pressure_mean_24
vibration_mean_24
error3_count
pressure_mean_3
rotate_mean_24
volt_mean_3
vibration_mean_3

Model Evaluation

In [141]:

```
# Get accuracy score
randomforest.score(X_test, y_test)
```

Out[141]:

```
0.9989852248741679
```

In [142]:

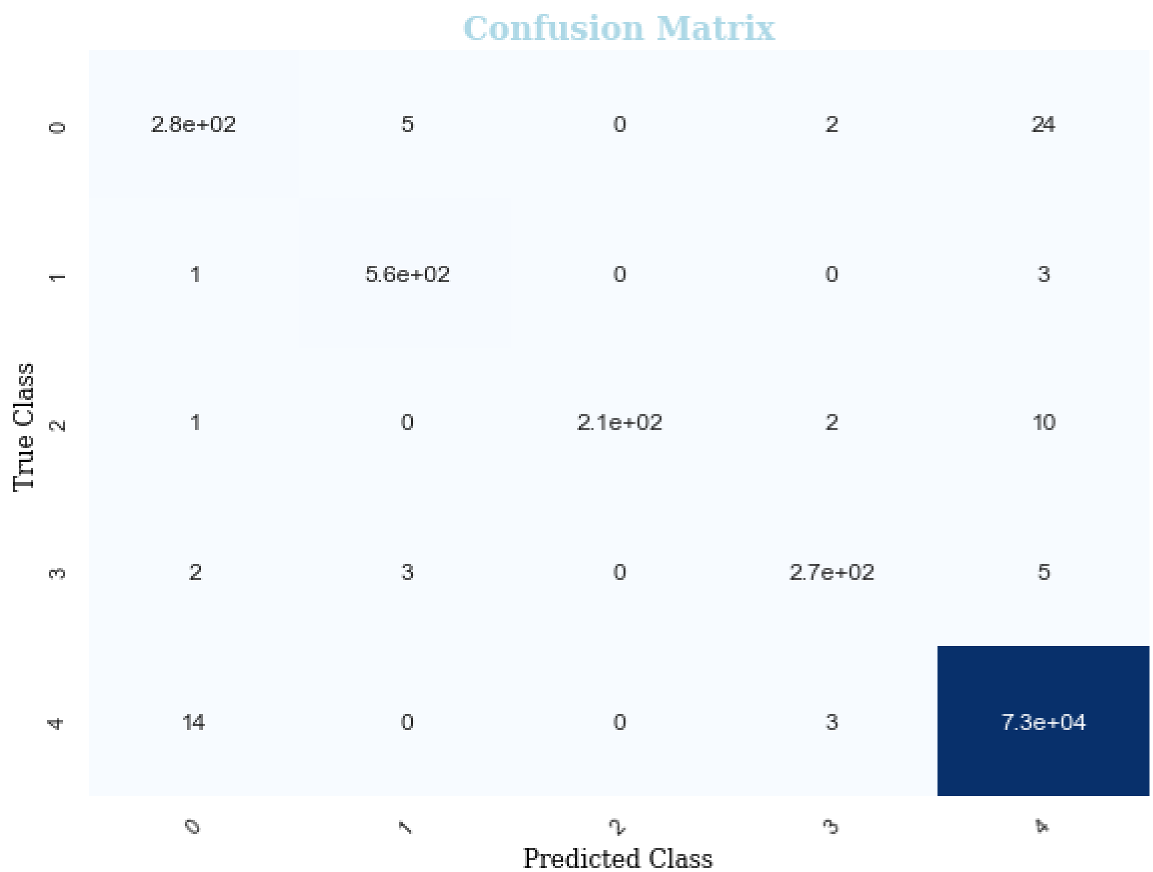
```
# Create confusion matrix
y_pred = model.predict(X_test)
matrix = confusion_matrix(y_test, y_pred)
matrix
```

Out[142]:

```
array([[  284,     5,     0,     2,    24],
       [    1,   555,     0,     0,     3],
       [    1,     0,   206,     2,    10],
       [    2,     3,     0,   273,     5],
       [   14,     0,     0,     3, 72515]], dtype=int64)
```

In [143]:

```
# Create pandas dataframe
dataframe = pd.DataFrame(matrix) #, index=class_names, columns=class_names)


# Create heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(dataframe, annot=True, cbar=None, cmap="Blues")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.xlabel("Predicted Class", fontdict=labelfont)
plt.ylabel("True Class", fontdict=labelfont)

plt.xticks(rotation=45)
plt.title("Confusion Matrix" , fontdict=titlefont)
plt.show()
```



In [144]:
```
# printing classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       comp1       0.94      0.90      0.92       315
       comp2       0.99      0.99      0.99       559
       comp3       1.00      0.94      0.97       219
       comp4       0.97      0.96      0.97       283
        none       1.00      1.00      1.00     72532

    accuracy                           1.00     73908
   macro avg       0.98      0.96      0.97     73908
weighted avg       1.00      1.00      1.00     73908
```

In [145]:
```
def fn_multiclass_metrics(actual_label, predicted_label):
    """
    function that takes acutal labels and predicted labels and returns
    accuracy, auc, precision, recall and f1 scores
    average = 'weighted' for multi class classification
    """
    accuracy = accuracy_score(actual_label, predicted_label)
    precision = precision_score(actual_label, predicted_label, average = 'wei
    recall = recall_score(actual_label, predicted_label, average = 'weighted'
    f1 = f1_score(actual_label, predicted_label, average = 'weighted')

    return (accuracy, precision, recall, f1)
```

In [146]:
```
acc, prec, recall, f1 = fn_multiclass_metrics(y_test, y_pred)

acc, prec, recall, f1
```

Out[146]:
```
(0.9989852248741679,
 0.9989746911742967,
 0.9989852248741679,
 0.9989752956297505)
```

In preventive maintenance prediction, the most important metric to evaluate the model is recall, which conveys the actual number of failures predicted by the model. Here in the model built. it is around 99.8%. I suspect this could be due to large portion of failure = 'none'. I am sure, model could be further tweaked to nullify this bias with the help of domain experts.

Deployment

Create the model with best parameters obtained from tuning.

Save the model using joblib module as a pickle.

Deploy the pickle on the server and use it for fitting new unseen data.

In [147]:
```python
cwd = os.getcwd()
print(cwd)

projdir = os.path.dirname(cwd)
modeldir = os.path.join(projdir, 'Model')

# importing telemetry data

modelfile = os.path.join(modeldir, 'predictivemodel.pkl')
```

```
C:\Users\14802\OneDrive\Desktop\DSC 680-PROJECTS\Projects\Week1\Code
```

In [148]:
```python
# Save the model as a pickle in a file
joblib.dump(model, modelfile)
```

Out[148]:
```
['C:\\Users\\14802\\OneDrive\\Desktop\\DSC 680-PROJECTS\\Projects\\Week1\\M
odel\\predictivemodel.pkl']
```

In [149]:
```python
# Load the model from the file
tunedmodel_from_joblib = joblib.load(modelfile)
```

In [150]:
```python
# Fitting deployed model on new data ( assume here X_train and y_train are ne
deployed_model = tunedmodel_from_joblib.fit(X_train, y_train)
```

Conslusion

The accuracy of the model appears to be around 99 percent, which is incredible. I am confident that the model can be improved further by reducing bias and other factors. Another way to improve is to develop and train the model using the essential feature variables listed above.Building a predictive model for preventive maintenance, like any other predictive modeling, necessitates a great deal of domain knowledge and the creation of several feature variables. In this model, I used rolling mean for the last 24 hours and last 3 hours for each 3 hour window to produce telemetry feature variables. We may need to explore longer windows for these rolling computations at times. However, feature engineering is a large task, and data scientists in this preventative maintenance use case will need some help from domain experts.