In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as plt
%matplotlib inline
```

In [2]:

```python
df = pd.read_csv('Maintenance.csv')
df.head()
```

Out[2]:

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

In [3]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   UDI                      10000 non-null  int64  
 1   Product ID               10000 non-null  object 
 2   Type                     10000 non-null  object 
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64  
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64  
 8   Machine failure          10000 non-null  int64  
 9   TWF                      10000 non-null  int64  
 10  HDF                      10000 non-null  int64  
 11  PWF                      10000 non-null  int64  
 12  OSF                      10000 non-null  int64  
 13  RNF                      10000 non-null  int64  
dtypes: float64(3), int64(9), object(2)
memory usage: 1.1+ MB
```

In [4]:

```
duplicated = df[df.duplicated()]
duplicated
```

Out[4]:

UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDI

In [5]:

```
df
```

Out[5]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TW
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	
...	...	...	...	...	...	...	...	...	...	.
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0	
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0	
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0	
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0	
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0	

10000 rows × 14 columns

In [6]:

```
df = df.drop(['UDI', 'Product ID'], axis=1)
```

In [7]:

```
df=df.rename({'Air temperature [K]':'Air_temp','Process temperature [K]':'Process_temp','Ro
df.head()
```

Out[7]:

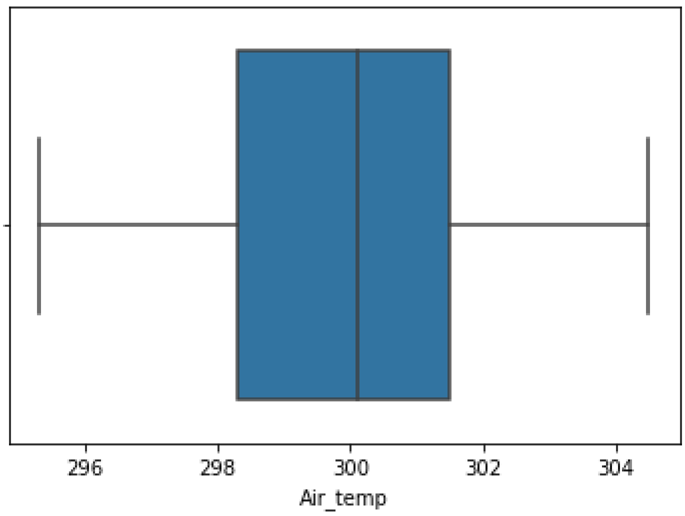
	Type	Air_temp	Process_temp	Rot_speed	Torque	Tool_wear	Mach_failure	TWF	HDF	PV
0	M	298.1	308.6	1551	42.8	0	0	0	0	
1	L	298.2	308.7	1408	46.3	3	0	0	0	
2	L	298.1	308.5	1498	49.4	5	0	0	0	
3	L	298.2	308.6	1433	39.5	7	0	0	0	
4	L	298.2	308.7	1408	40.0	9	0	0	0	

In [8]:

```
sns.boxplot(x=df["Air_temp"])
```

Out[8]:

<AxesSubplot:xlabel='Air_temp'>

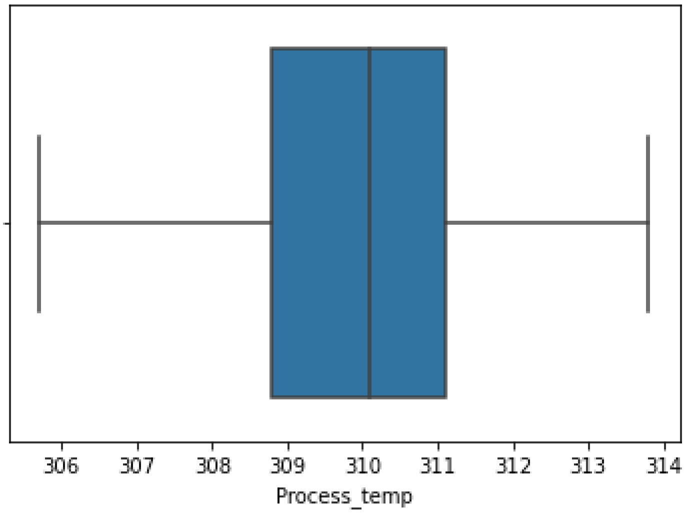


In [9]:

```
sns.boxplot(x=df["Process_temp"])
```

Out[9]:

<AxesSubplot:xlabel='Process_temp'>

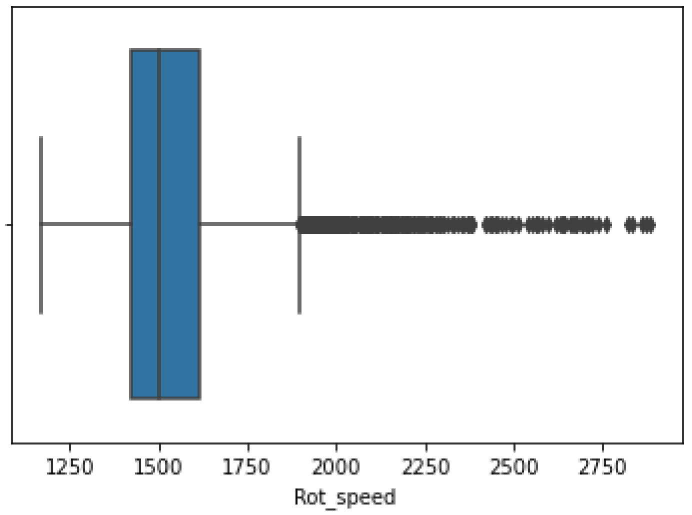


In [10]:

```
sns.boxplot(x=df["Rot_speed"])
```

Out[10]:

<AxesSubplot:xlabel='Rot_speed'>

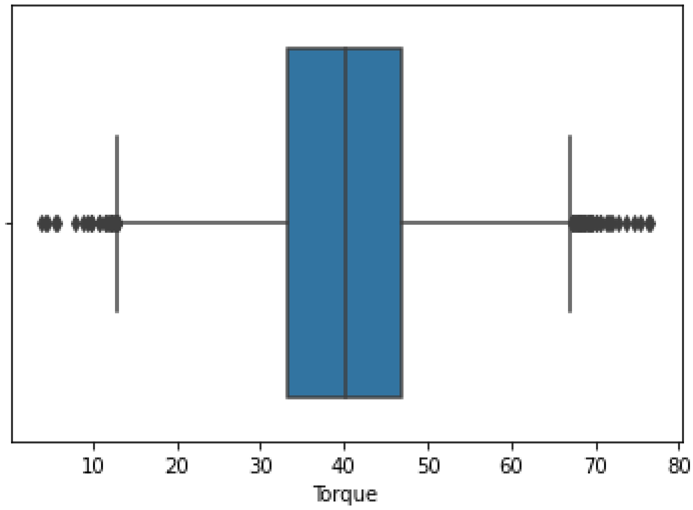


In [11]:

```
sns.boxplot(x=df["Torque"])
```

Out[11]:

<AxesSubplot:xlabel='Torque'>

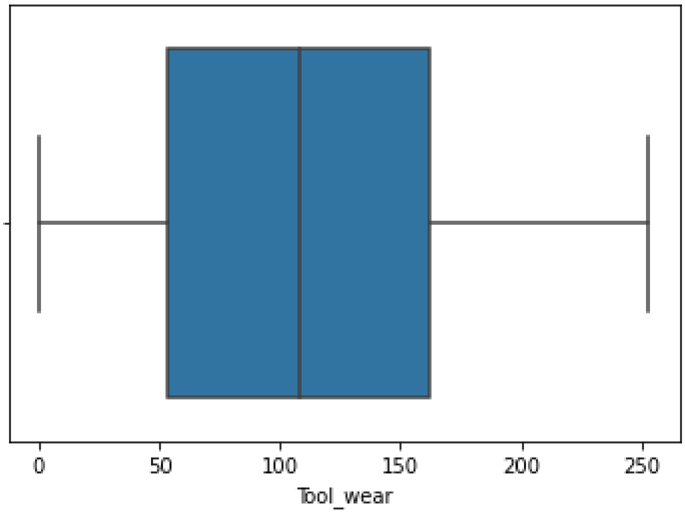


In [12]:

```
sns.boxplot(x=df["Tool_wear"])
```

Out[12]:

```
<AxesSubplot:xlabel='Tool_wear'>
```

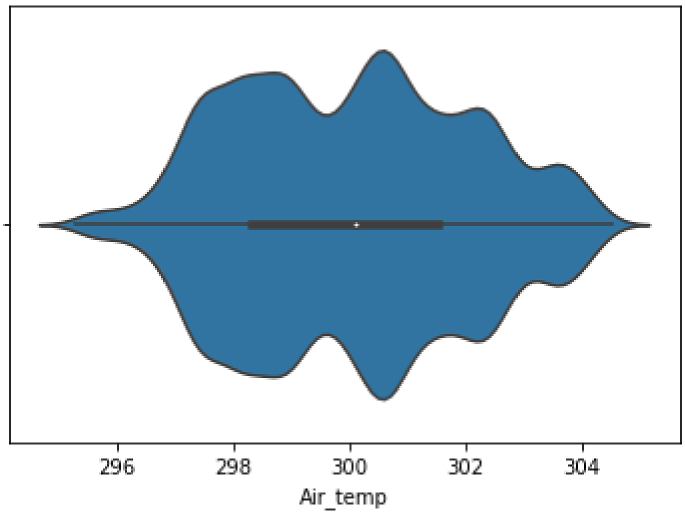


In [13]:

```
sns.violinplot(x=df["Air_temp"])
```

Out[13]:

```
<AxesSubplot:xlabel='Air_temp'>
```

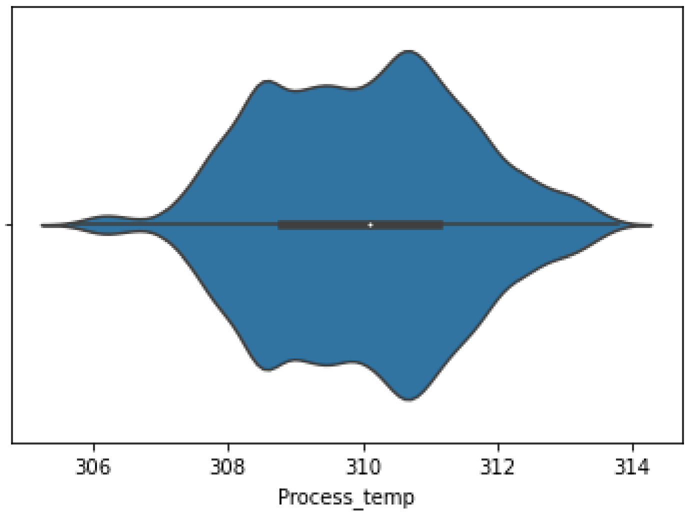


In [14]:

```
sns.violinplot(x=df["Process_temp"])
```

Out[14]:

<AxesSubplot:xlabel='Process_temp'>

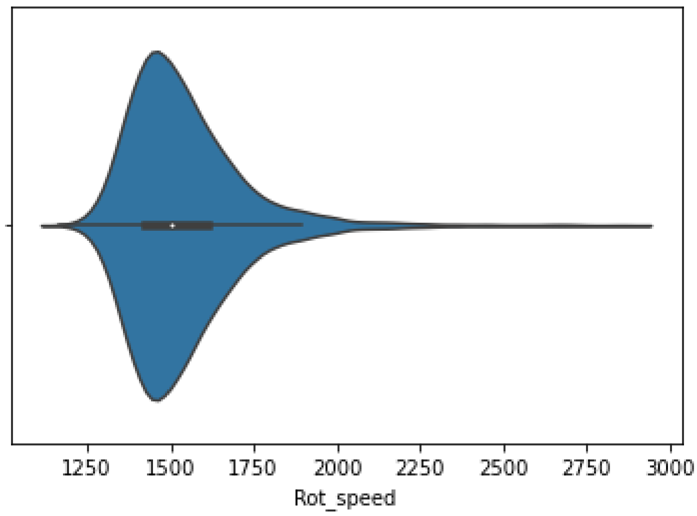


In [15]:

```
sns.violinplot(x=df["Rot_speed"])
```

Out[15]:

<AxesSubplot:xlabel='Rot_speed'>

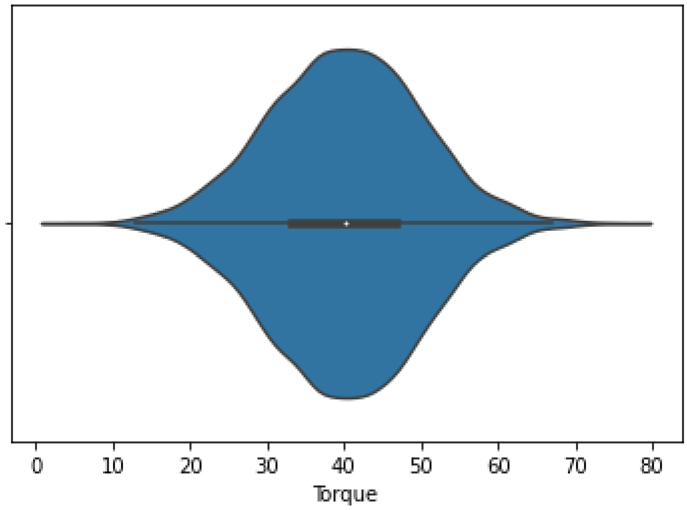


In [16]:

```
sns.violinplot(x=df["Torque"])
```

Out[16]:

<AxesSubplot:xlabel='Torque'>

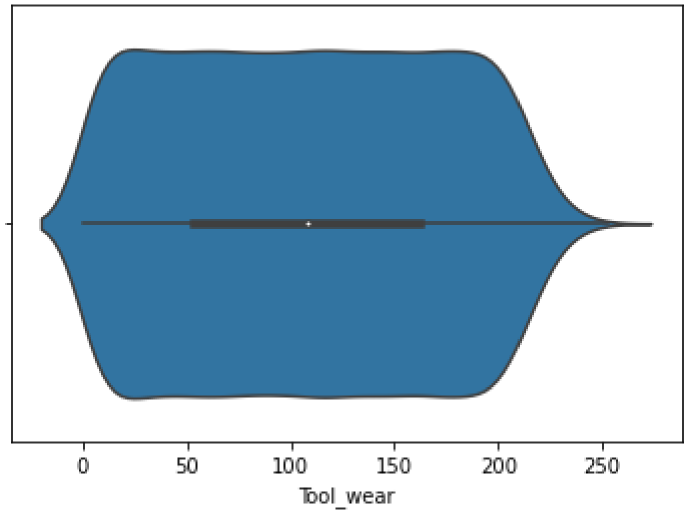


In [17]:

```
sns.violinplot(x=df["Tool_wear"])
```

Out[17]:

<AxesSubplot:xlabel='Tool_wear'>



In [18]:

```
df["Air_temp"].describe()
```

Out[18]:

```
count    10000.000000
mean       300.004930
std          2.000259
min        295.300000
25%        298.300000
50%        300.100000
75%        301.500000
max        304.500000
Name: Air_temp, dtype: float64
```

In [19]:

```
df["Process_temp"].describe()
```

Out[19]:

```
count    10000.000000
mean       310.005560
std          1.483734
min        305.700000
25%        308.800000
50%        310.100000
75%        311.100000
max        313.800000
Name: Process_temp, dtype: float64
```

In [20]:

```
df["Process_temp"].describe()
```

Out[20]:

```
count    10000.000000
mean       310.005560
std          1.483734
min        305.700000
25%        308.800000
50%        310.100000
75%        311.100000
max        313.800000
Name: Process_temp, dtype: float64
```

In [21]:

```
df["Rot_speed"].describe()
```

Out[21]:

```
count    10000.000000
mean      1538.776100
std        179.284096
min       1168.000000
25%       1423.000000
50%       1503.000000
75%       1612.000000
max       2886.000000
Name: Rot_speed, dtype: float64
```

In [22]:

```
df["Torque"].describe()
```

Out[22]:

```
count    10000.000000
mean        39.986910
std          9.968934
min          3.800000
25%         33.200000
50%         40.100000
75%         46.800000
max         76.600000
Name: Torque, dtype: float64
```

In [23]:

```
df["Tool_wear"].describe()
```

Out[23]:

```
count    10000.000000
mean       107.951000
std         63.654147
min          0.000000
25%         53.000000
50%        108.000000
75%        162.000000
max        253.000000
Name: Tool_wear, dtype: float64
```

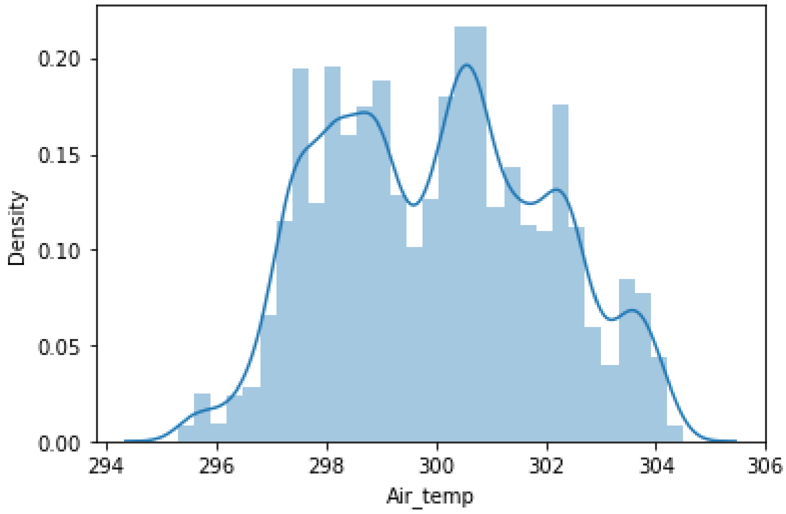
In [24]:

```
sns.distplot(df["Air_temp"])
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture 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[24]:

```
<AxesSubplot:xlabel='Air_temp', ylabel='Density'>
```



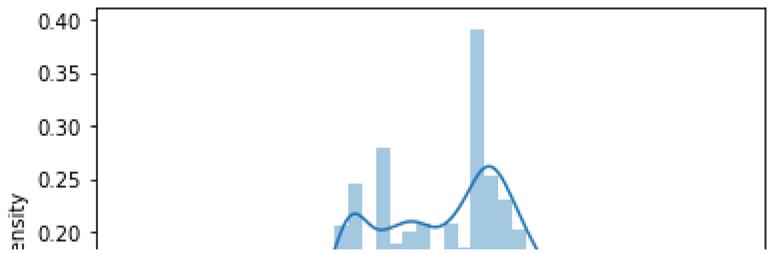
In [25]:

```
sns.distplot(df["Process_temp"])
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[25]:

```
<AxesSubplot:xlabel='Process_temp', ylabel='Density'>
```



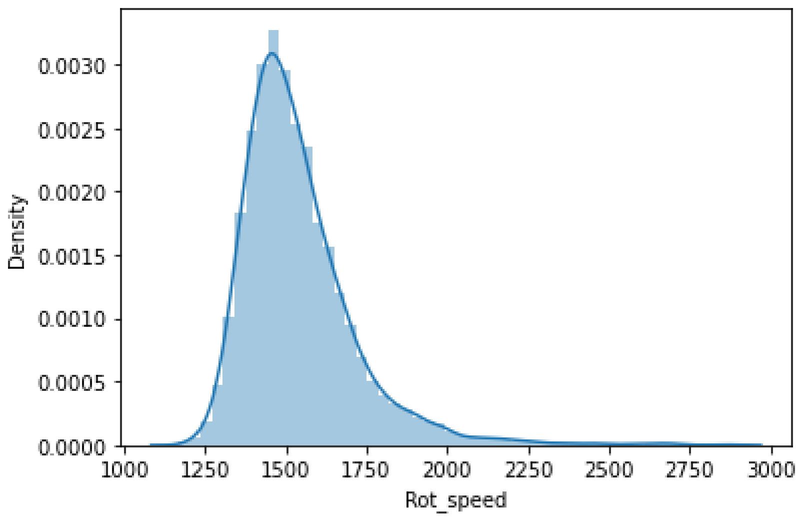
In [26]:

```
sns.distplot(df["Rot_speed"])
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture 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[26]:

```
<AxesSubplot:xlabel='Rot_speed', ylabel='Density'>
```



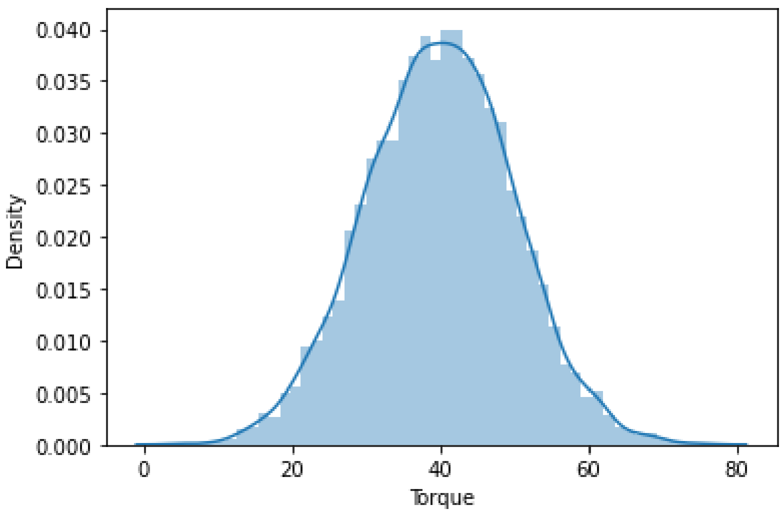
In [27]:

```
sns.distplot(df["Torque"])
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture 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[27]:

```
<AxesSubplot:xlabel='Torque', ylabel='Density'>
```



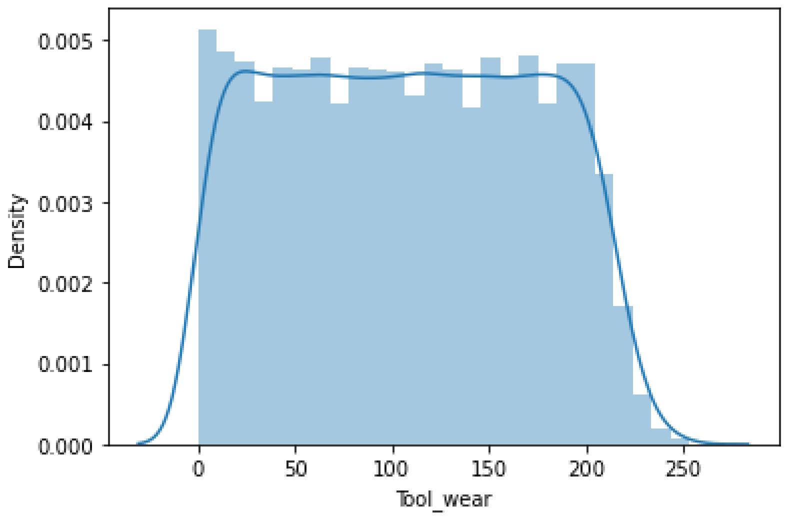
In [28]:

```
sns.distplot(df["Tool_wear"])
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture 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[28]:

```
<AxesSubplot:xlabel='Tool_wear', ylabel='Density'>
```



In [29]:

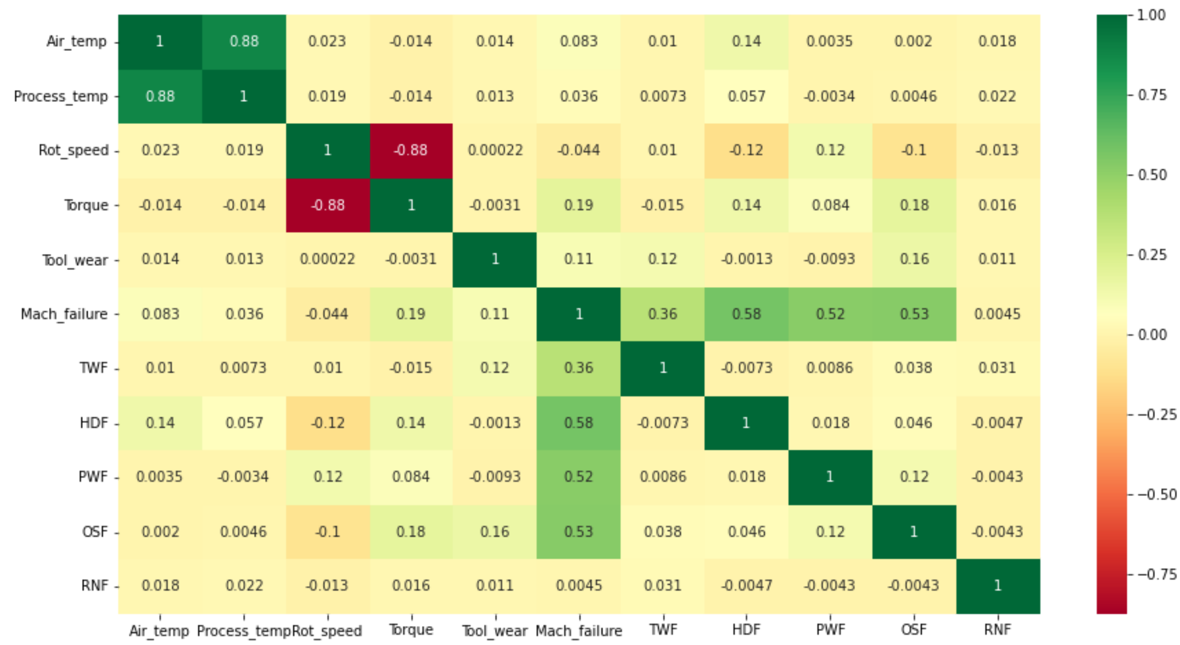
```
df.corr()
```

Out[29]:

	Air_temp	Process_temp	Rot_speed	Torque	Tool_wear	Mach_failure	T
Air_temp	1.000000	0.876107	0.022670	-0.013778	0.013853	0.082556	0.009
Process_temp	0.876107	1.000000	0.019277	-0.014061	0.013488	0.035946	0.007
Rot_speed	0.022670	0.019277	1.000000	-0.875027	0.000223	-0.044188	0.010
Torque	-0.013778	-0.014061	-0.875027	1.000000	-0.003093	0.191321	-0.014
Tool_wear	0.013853	0.013488	0.000223	-0.003093	1.000000	0.105448	0.115
Mach_failure	0.082556	0.035946	-0.044188	0.191321	0.105448	1.000000	0.362
TWF	0.009955	0.007315	0.010389	-0.014662	0.115792	0.362904	1.000
HDF	0.137831	0.056933	-0.121241	0.142610	-0.001287	0.575800	-0.007
PWF	0.003470	-0.003355	0.123018	0.083781	-0.009334	0.522812	0.008
OSF	0.001988	0.004554	-0.104575	0.183465	0.155894	0.531083	0.038
RNF	0.017688	0.022279	-0.013088	0.016136	0.011326	0.004516	0.030

In [30]:

```
import matplotlib.pyplot as plt
corr= df.corr()
plt.figure(figsize=(15,8))
sns.heatmap(corr,annot=True,cmap='RdYlGn')
plt.show()
```

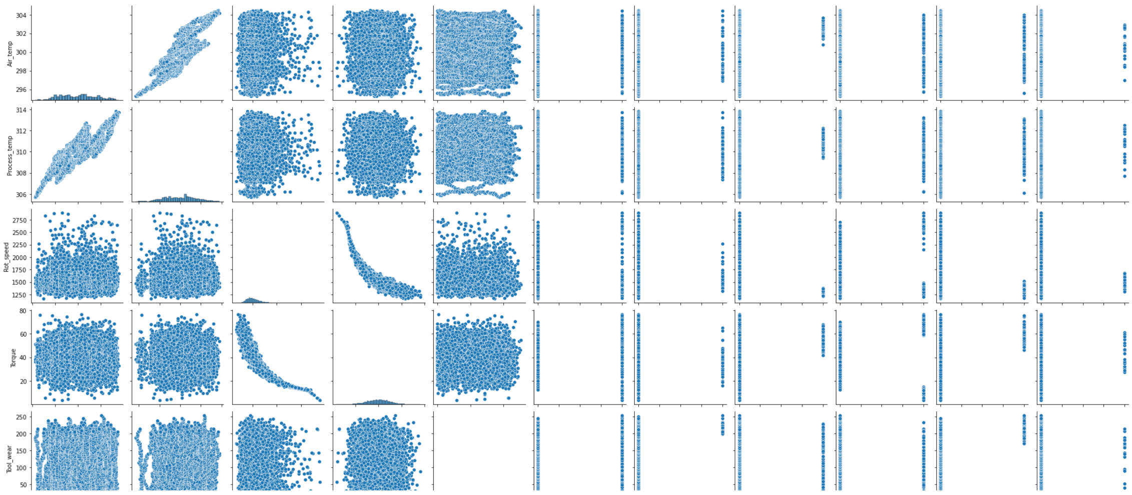


In [31]:

```
sns.pairplot(df)
```

Out[31]:

<seaborn.axisgrid.PairGrid at 0x1e2ecb2ba60>



In [32]:

```
df
```

Out[32]:

	Type	Air_temp	Process_temp	Rot_speed	Torque	Tool_wear	Mach_failure	TWF	HDF
0	M	298.1	308.6	1551	42.8	0	0	0	0
1	L	298.2	308.7	1408	46.3	3	0	0	0
2	L	298.1	308.5	1498	49.4	5	0	0	0
3	L	298.2	308.6	1433	39.5	7	0	0	0
4	L	298.2	308.7	1408	40.0	9	0	0	0
...	...	...	...	...	...	...	...	...	...
9995	M	298.8	308.4	1604	29.5	14	0	0	0
9996	H	298.9	308.4	1632	31.8	17	0	0	0
9997	M	299.0	308.6	1645	33.4	22	0	0	0
9998	H	299.0	308.7	1408	48.5	25	0	0	0
9999	M	299.0	308.7	1500	40.2	30	0	0	0

10000 rows × 12 columns

In [33]:

```
df["Type"].value_counts()
```

Out[33]:

```
L    6000
M    2997
H    1003
Name: Type, dtype: int64
```

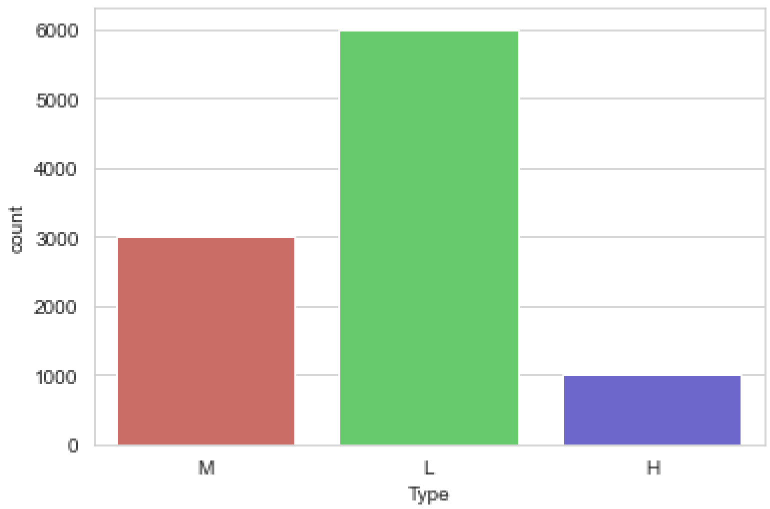
In [34]:

```
sns.set_style("whitegrid")
sns.countplot(df["Type"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[34]:

```
<AxesSubplot:xlabel='Type', ylabel='count'>
```



In [35]:

```
df["Mach_failure"].value_counts()
```

Out[35]:

```
0    9661
1     339
Name: Mach_failure, dtype: int64
```

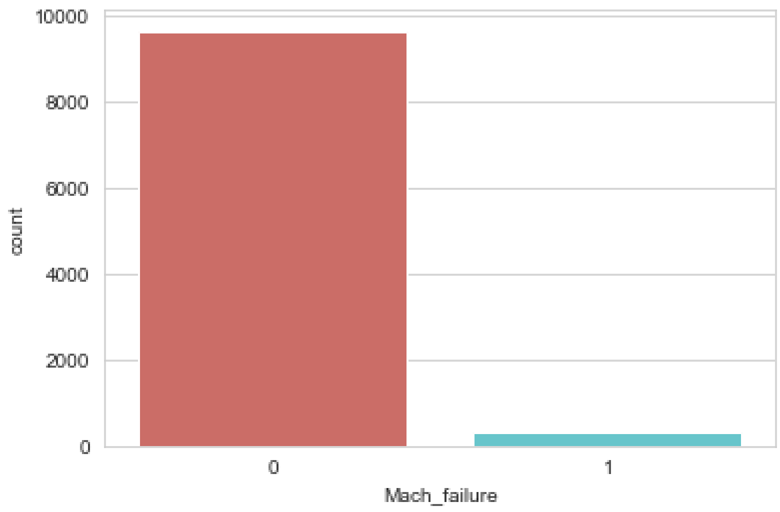
In [36]:

```
sns.set_style("whitegrid")
sns.countplot(df["Mach_failure"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[36]:

```
<AxesSubplot:xlabel='Mach_failure', ylabel='count'>
```



In [37]:

```
df["TWF"].value_counts()
```

Out[37]:

```
0    9954
1      46
Name: TWF, dtype: int64
```

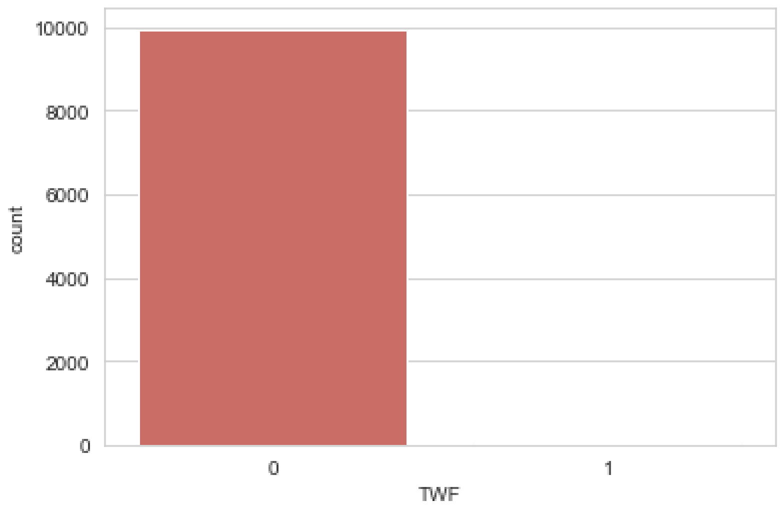
In [38]:

```
sns.set_style("whitegrid")
sns.countplot(df["TWF"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[38]:

```
<AxesSubplot:xlabel='TWF', ylabel='count'>
```



In [39]:

```
df["HDF"].value_counts()
```

Out[39]:

```
0    9885
1     115
Name: HDF, dtype: int64
```

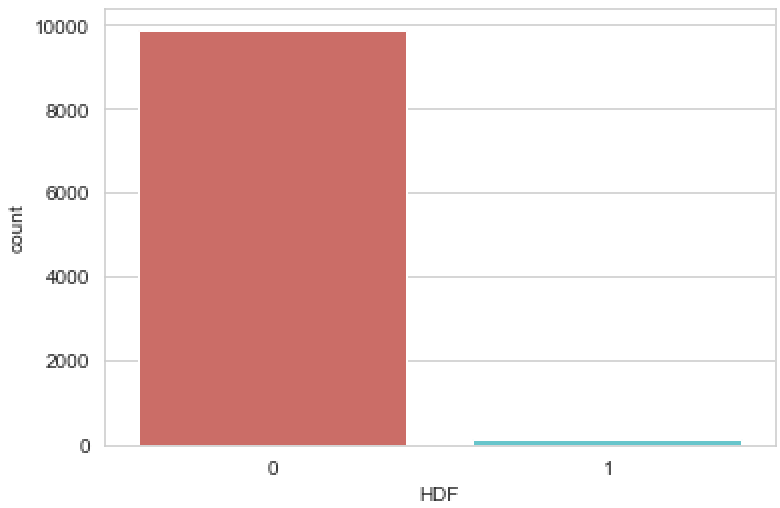
In [40]:

```
sns.set_style("whitegrid")
sns.countplot(df["HDF"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[40]:

```
<AxesSubplot:xlabel='HDF', ylabel='count'>
```



In [41]:

```
df["PWF"].value_counts()
```

Out[41]:

```
0    9905
1      95
Name: PWF, dtype: int64
```

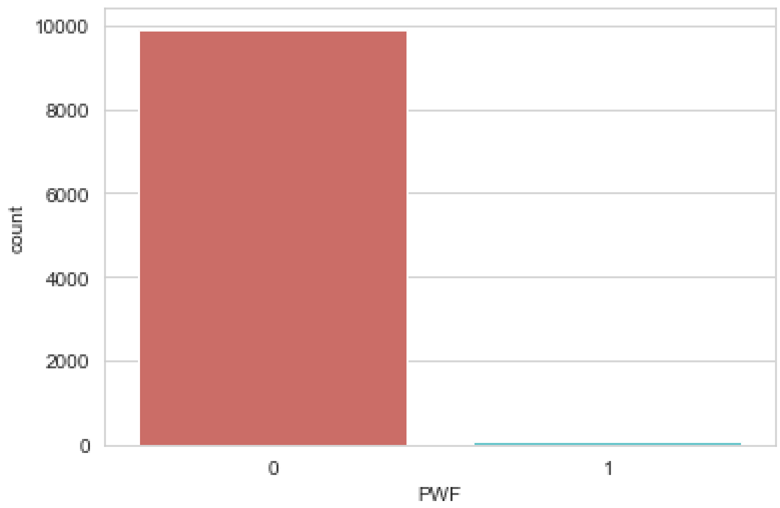
In [42]:

```
sns.set_style("whitegrid")
sns.countplot(df["PWF"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[42]:

```
<AxesSubplot:xlabel='PWF', ylabel='count'>
```



In [43]:

```
df["OSF"].value_counts()
```

Out[43]:

```
0    9902
1      98
Name: OSF, dtype: int64
```

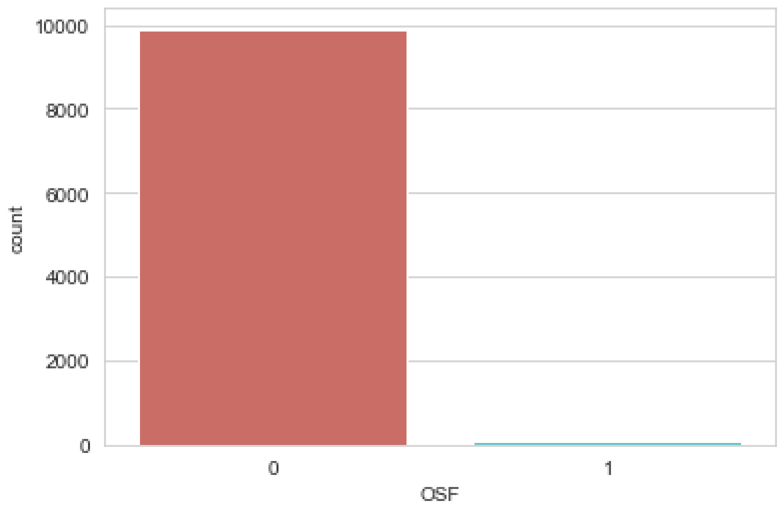
In [44]:

```
sns.set_style("whitegrid")
sns.countplot(df["OSF"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[44]:

```
<AxesSubplot:xlabel='OSF', ylabel='count'>
```



In [45]:

```
df["RNF"].value_counts()
```

Out[45]:

```
0    9981
1      19
Name: RNF, dtype: int64
```

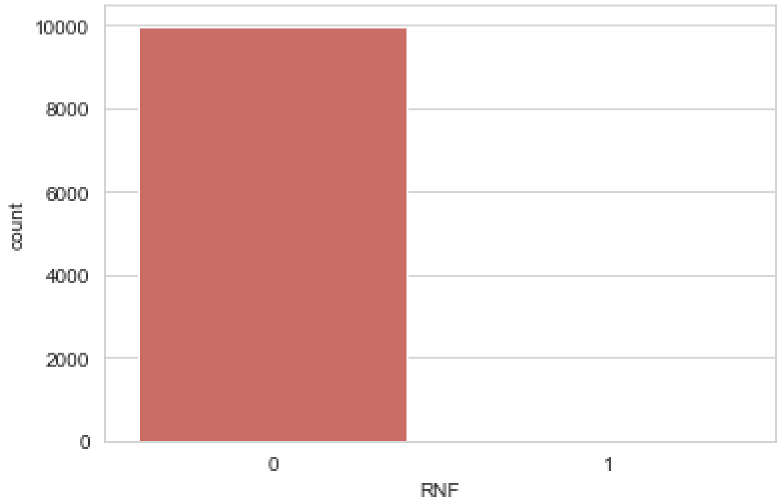
In [46]:

```
sns.set_style("whitegrid")
sns.countplot(df["RNF"], palette = "hls")
```

```
C:\Users\cricl\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[46]:

```
<AxesSubplot:xlabel='RNF', ylabel='count'>
```



In [47]:

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Type']= label_encoder.fit_transform(df['Type'])

df['Type'].unique()
```

Out[47]:

```
array([2, 1, 0])
```

In [48]:

```
X = df.drop(['Mach_failure'], axis=1)
X
```

	Type	Air_temp	Process_temp	Rot_speed	Torque	Tool_wear	TWF	HDF	PWF	OSF	RNF
0	2	298.1	308.6	1551	42.8	0	0	0	0	0	0
1	1	298.2	308.7	1408	46.3	3	0	0	0	0	0
2	1	298.1	308.5	1498	49.4	5	0	0	0	0	0
3	1	298.2	308.6	1433	39.5	7	0	0	0	0	0
4	1	298.2	308.7	1408	40.0	9	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
9995	2	298.8	308.4	1604	29.5	14	0	0	0	0	0
9996	0	298.9	308.4	1632	31.8	17	0	0	0	0	0
9997	2	299.0	308.6	1645	33.4	22	0	0	0	0	0
9998	0	299.0	308.7	1408	48.5	25	0	0	0	0	0
9999	2	299.0	308.7	1500	40.2	30	0	0	0	0	0

In [49]:

```
Y=df["Mach_failure"]
Y
```

Out[49]:

```
0       0
1       0
2       0
3       0
4       0
       ..
9995    0
9996    0
9997    0
9998    0
9999    0
Name: Mach_failure, Length: 10000, dtype: int64
```

In [50]:

```
from sklearn.preprocessing import StandardScaler
data = StandardScaler()
X = data.fit_transform(X)
```

In [51]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.3, random_state=100)
```

In [52]:

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(k_neighbors=1)
X_res, Y_res = sm.fit_resample(X_train,Y_train)
print("After Oversampling the shape of X_train:{}".format(X_res.shape))
print("After Oversampling the shape of y_train: {} \n".format(Y_res.shape))
```

```
After Oversampling the shape of X_train:(13526, 11)
After Oversampling the shape of y_train: (13526,)
```

In [53]:

```
ydf=pd.DataFrame(Y_res)
ydf.value_counts()
```

Out[53]:

```
Mach_failure
0               6763
1               6763
dtype: int64
```

In [54]:

```
#!pip install sweetviz
```

In [55]:

```
### importing required libraries
import pandas as pd
import sweetviz
my_report = sweetviz.analyze(df,target_feat='Mach_failure')
```

Done! Use 'show' commands to display/save. [100%] 00:00 -> (00:00 left)

In [56]:

```
### create a whole report in form of HTML file
my_report.show_html('Sweetviz_Report.html')
```

```
Report Sweetviz_Report.html was generated! NOTEBOOK/COLAB USERS: the web bro
wser MAY not pop up, regardless, the report IS saved in your notebook/colab
files.
```

In [57]:

```
from sklearn.model_selection import train_test_split
train,test = train_test_split(df, test_size=0.3, random_state=100)
```

In [58]:

```
### in case of testing data pass seperate list for testing data as parameter
my_report1 = sweetviz.compare([train, "Train"], [test, "Test"], "Mach_failure")
my_report1.show_html('Train_test_comparision.html')
```

Done! Use 'show' commands to display/save. [100%] 00:00 -> (00:00 left)

```
Report Train_test_comparision.html was generated! NOTEBOOK/COLAB USERS: the
web browser MAY not pop up, regardless, the report IS saved in your noteboo
k/colab files.
```

In [59]:

```
#!pip install pandas-profiling
```

In [60]:

```
from pandas_profiling import ProfileReport

profile = ProfileReport(df, title="Pandas Profiling Report", explorative=True)
profile.to_file("Pandas_Profiling_Report.html")
```

Summarize dataset: 100% 50/50 [00:08<00:00, 8.91it/s, Completed]

Generate report structure: 100% 1/1 [00:02<00:00, 3.00s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.05s/it]

Export report to file: 100% 1/1 [00:00<00:00, 30.68it/s]

MODEL BUILDING

LogisticRegression

In [61]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
```

In [62]:

```
model_log_reg = LogisticRegression(max_iter=1000)
model_log_reg.fit(X_res,Y_res)
```

Out[62]:

```
LogisticRegression(max_iter=1000)
```

In [63]:

```
# make predictions for test data
log_reg_y_pred = model_log_reg.predict(X_test)

# evaluate predictions
acc_log_reg = accuracy_score(Y_test, log_reg_y_pred)
print("Accuracy: %.2f%%" % (acc_log_reg * 100.0))
```

```
Accuracy: 99.87%
```

In [64]:

```
print(classification_report(Y_test,log_reg_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2898
           1       1.00      0.96      0.98       102

    accuracy                           1.00      3000
   macro avg       1.00      0.98      0.99      3000
weighted avg       1.00      1.00      1.00      3000
```

In [65]:

```
R2_log_reg = r2_score(Y_test, log_reg_y_pred)
print(R2_log_reg)
```

```
0.9594040514756628
```

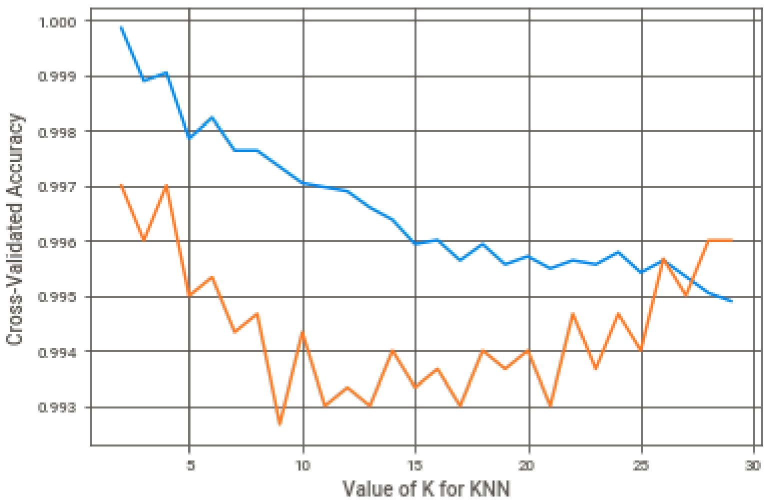
KNN

In [66]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

In [67]:

```
import matplotlib.pyplot as plt
%matplotlib inline
# choose k between 1 to 41
k_range = range(2,30)
k_scores_train = []
k_scores_test = []
# use iteration to caclulator different k in models, then return the average accuracy based
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_res,Y_res)
    pred= knn.predict(X_res)
    score = np.mean(Y_res == pred)
    k_scores_train.append(score)
    pred_test = knn.predict(X_test)
    score_test = np.mean(Y_test == pred_test)
    k_scores_test.append(score_test)
# plot to see clearly
plt.plot(k_range, k_scores_train)
plt.plot(k_range, k_scores_test)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.grid()
plt.show()
```



In [68]:

```
KNN_model = KNeighborsClassifier(n_neighbors=3)
KNN_model.fit(X_res,Y_res)
```

Out[68]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [69]:

```
# make predictions for test data
KNN_y_pred = KNN_model.predict(X_test)

# evaluate predictions
acc_KNN = accuracy_score(Y_test, KNN_y_pred)
print("Accuracy: %.2f%%" % (acc_KNN * 100.0))
```

```
Accuracy: 99.60%
```

In [70]:

```
print(classification_report(Y_test,KNN_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2898
           1       0.93      0.95      0.94       102

    accuracy                           1.00      3000
   macro avg       0.97      0.97      0.97      3000
weighted avg       1.00      1.00      1.00      3000
```

In [71]:

```
R2_KNN = r2_score(Y_test, KNN_y_pred)
print(R2_KNN)
```

```
0.8782121544269882
```

RANDOM FOREST

In [72]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [73]:

```
RF_model = RandomForestClassifier()
RF_model.fit(X_res, Y_res)
```

Out[73]:

```
RandomForestClassifier()
```

In [74]:

```
# make predictions for test data
RF_y_pred = RF_model.predict(X_test)

# evaluate predictions
acc_RF = accuracy_score(Y_test, RF_y_pred)
print("Accuracy: %.2f%%" % (acc_RF * 100.0))
```

```
Accuracy: 99.87%
```

In [75]:

```
print(classification_report(Y_test,RF_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2898
           1       1.00      0.96      0.98       102

    accuracy                           1.00      3000
   macro avg       1.00      0.98      0.99      3000
weighted avg       1.00      1.00      1.00      3000
```

In [76]:

```
R2_RF = r2_score(Y_test, RF_y_pred)
print(R2_RF)
```

```
0.9594040514756628
```

AdaBoost Classification

In [77]:

```
from sklearn.ensemble import AdaBoostClassifier
```

In [78]:

```
Ada_model = AdaBoostClassifier()
Ada_model.fit(X_res, Y_res)
```

Out[78]:

```
AdaBoostClassifier()
```

In [79]:

```
# make predictions for test data
Ada_y_pred = Ada_model.predict(X_test)

# evaluate predictions
acc_Ada = accuracy_score(Y_test, Ada_y_pred)
print("Accuracy: %.2f%%" % (acc_Ada * 100.0))
```

```
Accuracy: 99.63%
```

In [80]:

```
print(classification_report(Y_test,Ada_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2898
           1       0.93      0.96      0.95       102

    accuracy                           1.00      3000
   macro avg       0.97      0.98      0.97      3000
weighted avg       1.00      1.00      1.00      3000
```

In [81]:

```
R2_Ada = r2_score(Y_test, Ada_y_pred)
print(R2_Ada)
```

```
0.8883611415580726
```

SUPPORT VECTOR MACHINE

In [82]:

```
from sklearn import svm
from sklearn.svm import SVC
```

In [83]:

```
SVC_model = SVC()
SVC_model.fit(X_res,Y_res)
```

Out[83]:

```
SVC()
```

In [84]:

```
# make predictions for test data
SVC_y_pred = SVC_model.predict(X_test)

# evaluate predictions
acc_SVC= accuracy_score(Y_test, SVC_y_pred)
print("Accuracy: %.2f%%" % (acc_SVC * 100.0))
```

```
Accuracy: 99.87%
```

In [85]:

```
print(classification_report(Y_test,SVC_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2898
           1       1.00      0.96      0.98       102

    accuracy                           1.00      3000
   macro avg       1.00      0.98      0.99      3000
weighted avg       1.00      1.00      1.00      3000
```

In [86]:

```
R2_SVC = r2_score(Y_test, SVC_y_pred)
print(R2_SVC)
```

```
0.9594040514756628
```

XGBoost model

In [87]:

```
from xgboost import XGBClassifier
```

In [88]:

```
XGB_model = XGBClassifier(use_label_encoder=False)
XGB_model.fit(X_res, Y_res)
```

```
[00:56:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behav
ior.
```

Out[88]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=None)
```

In [89]:

```
# make predictions for test data
XGB_y_pred = XGB_model.predict(X_test)

# evaluate predictions
acc_XGB= accuracy_score(Y_test, XGB_y_pred)
print("Accuracy: %.2f%%" % (acc_XGB * 100.0))
```

```
Accuracy: 99.77%
```

In [90]:

```
R2_XGB = r2_score(Y_test, XGB_y_pred)
print(R2_XGB)
```

```
0.9289570900824098
```

COMPARISION

In [91]:

```
Comp={'Models':['Logistic Regression', 'KNN','Random Forest','AdaBoost','SVC','XGBOOST'],
    'Accuracy':[acc_log_reg,acc_KNN,acc_RF,acc_Ada,acc_SVC,acc_XGB],
   'R2 score' :[R2_log_reg,R2_KNN,R2_RF,R2_Ada,R2_SVC,R2_XGB]
   }
```

In [92]:

```
Compare=pd.DataFrame(Comp)
```

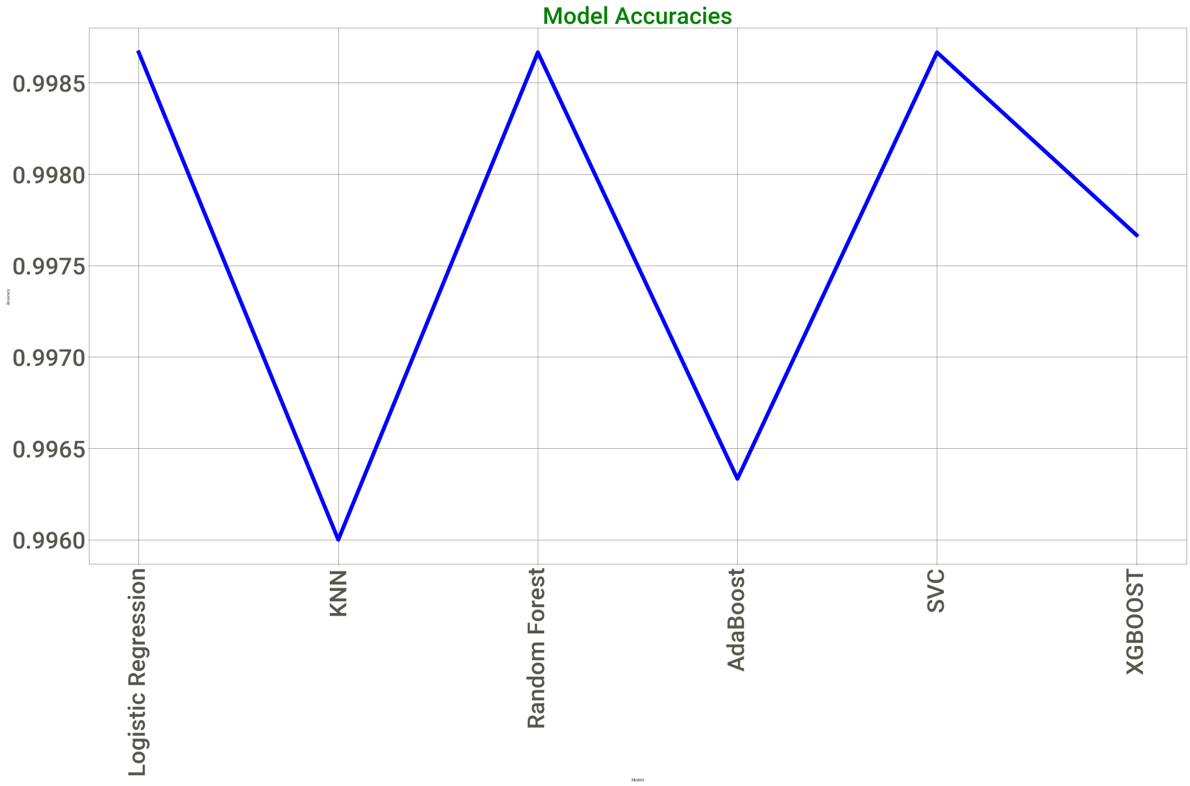
In [93]:

```
print(Compare)
```

```
                Models  Accuracy  R2 score
0  Logistic Regression  0.998667  0.959404
1                  KNN  0.996000  0.878212
2        Random Forest  0.998667  0.959404
3             AdaBoost  0.996333  0.888361
4                  SVC  0.998667  0.959404
5              XGBOOST  0.997667  0.928957
```

In [94]:

```
plt.figure(figsize =(50, 25))
plt.plot(Compare['Models'],Compare['Accuracy'],c='blue', lw=10)
plt.title('Model Accuracies',fontdict={'fontsize': 60,'fontweight' : 60,'color' : 'g'})
plt.xlabel('Models')
plt.ylabel('Acuuracy')
plt.yticks(fontsize=60)
plt.xticks(rotation=90, fontsize=60)
plt.grid()
plt.show()
```



In [ ]: