Created By Rekha Sundararajan

1.Extract Fan Degradation Specific Engine data from the given dataset.

Steps followed:

1. Retrieve the HPC degradation only data for condition1 (sealevel)(FD001)

2. Retrieve the HPC and Fan degradation combo engine dataset for condition1 sealevel (FD003)

3. Since the training data is provided for until failure cycle consider the last cycle for a given engine as max cycle and calculate for each cycle datapoint, the efficiency percentage based on that. (Refer function add_EFFICIENCY_column)

4. Plot features to visualize the distribution for engine efficiency per engine.

5. Identify the common pattern engines from FD003 to FD001 and define a filter to extract only those engines that do not follow the same pattern to form fan degradation specific dataset.

In [1]:

```
#Created by Rekha Sundararajan(RS)
# Fan Degradation prediction using the NASA public dataset provided for TurboFan Engine
```

```python
#Import required Libraries
import pandas as pd
import numpy as np
import random
import os
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
np.random.seed(34)
```

In [2]:

```python
#RS: Retrieve the given data into the dataframe for analysis.
# The given dataset consists of FD001 and FD003 for train, test and separate test file j
# FD001 and FD003 dataset were provided for sealevel condition only and FD001 data conte
index_names=['unit_number','time_cycles']
setting_names = ['setting_1','setting_2','setting_3']
sensor_names = ['s_{}'.format(i+1) for i in range(21)]
col_names  = index_names + setting_names + sensor_names

# train dataset is provided for until the failure timecycle.
dftrain1 = pd.read_csv("train_FD001.txt",sep="\s+",header=None,index_col=False,names=co
dftrain3 = pd.read_csv("train_FD003.txt",sep="\s+",header=None,index_col=False,names=co

#Test dataset provided way before failure
dftest1 = pd.read_csv("test_FD001.txt",sep="\s+",header=None,index_col=False,names=col_
dftest3 = pd.read_csv("test_FD003.txt",sep="\s+",header=None,index_col=False,names=col_

# for the given testdataset for each engine identified by unit number the remaining tim
y_test1 = pd.read_csv("RUL_FD001.txt",sep="\s+",header=None,index_col=False,names=['las
y_test3 = pd.read_csv("RUL_FD003.txt",sep="\s+",header=None,index_col=False,names=['las
```

In [3]:

```python
# create a deep copy so that a back up for original is retained.
train1 = dftrain1.copy()
train3 = dftrain3.copy()
```

In [4]:

```python
# Define a function to add a new Efficiency column. The data for train contains till fa
# based on this efficiency is calculated per cycle as a difference from max.
def add_EFFICIENCY_column(df):
    train_grouped_by_unit = df.groupby(by='unit_number')
    max_time_cycles = train_grouped_by_unit['time_cycles'].max()
    merged = df.merge(max_time_cycles.to_frame(name='max_time_cycles'),left_on='unit_nu
    merged['EFFICIENCY'] = 100*((merged['max_time_cycles'] - merged['time_cycles'])/mer
    merged = merged.drop("max_time_cycles",axis=1)
    return merged
```

In [5]:

```python
# add the efficiency column to the training dataset.
train1 = add_EFFICIENCY_column(train1)
train3 = add_EFFICIENCY_column(train3)
```

In [6]:

```python
# For the given NASA public dataset 21 sensor data have been provided and created a bel
Sensor_dictionary = {}
dict_list=[
    "Fan Inlet Temperature",
```

```
    "LPC Outlet Temperature",
    "HPC Outlet Temperature",
    "LPT Outlet Temperature",
    "Fan Inlet Pressure",
    "Bypass-duct Pressure",
    "HPC Outlet Pressure",
    "Physical Fan Speed",
    "Physical Core Speed",
    "Engine pressure ratio",
    "HPC Outlet Static Pressure",
    "Ratio of Fuel Flow to PS30",
    "Corrected Fan Speed",
    "Corrected Core Speed",
    "Bypass ratio",
    "Burner Fuel-air Ratio",
    "Bleed Enthalpy",
    "Required Fan Speed",
    "Required Fan Conversion Speed",
    "High-pressure turbines cool air flow",
    "Low-pressure turbines cool air flow"
]

si = 1
for x in dict_list:
    Sensor_dictionary['s_'+str(si)] = x
    si += 1
Sensor_dictionary
```

Out[6]:

```
{'s_1': 'Fan Inlet Temperature',
 's_2': 'LPC Outlet Temperature',
 's_3': 'HPC Outlet Temperature',
 's_4': 'LPT Outlet Temperature',
 's_5': 'Fan Inlet Pressure',
 's_6': 'Bypass-duct Pressure',
 's_7': 'HPC Outlet Pressure',
 's_8': 'Physical Fan Speed',
 's_9': 'Physical Core Speed',
 's_10': 'Engine pressure ratio',
 's_11': 'HPC Outlet Static Pressure',
 's_12': 'Ratio of Fuel Flow to PS30',
 's_13': 'Corrected Fan Speed',
 's_14': 'Corrected Core Speed',
 's_15': 'Bypass ratio',
 's_16': 'Burner Fuel-air Ratio',
 's_17': 'Bleed Enthalpy',
 's_18': 'Required Fan Speed',
 's_19': 'Required Fan Conversion Speed',
 's_20': 'High-pressure turbines cool air flow',
 's_21': 'Low-pressure turbines cool air flow'}
```

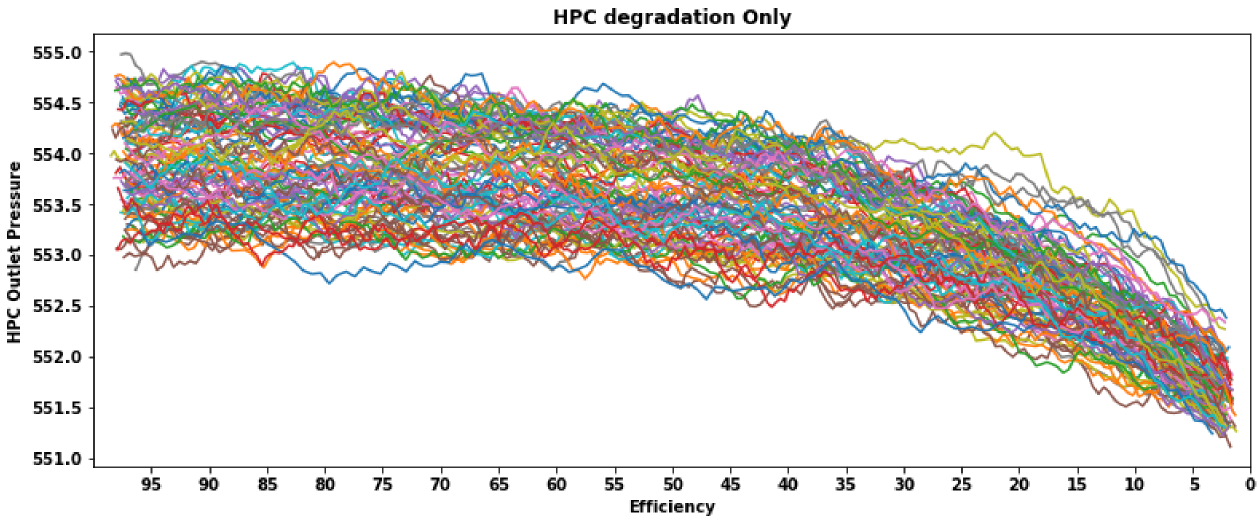
In [7]:

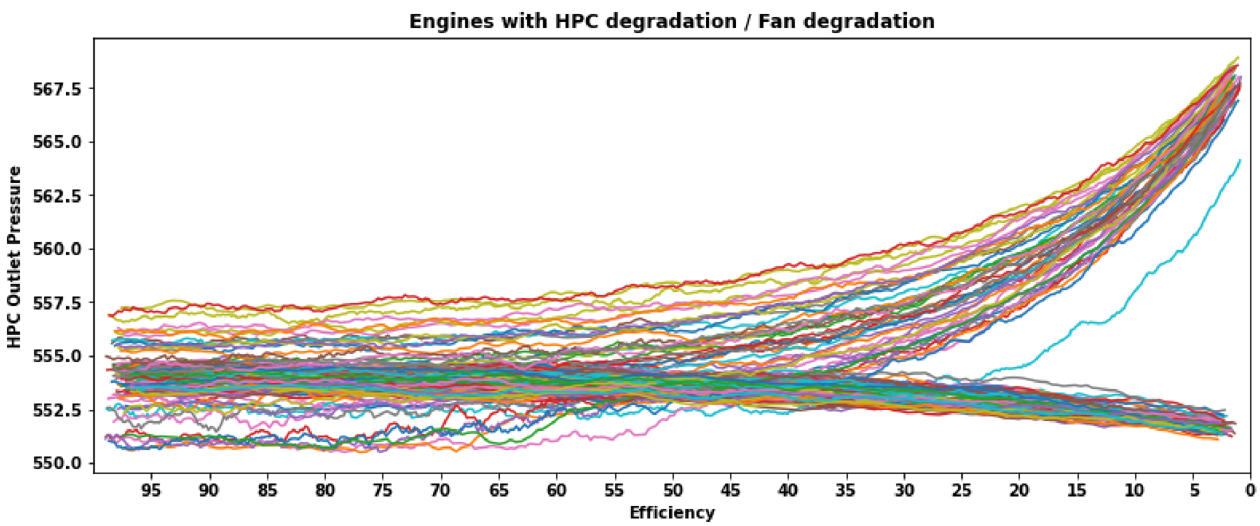
```
#define a function to plot each sensor time series data against efficiency per engine.Ti
def plot_signal(df,sensor_dic,signal_name,tlabel):
    plt.figure(figsize=(13,5))
    for i in df['unit_number'].unique():
        #if i%10 == 0:
            plt.plot('EFFICIENCY',signal_name,data=df[df['unit_number'] == i].rolling(1
    plt.xlim(100,0)
    plt.xticks(np.arange(0,100,5))
    plt.ylabel(sensor_dic[signal_name])
    plt.xlabel('Efficiency')
```

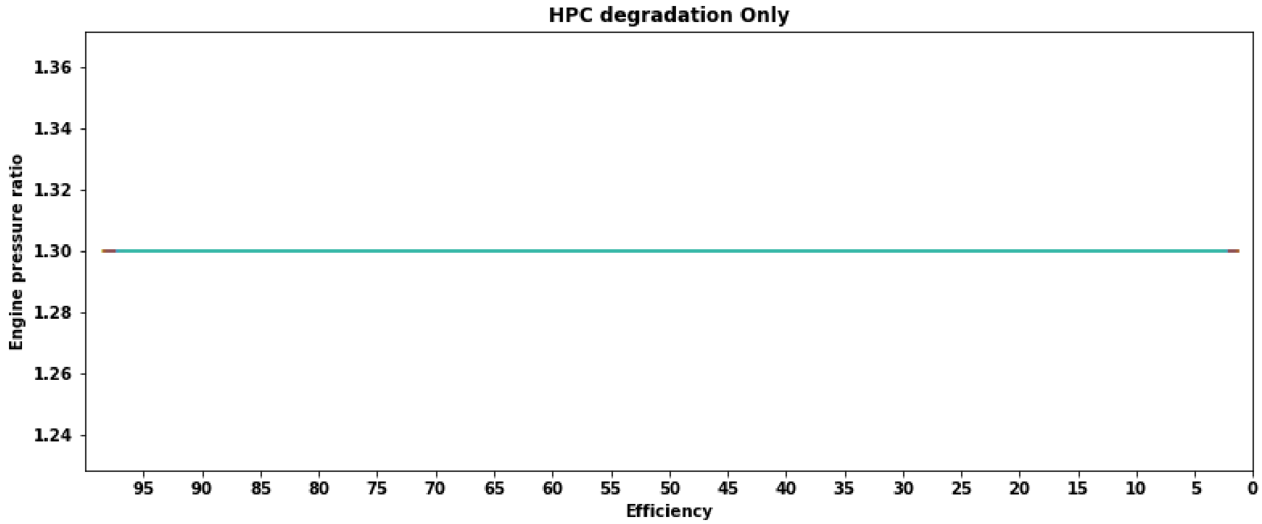
```
    plt.title(tlabel)
    plt.show()
```

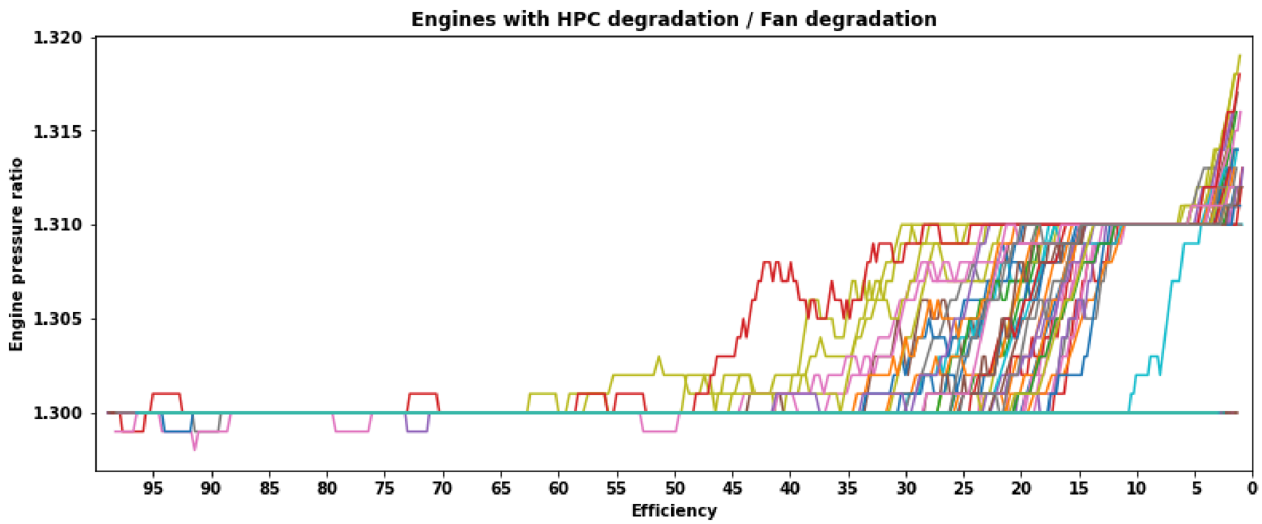
In [8]:

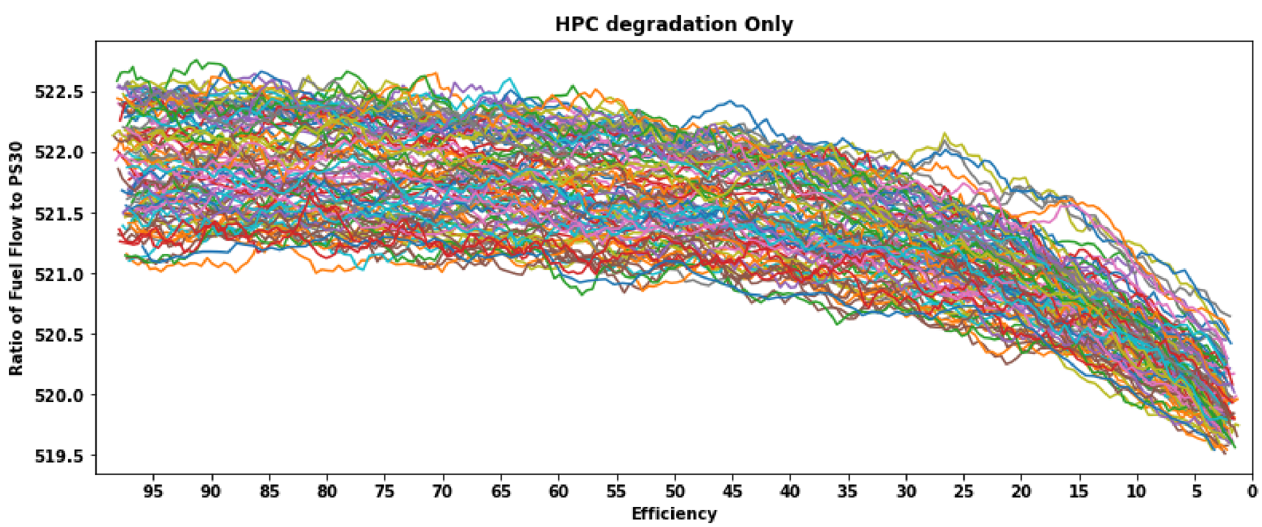
```
# call the plot signal function above to visualize the sensor data.
s=[7,10,12,20,21]#[5,6,15,20,21]#21,7,8,9,13,14,15]
for j in range(0,len(s)):
    try:
        plot_signal(train1,Sensor_dictionary,'s_'+str(s[j]),'HPC degradation Only')
        plot_signal(train3,Sensor_dictionary,'s_'+str(s[j]),'Engines with HPC degradati
    except:
        pass
```

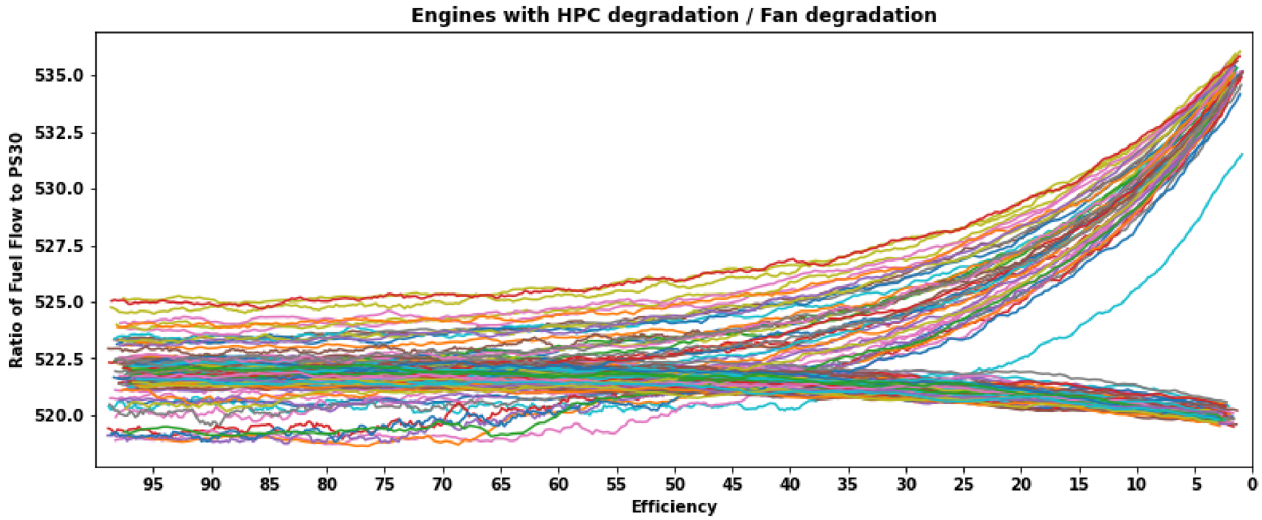




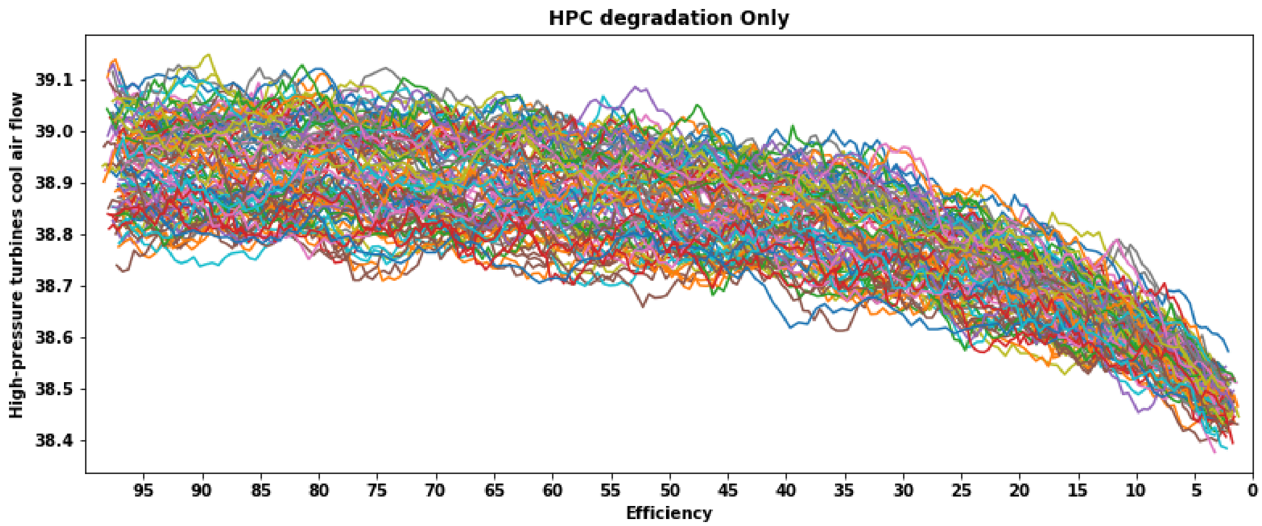

HPC degradation Only
Engine pressure ratio
1.36
1.34
1.32
1.30
1.28
1.26
1.24
95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
Efficiency


Engines with HPC degradation / Fan degradation
Engine pressure ratio
1.320
1.315
1.310
1.305
1.300
95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
Efficiency

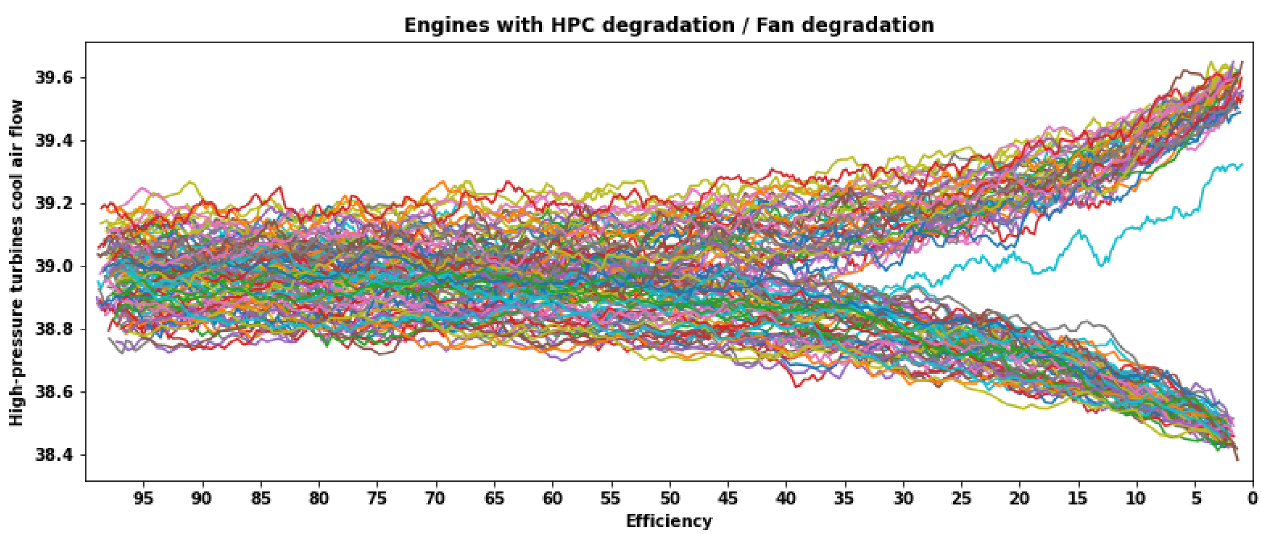

HPC degradation Only
Ratio of Fuel Flow to PS30
522.5
522.0
521.5
521.0
520.5
520.0
519.5
95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
Efficiency



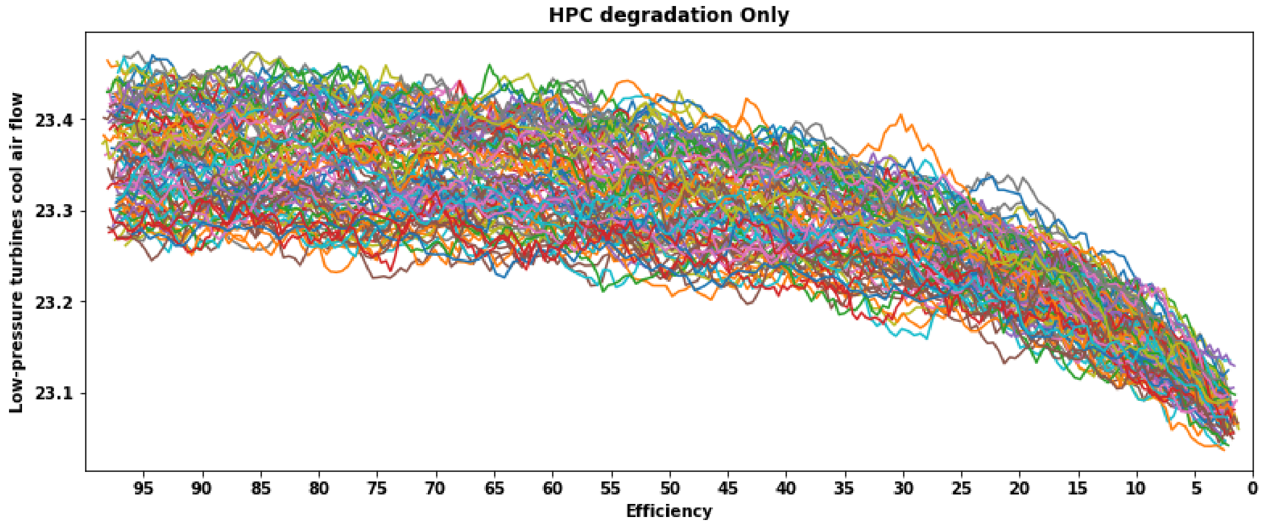
Engines with HPC degradation / Fan degradation
Ratio of Fuel Flow to PS30
Efficiency

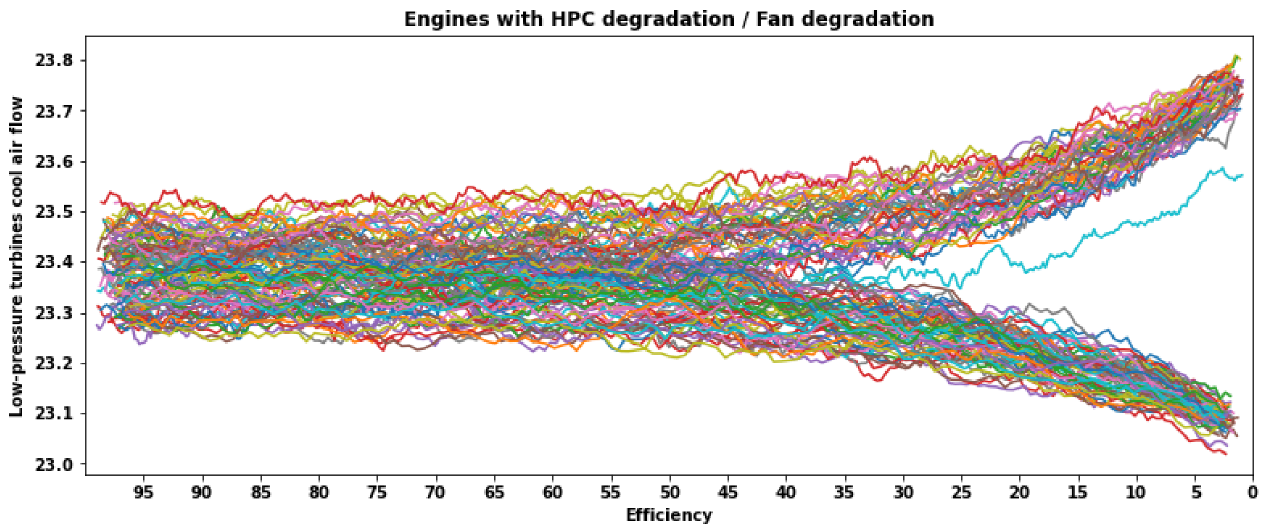


HPC degradation Only
High-pressure turbines cool air flow
Efficiency



Engines with HPC degradation / Fan degradation
High-pressure turbines cool air flow
Efficiency





In [9]:

```
# RS: The FD001 data is provided for HPC degradation mode only and FD003 is provided for
def fandegunitNumList(df):
    # use the HPC outlet pressure deviation > 6 as to have fan degradation specific eng

    train_grouped_by_unit = df.groupby(by='unit_number')
    maxs7 = train_grouped_by_unit['s_7'].max()
    mins7 = train_grouped_by_unit['s_7'].min()
    diffce = maxs7 - mins7
    merged = df.merge(diffce.to_frame(name='diffce'),left_on='unit_number',right_index=

    return merged[merged['diffce'] > 6]['unit_number'].unique()
```

In [10]:

```
train_fan =  train3[train3['unit_number'].isin(fandegunitNumList(train3))]
```

2.Using only the extracted fan degradation only engine dataset find the highly correlating features to the fan efficiency

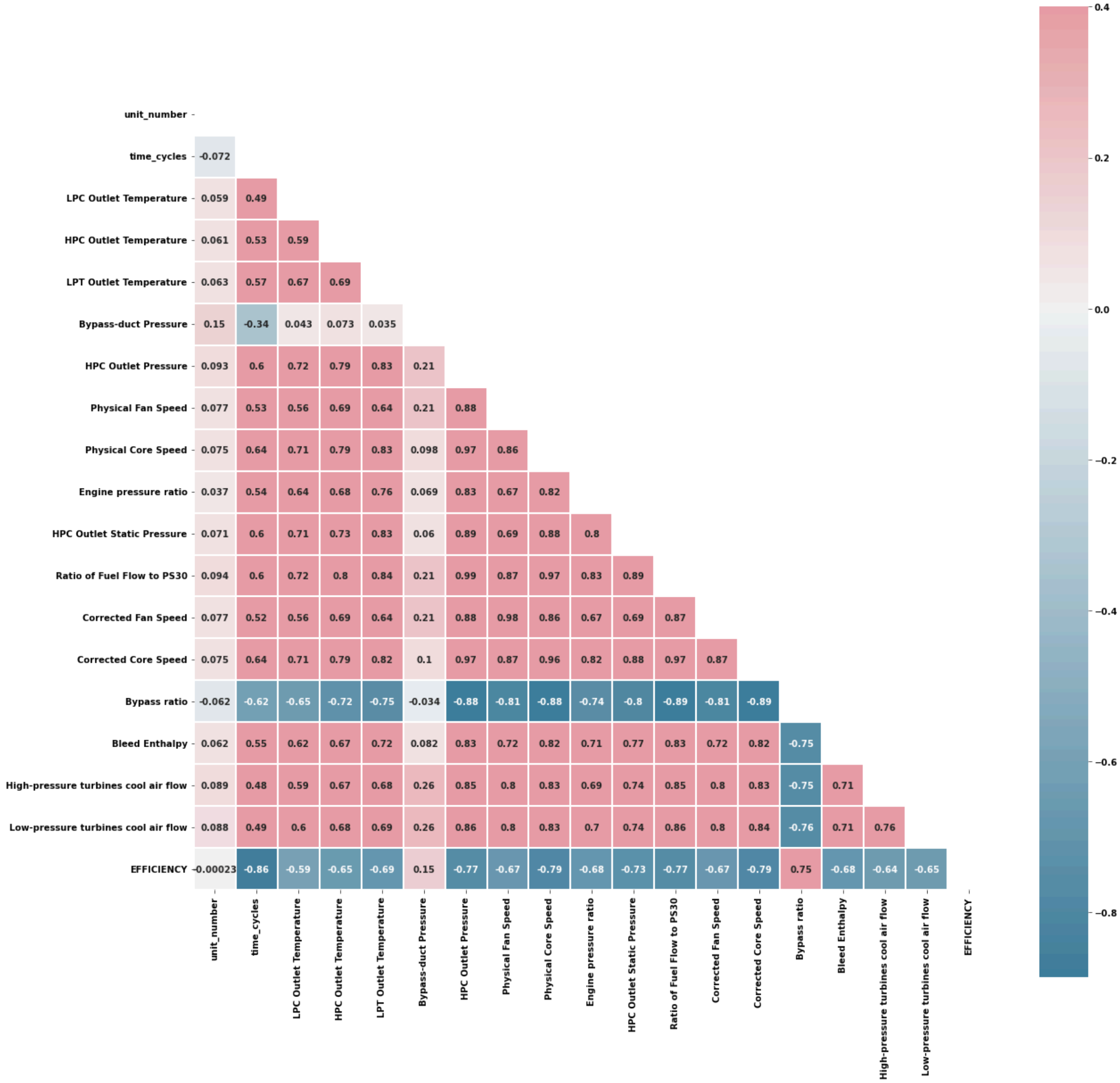
Steps followed:

1. Plot the heatmap to visualize the highly correlating features that can be used for defining a model that can predict fan efficiency.

2. Pick only those features that has > 70% correlation to fan efficiency (either +ve or -ve direction correlation.)

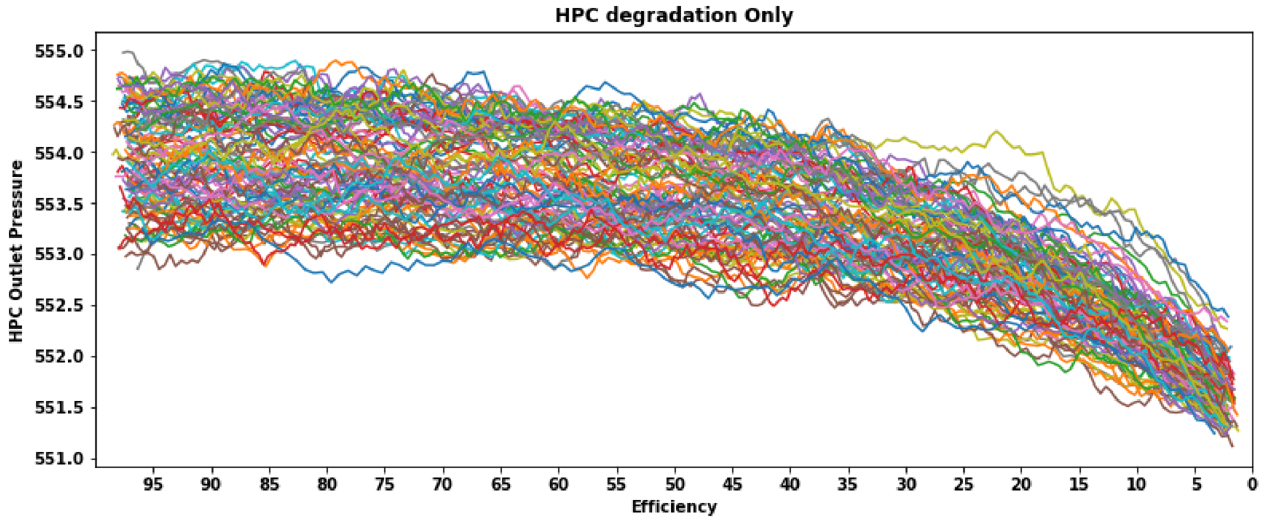
In [11]:

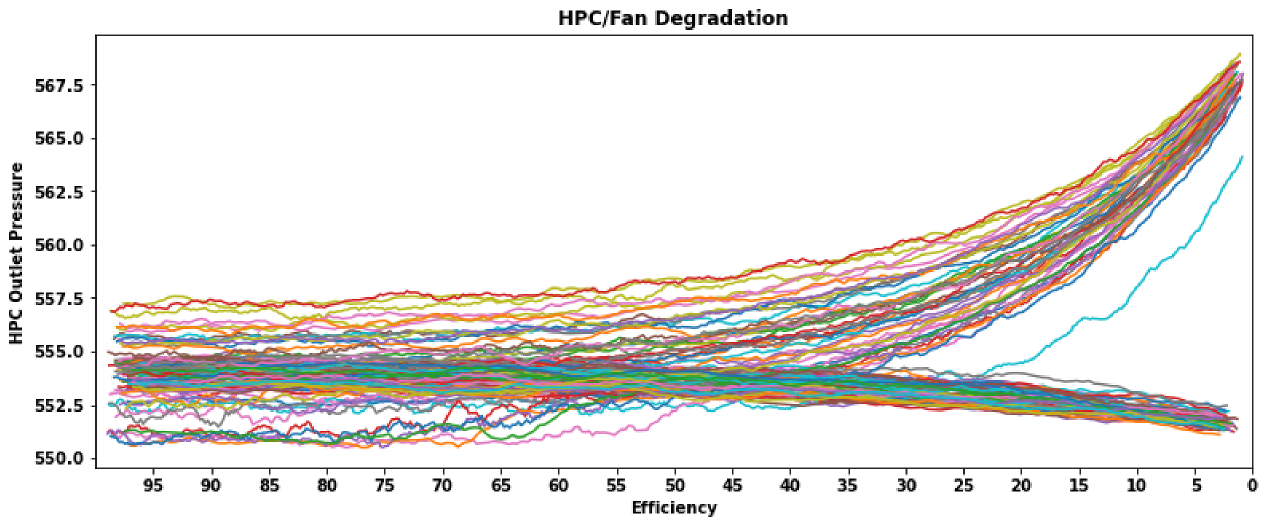
```
# RS: Using Correlation find the sensor data that are highly correlating to identifying
# this will aid in choosing the features to be used in model for fan efficency predicti
train = train_fan
train = train.rename(columns=Sensor_dictionary)
train.drop(columns=['setting_1','setting_2','setting_3','Fan Inlet Temperature','Burner
corr = train.corr()
mask = np.triu(np.ones_like(corr,dtype=bool))
f,ax = plt.subplots(figsize=(10,10))
cmap = sns.diverging_palette(230,10,as_cmap=True)
sns.heatmap(corr,mask=mask,cmap=cmap,vmax=0.4,center=0,square=True,linewidths=0.4,annot
plt.gcf().set_size_inches(20, 20)
plt.show()
```

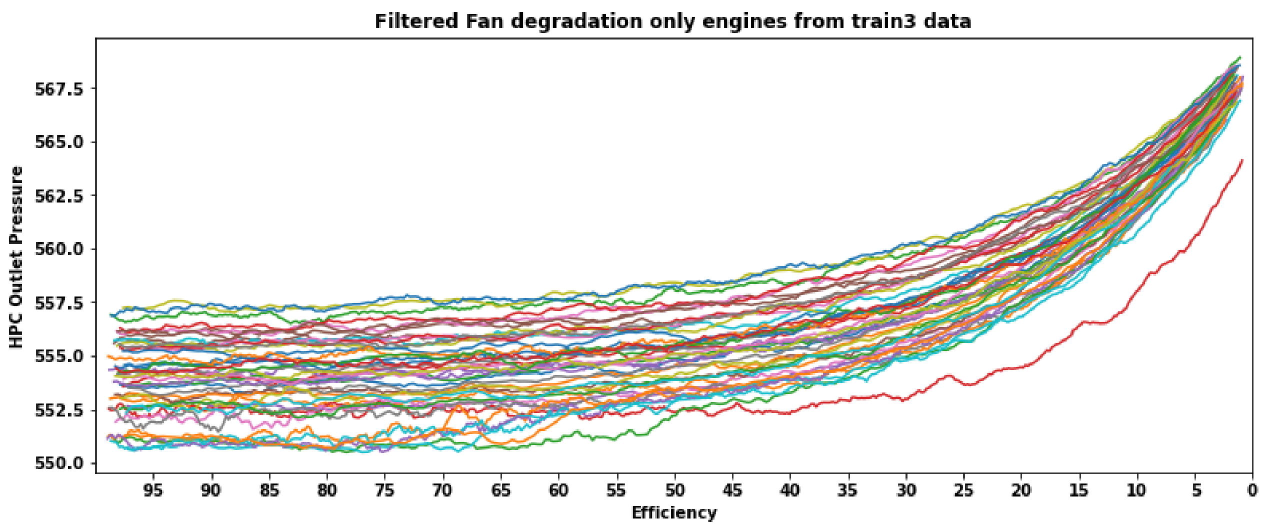


In [12]:

```
s=[7]
for j in range(0,len(s)):
    try:
        plot_signal(train1,Sensor_dictionary,'s_'+str(s[j]),'HPC degradation Only')
        plot_signal(train3,Sensor_dictionary,'s_'+str(s[j]),'HPC/Fan Degradation')
        plot_signal(train_fan,Sensor_dictionary,'s_'+str(s[j]),'Filtered Fan degradatio
    except:
        pass
```







In [13]:

```
corr[corr['EFFICIENCY'] > 0.70]['EFFICIENCY'].sort_values()
```

Out[13]:

```
Bypass ratio    0.745698
EFFICIENCY      1.000000
Name: EFFICIENCY, dtype: float64
```

In [14]:

```
corr[corr['EFFICIENCY'] < -0.70]['EFFICIENCY'].sort_values()
```

Out[14]:
```
time_cycles                  -0.857007
Corrected Core Speed         -0.787749
Physical Core Speed          -0.786991
Ratio of Fuel Flow to PS30   -0.770834
HPC Outlet Pressure          -0.769723
HPC Outlet Static Pressure   -0.728174
Name: EFFICIENCY, dtype: float64
```

3. Create a test dataset and add the efficiency column for test data using the separate RUL dataset provided. so that the actual provided last cycle of failure can be compared to the predicted value for performance evaluation.

In [15]:
```
def add_EFFICIENCY_column_fortest(df,ytest):
    train_grouped_by_unit = df.groupby(by='unit_number')
    ytest.index =np.arange(df['unit_number'].min(),len(ytest)+df['unit_number'].min())
    max_time_cycles = train_grouped_by_unit['time_cycles'].max() + ytest
    merged = df.merge(max_time_cycles.to_frame(name='max_time_cycles'),left_on='unit_nu
    merged['EFFICIENCY'] = 100*((merged['max_time_cycles'] - merged['time_cycles'])/mer
    merged = merged.drop("max_time_cycles",axis=1)
    return merged

test3 = dftest3.copy()
test3 = add_EFFICIENCY_column_fortest(test3,y_test3['lastcycle'])
test3['fan_degradation'] = 1
```

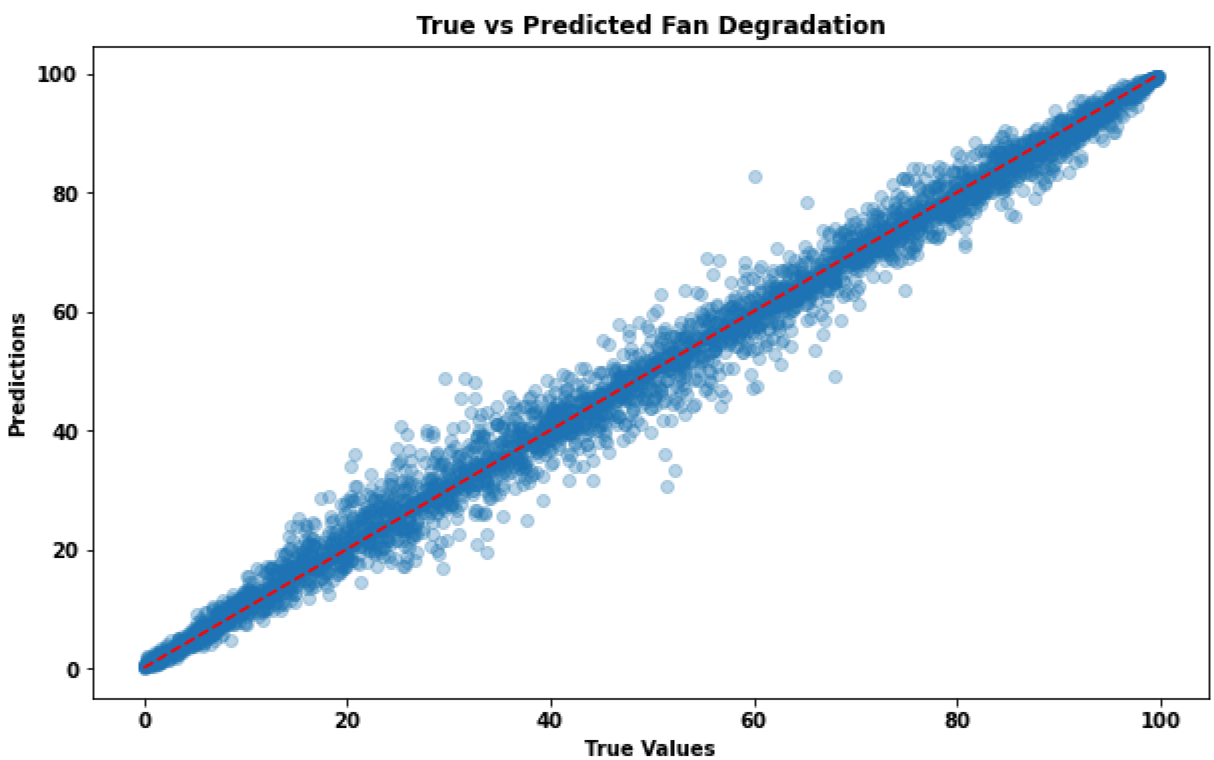
In [16]:
```
features = ['unit_number','time_cycles','s_7','s_9','s_11','s_12','s_14','s_15']
target = ['EFFICIENCY']
train_fan3 = train3[train3['unit_number'].isin(fandegunitNumList(train3))]
x_train = train_fan3[features]
y_train = train_fan3[target]
# commented out scaling as it gave lower performance and hence scaling is not prefereab
#from sklearn.preprocessing import MinMaxScaler,StandardScaler
#scaler=StandardScaler()#MinMaxScaler()
#x_train_scaled = scaler.fit_transform(x_train)
```

4.Design a ML Model using Random regressor and the training dataset - 70% for training and 30% for evaluation and calculate the metrics - MAE,RMSE and R2 score.

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
x_train,x_test,y_train,y_test = train_test_split(x_train,y_train,test_size=0.3,random_s
#x_test_scaled = scaler.fit_transform(x_test)
```

```
# Train a Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(x_train, y_train)
# Tried other ML models and random regressor performed better than xgboost.
#!pip install xgboost
#import xgboost
#xgb = xgboost.XGBRegressor(n_estimators=110, learning_rate=0.02,gamma=0,subsample=0.8,
#xgb.fit(x_train_scaled,y_train)


# Make predictions using the test split of training data for evaluation
y_pred = rf_regressor.predict(x_test)
#y_pred = xgb.predict(x_test_scaled)
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R² Score: {r2}")

# Plot true vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', color='red')

plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True vs Predicted Fan Degradation')
plt.show()
```

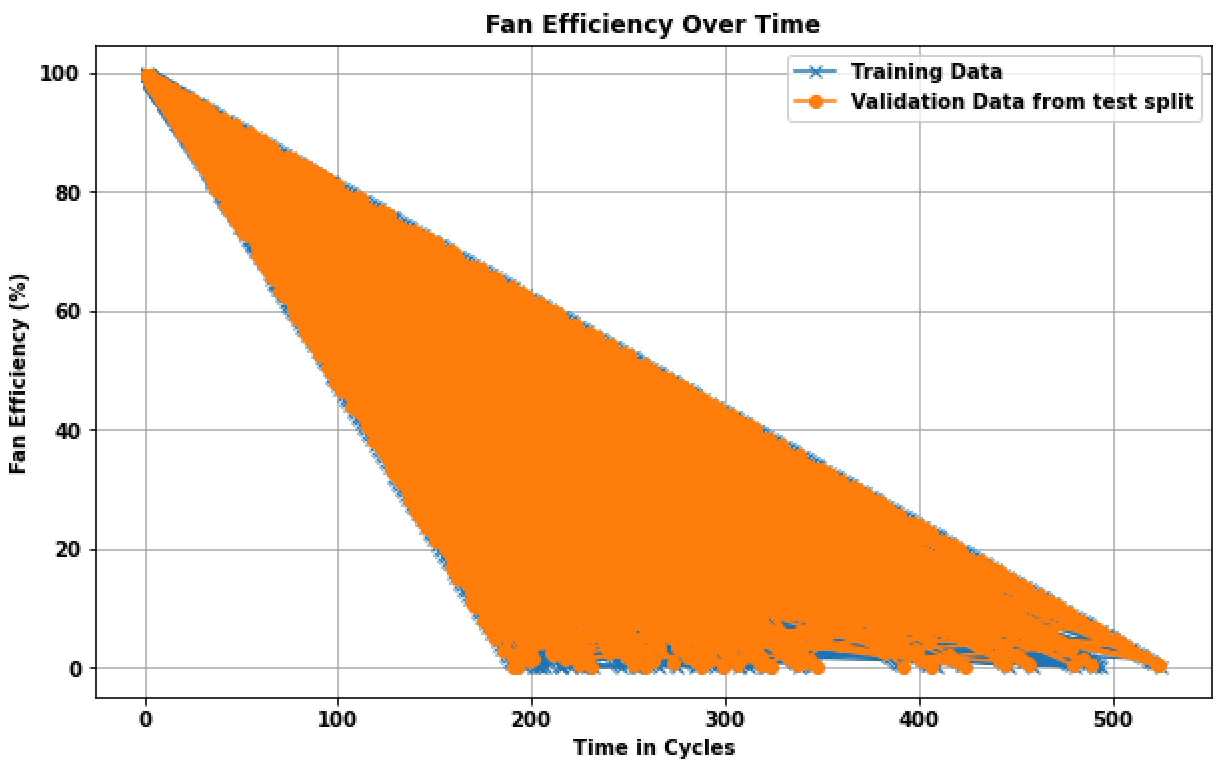
```
Mean Absolute Error (MAE): 2.1979486748268364
Root Mean Squared Error (RMSE): 3.213318406825154
R² Score: 0.9874435469458415
```



5. Plot the time in cycles against the fan efficiency for both training and evaluation dataset.

In [18]:

```
# Plot fan efficiency over time
plt.figure(figsize=(10, 6))
plt.plot(x_train['time_cycles'], y_train['EFFICIENCY'], marker='x', label='Training Dat
plt.plot(x_test['time_cycles'], y_test['EFFICIENCY'], marker='o', label='Validation Dat
plt.xlabel('Time in Cycles')
plt.ylabel('Fan Efficiency (%)')
plt.title('Fan Efficiency Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



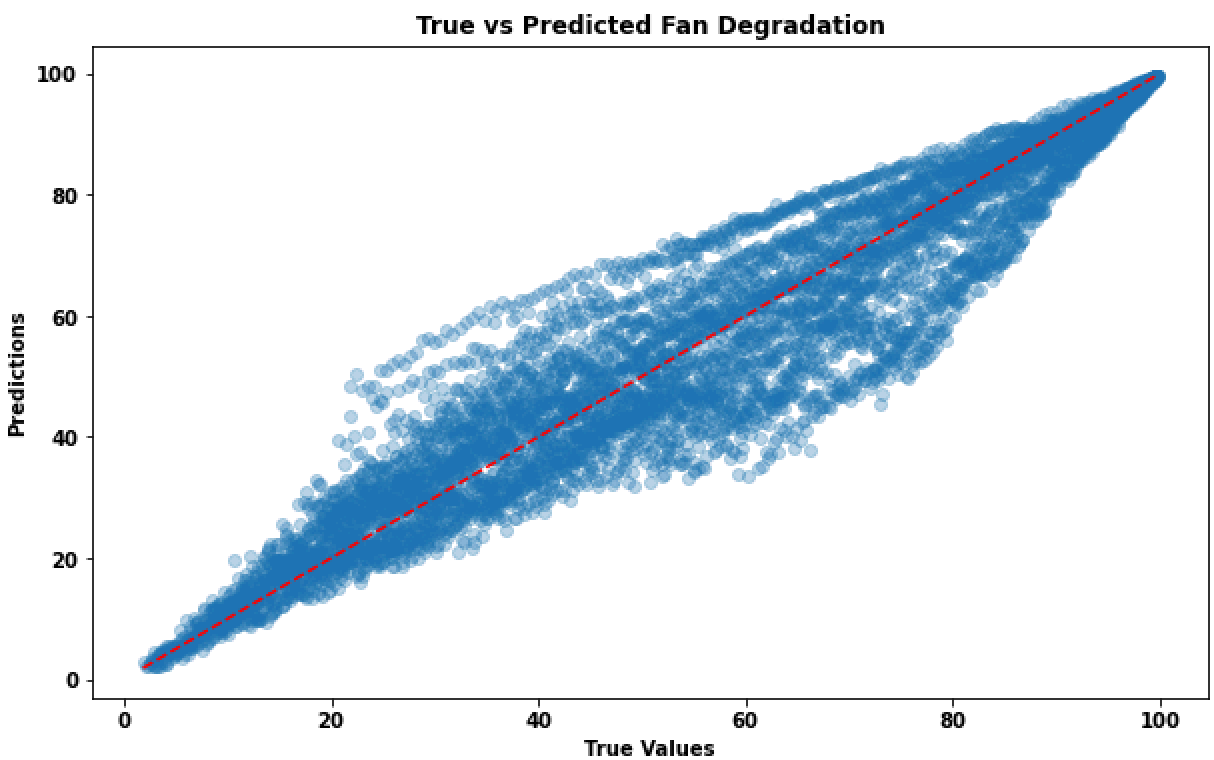
6.using the defined Random regressor Model and the actual test dataset perform the performance evaluation and calculate the metrics - MAE,RMSE and R2 score.

Note that the model prediction accuracy is better when close to 20% or less efficiency.

In [19]:

```python
test_fan3 = test3[test3['unit_number'].isin(fandegunitNumList(test3))]
x_test = test_fan3[features]

y_test = test_fan3[target]

# Make predictions using the real test data provided for efficiency predictions.
y_pred = rf_regressor.predict(x_test)
#y_pred = xgb.predict(x_test_scaled)
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R² Score: {r2}")

# Plot true vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', color='red')

plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True vs Predicted Fan Degradation')
plt.show()
```

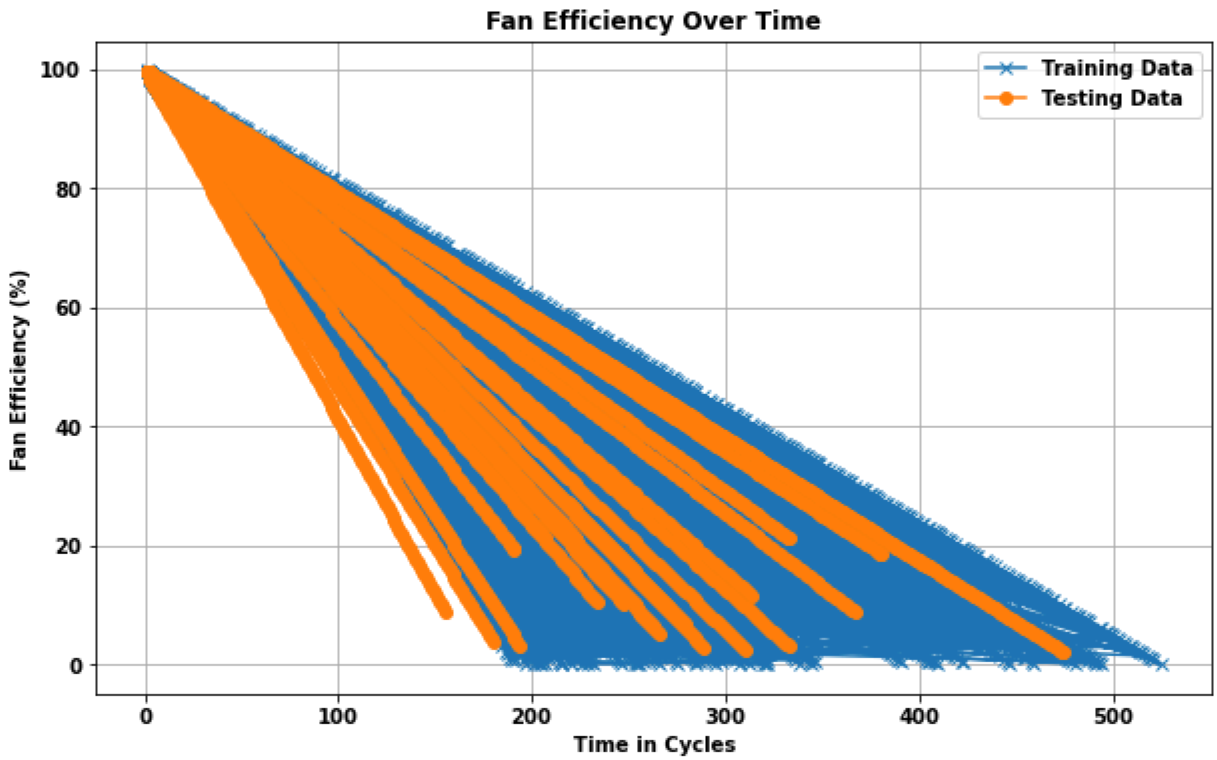
```
Mean Absolute Error (MAE): 6.141201279140579
Root Mean Squared Error (RMSE): 8.343902079353155
R² Score: 0.8983191388679077
```



7. Plot the time in cycles against the fan efficiency for both training and test dataset.

In [20]:

```
# Plot fan efficiency over time
plt.figure(figsize=(10, 6))
plt.plot(x_train['time_cycles'], y_train['EFFICIENCY'], marker='x', label='Training Dat
plt.plot(x_test['time_cycles'], y_test['EFFICIENCY'], marker='o', label='Testing Data')
plt.xlabel('Time in Cycles')
plt.ylabel('Fan Efficiency (%)')
plt.title('Fan Efficiency Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



8. Details of the training and test engine dataset considered

In [21]:

```
print("Engines used for training the model that are considered to have Fan degradation o
#[  1   3  20  21  23  24  30  39  40  46  62  64  71  72  75  77  78  81
 # 82  92  94  99 100]
lst = train_fan3['unit_number'].unique()
print("total engines considered for training:",int(len(lst)*0.7))
print("total engines considered for validation:",int(len(lst)*0.3))
for i in range(len(lst)):
   print("Number of cycles provided until failure for Engine with id ",lst[i]," is ",le
```

```
Engines used for training the model that are considered to have Fan degradation only:
 [ 2  7  9 10 11 16 17 18 19 20 21 24 27 33 34 37 38 39 41 42 43 45 46 49
 55 57 59 60 62 71 72 73 75 77 81 82 84 85 88 89 94 96 97 98]
total engines considered for training: 30
total engines considered for validation: 13
Number of cycles provided until failure for Engine with id  2  is  253
Number of cycles provided until failure for Engine with id  7  is  424
Number of cycles provided until failure for Engine with id  9  is  406
```

```
Number of cycles provided until failure for Engine with id  10  is  481
Number of cycles provided until failure for Engine with id  11  is  197
Number of cycles provided until failure for Engine with id  16  is  344
Number of cycles provided until failure for Engine with id  17  is  312
Number of cycles provided until failure for Engine with id  18  is  447
Number of cycles provided until failure for Engine with id  19  is  229
Number of cycles provided until failure for Engine with id  20  is  338
Number of cycles provided until failure for Engine with id  21  is  220
Number of cycles provided until failure for Engine with id  24  is  494
Number of cycles provided until failure for Engine with id  27  is  320
Number of cycles provided until failure for Engine with id  33  is  231
Number of cycles provided until failure for Engine with id  34  is  459
Number of cycles provided until failure for Engine with id  37  is  324
Number of cycles provided until failure for Engine with id  38  is  201
Number of cycles provided until failure for Engine with id  39  is  288
Number of cycles provided until failure for Engine with id  41  is  295
Number of cycles provided until failure for Engine with id  42  is  193
Number of cycles provided until failure for Engine with id  43  is  321
Number of cycles provided until failure for Engine with id  45  is  205
Number of cycles provided until failure for Engine with id  46  is  204
Number of cycles provided until failure for Engine with id  49  is  256
Number of cycles provided until failure for Engine with id  55  is  525
Number of cycles provided until failure for Engine with id  57  is  215
Number of cycles provided until failure for Engine with id  59  is  299
Number of cycles provided until failure for Engine with id  60  is  190
Number of cycles provided until failure for Engine with id  62  is  246
Number of cycles provided until failure for Engine with id  71  is  409
Number of cycles provided until failure for Engine with id  72  is  232
Number of cycles provided until failure for Engine with id  73  is  215
Number of cycles provided until failure for Engine with id  75  is  259
Number of cycles provided until failure for Engine with id  77  is  255
Number of cycles provided until failure for Engine with id  81  is  347
Number of cycles provided until failure for Engine with id  82  is  285
Number of cycles provided until failure for Engine with id  84  is  226
Number of cycles provided until failure for Engine with id  85  is  266
Number of cycles provided until failure for Engine with id  88  is  322
Number of cycles provided until failure for Engine with id  89  is  207
Number of cycles provided until failure for Engine with id  94  is  392
Number of cycles provided until failure for Engine with id  96  is  491
Number of cycles provided until failure for Engine with id  97  is  275
Number of cycles provided until failure for Engine with id  98  is  307
```

In [22]:

```python
print("Note: (The id used in training and test are different and donot represent the sa
print("Engines used for testing the model that are considered to have Fan degradation o
#[  1   3  20  21  23  24  30  39  40  46  62  64  71  72  75  77  78  81
 # 82  92  94  99 100]
lst = test_fan3['unit_number'].unique()
print("total engines considered for testing:",len(lst))
for i in range(len(lst)):
   print("Number of cycles provided before failure for Engine with id ",lst[i]," is ",l
```

```
Note: (The id used in training and test are different and donot represent the same engin
e.
 These are just numbers to identify different engines.)
Engines used for testing the model that are considered to have Fan degradation only:
 [  1   3  20  21  23  24  30  39  40  46  62  64  71  72  75  77  78  81
  82  92  94  99 100]
total engines considered for testing: 23
Number of cycles provided before failure for Engine with id  1  is  233
Number of cycles provided before failure for Engine with id  3  is  234
Number of cycles provided before failure for Engine with id  20  is  207
Number of cycles provided before failure for Engine with id  21  is  263
Number of cycles provided before failure for Engine with id  23  is  405
```

```
Number of cycles provided before failure for Engine with id  24  is  475
Number of cycles provided before failure for Engine with id  30  is  333
Number of cycles provided before failure for Engine with id  39  is  310
Number of cycles provided before failure for Engine with id  40  is  313
Number of cycles provided before failure for Engine with id  46  is  180
Number of cycles provided before failure for Engine with id  62  is  224
Number of cycles provided before failure for Engine with id  64  is  271
Number of cycles provided before failure for Engine with id  71  is  367
Number of cycles provided before failure for Engine with id  72  is  232
Number of cycles provided before failure for Engine with id  75  is  191
Number of cycles provided before failure for Engine with id  77  is  381
Number of cycles provided before failure for Engine with id  78  is  279
Number of cycles provided before failure for Engine with id  81  is  155
Number of cycles provided before failure for Engine with id  82  is  194
Number of cycles provided before failure for Engine with id  92  is  266
Number of cycles provided before failure for Engine with id  94  is  333
Number of cycles provided before failure for Engine with id  99  is  289
Number of cycles provided before failure for Engine with id  100  is  247
```