```python
import pandas as pd

# Load training data for FD001
train_fd001 = pd.read_csv('train_FD001.txt', sep=' ', header=None)

# Load test data for FD001
test_fd001 = pd.read_csv('test_FD001.txt', sep=' ', header=None)

# Load true RUL values for FD001
rul_fd001 = pd.read_csv('RUL_FD001.txt', sep=' ', header=None)

train_fd001 = train_fd001.dropna(axis=1, how='all')
test_fd001 = test_fd001.dropna(axis=1, how='all')

column_names = [
    'unit_number', 'time_in_cycles', 'operational_setting_1',
'operational_setting_2',
    'operational_setting_3'] + [f'sensor_measurement_{i}' for i in
range(1, 22)]

train_fd001.columns = column_names
test_fd001.columns = column_names

print(train_fd001.head())
print(test_fd001.head())
print(rul_fd001.head())
```

```
   unit_number  time_in_cycles  operational_setting_1
operational_setting_2  \
0            1               1                -0.0007                 -
0.0004
1            1               2                 0.0019                 -
0.0003
2            1               3                -0.0043
0.0003
3            1               4                 0.0007
0.0000
4            1               5                -0.0019                 -
0.0002

   operational_setting_3  sensor_measurement_1
sensor_measurement_2  \
0                  100.0                518.67                641.82

1                  100.0                518.67                642.15

2                  100.0                518.67                642.35

3                  100.0                518.67                642.35

4                  100.0                518.67                642.37
```

```
   sensor_measurement_3  sensor_measurement_4
sensor_measurement_5  ...  \
0               1589.70               1400.60
14.62  ...
1               1591.82               1403.14
14.62  ...
2               1587.99               1404.20
14.62  ...
3               1582.79               1401.87
14.62  ...
4               1582.85               1406.22
14.62  ...

   sensor_measurement_12  sensor_measurement_13  sensor_measurement_14
\
0                 521.66                2388.02                8138.62

1                 522.28                2388.07                8131.49

2                 522.42                2388.03                8133.23

3                 522.86                2388.08                8133.83

4                 522.19                2388.04                8133.80


   sensor_measurement_15  sensor_measurement_16  sensor_measurement_17
\
0                 8.4195                   0.03                    392

1                 8.4318                   0.03                    392

2                 8.4178                   0.03                    390

3                 8.3682                   0.03                    392

4                 8.4294                   0.03                    393


   sensor_measurement_18  sensor_measurement_19  sensor_measurement_20
\
0                   2388                  100.0                  39.06

1                   2388                  100.0                  39.00

2                   2388                  100.0                  38.95

3                   2388                  100.0                  38.88
```

```
4                    2388                   100.0                   38.90


   sensor_measurement_21
0                23.4190
1                23.4236
2                23.3442
3                23.3739
4                23.4044

[5 rows x 26 columns]
   unit_number  time_in_cycles  operational_setting_1
operational_setting_2  \
0            1               1                 0.0023
0.0003
1            1               2                -0.0027                 -
0.0003
2            1               3                 0.0003
0.0001
3            1               4                 0.0042
0.0000
4            1               5                 0.0014
0.0000

   operational_setting_3  sensor_measurement_1
sensor_measurement_2  \
0                  100.0                518.67                643.02

1                  100.0                518.67                641.71

2                  100.0                518.67                642.46

3                  100.0                518.67                642.44

4                  100.0                518.67                642.51


   sensor_measurement_3  sensor_measurement_4
sensor_measurement_5  ...  \
0               1585.29               1398.21
14.62  ...
1               1588.45               1395.42
14.62  ...
2               1586.94               1401.34
14.62  ...
3               1584.12               1406.42
14.62  ...
4               1587.19               1401.92
14.62  ...
```

```
   sensor_measurement_12  sensor_measurement_13  sensor_measurement_14  \
0                 521.72                2388.03                8125.55   
1                 522.16                2388.06                8139.62   
2                 521.97                2388.03                8130.10   
3                 521.38                2388.05                8132.90   
4                 522.15                2388.03                8129.54   

   sensor_measurement_15  sensor_measurement_16  sensor_measurement_17  \
0                 8.4052                   0.03                    392   
1                 8.3803                   0.03                    393   
2                 8.4441                   0.03                    393   
3                 8.3917                   0.03                    391   
4                 8.4031                   0.03                    390   

   sensor_measurement_18  sensor_measurement_19  sensor_measurement_20  \
0                   2388                  100.0                  38.86   
1                   2388                  100.0                  39.02   
2                   2388                  100.0                  39.08   
3                   2388                  100.0                  39.00   
4                   2388                  100.0                  38.99   

   sensor_measurement_21  
0                23.3735  
1                23.3916  
2                23.4166  
3                23.3737  
4                23.4130  

[5 rows x 26 columns]
     0   1
0  112 NaN
1   98 NaN
2   69 NaN
```

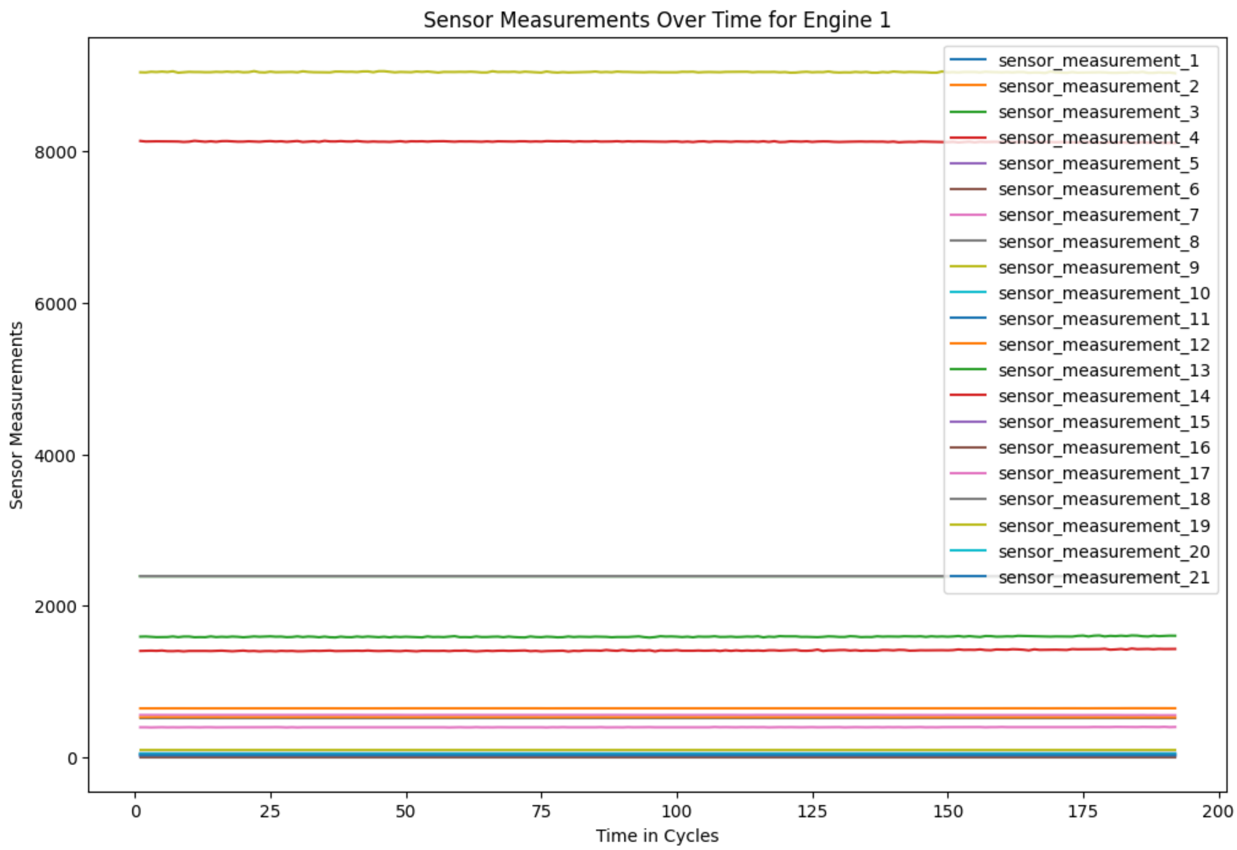
```
3   82 NaN
4   91 NaN
```

```
train_fd001.to_csv('train_fd001.csv', index=False)
test_fd001.to_csv('test_fd001.csv', index=False)
rul_fd001.to_csv('rul_fd001.csv', index=False)

print(train_fd001.isnull().sum())
```

```
unit_number              0
time_in_cycles           0
operational_setting_1    0
operational_setting_2    0
operational_setting_3    0
sensor_measurement_1     0
sensor_measurement_2     0
sensor_measurement_3     0
sensor_measurement_4     0
sensor_measurement_5     0
sensor_measurement_6     0
sensor_measurement_7     0
sensor_measurement_8     0
sensor_measurement_9     0
sensor_measurement_10    0
sensor_measurement_11    0
sensor_measurement_12    0
sensor_measurement_13    0
sensor_measurement_14    0
sensor_measurement_15    0
sensor_measurement_16    0
sensor_measurement_17    0
sensor_measurement_18    0
sensor_measurement_19    0
sensor_measurement_20    0
sensor_measurement_21    0
dtype: int64
```

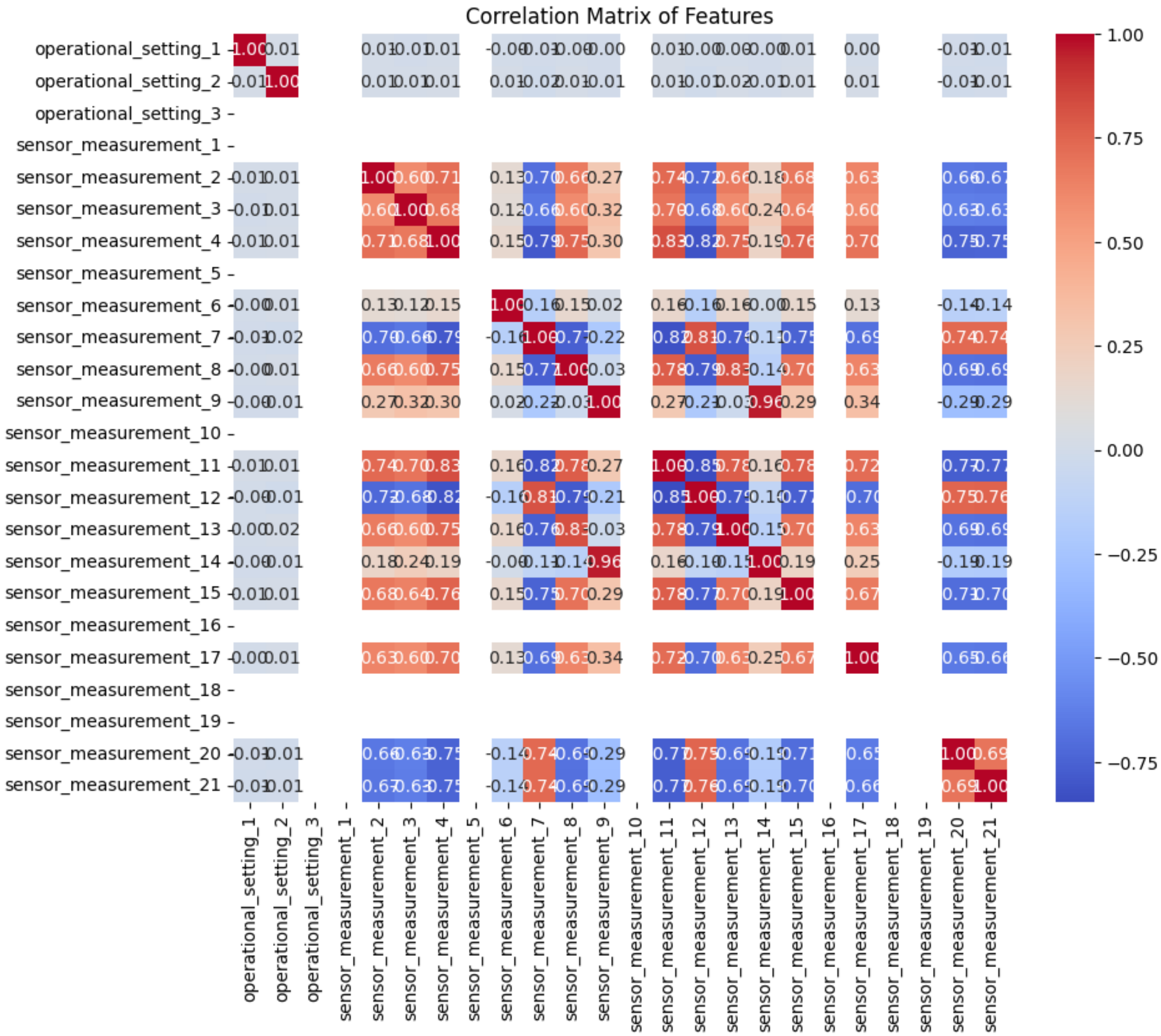
```
import matplotlib.pyplot as plt

# Plot sensor measurements for a single engine
engine_1 = train_fd001[train_fd001['unit_number'] == 1]
plt.figure(figsize=(12, 8))
plt.plot(engine_1['time_in_cycles'], engine_1.iloc[:, 5:],
label=engine_1.columns[5:])
plt.title('Sensor Measurements Over Time for Engine 1')
plt.xlabel('Time in Cycles')
plt.ylabel('Sensor Measurements')
plt.legend(loc='upper right')

# Save the figure to a file
```

```
plt.savefig('engine_1_sensor_measurements.png')

# Display the plot
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a figure
plt.figure(figsize=(10, 8))

# Generate the correlation matrix
corr_matrix = train_fd001.iloc[:, 2:].corr()  # Exclude unit_number
and time_in_cycles

# Plot the heatmap
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')

# Add a title to the plot
plt.title('Correlation Matrix of Features')

# Save the plot to a file
plt.savefig("correlation.png")
```

```
# Display the plot
plt.show()
```



Feature engineering

```
window_size = 5
for i in range(1, 22):  # sensor_measurement_1 to
sensor_measurement_21
    sensor_col = f'sensor_measurement_{i}'
    train_fd001[f'{sensor_col}_rolling_mean'] =
train_fd001.groupby('unit_number')
[sensor_col].rolling(window=window_size).mean().reset_index(0,
drop=True)
    train_fd001[f'{sensor_col}_rolling_std'] =
train_fd001.groupby('unit_number')
```

```
[sensor_col].rolling(window=window_size).std().reset_index(0,
drop=True)

for i in range(1, 22):
    sensor_col = f'sensor_measurement_{i}'
    train_fd001[f'{sensor_col}_diff'] =
train_fd001.groupby('unit_number')[sensor_col].diff().fillna(0)

max_cycle = train_fd001.groupby('unit_number')
['time_in_cycles'].transform('max')
train_fd001['normalized_cycle'] = train_fd001['time_in_cycles'] /
max_cycle
```

model develpoment

```
def apply_feature_engineering(df):
    # Apply rolling mean and standard deviation
    for i in range(1, 22):  # Assuming you have 21 sensor measurements
        col_name = f'sensor_measurement_{i}'
        df[f'{col_name}_rolling_mean'] =
df[col_name].rolling(window=5).mean()
        df[f'{col_name}_rolling_std'] =
df[col_name].rolling(window=5).std()
        df[f'{col_name}_diff'] = df[col_name].diff()

    # Normalize the cycle count
    df['normalized_cycle'] = df['time_in_cycles'] /
df['time_in_cycles'].max()
    return df

# Align the test set features with the training set features
X_test_last_cycle =
X_test_last_cycle.reindex(columns=X_train_fe.columns, fill_value=0)

# Impute missing values if needed
X_test_imputed = imputer.transform(X_test_last_cycle)

# Make predictions
y_test_pred = rf_model.predict(X_test_imputed)

# Check if NaNs are present after each operation
def check_for_nan(df, step_name):
    if df.isnull().values.any():
        print(f"NaN values found after {step_name}")
        print(df.isnull().sum())
    else:
        print(f"No NaN values found after {step_name}")

# Step 1: Check for NaNs after initial filtering
check_for_nan(X_test_last_cycle, "initial filtering")
```

```python
# Assuming 'normalized_cycle' is calculated as follows (example):
if 'cycle' in X_test_last_cycle.columns:
    X_test_last_cycle['normalized_cycle'] = X_test_last_cycle['cycle']
/ X_test_last_cycle.groupby('unit_number')['cycle'].transform('max')

# Step 2: Check for NaNs after calculating 'normalized_cycle'
check_for_nan(X_test_last_cycle, "calculating normalized_cycle")

# Rolling statistics
rolling_window = 5  # Example window size
for col in [f'sensor_measurement_{i}' for i in range(1, 22)]:
    X_test_last_cycle[f'{col}_rolling_mean'] =
X_test_last_cycle[col].rolling(window=rolling_window).mean()
    X_test_last_cycle[f'{col}_rolling_std'] =
X_test_last_cycle[col].rolling(window=rolling_window).std()

# Step 3: Check for NaNs after rolling statistics
check_for_nan(X_test_last_cycle, "calculating rolling statistics")

# Differences
for col in [f'sensor_measurement_{i}' for i in range(1, 22)]:
    X_test_last_cycle[f'{col}_diff'] = X_test_last_cycle[col].diff()

# Step 4: Check for NaNs after calculating differences
check_for_nan(X_test_last_cycle, "calculating differences")

# Drop rows with NaN values generated by the rolling and diff
operations
X_test_last_cycle = X_test_last_cycle.dropna()

# Step 5: Check for NaNs after dropping rows
check_for_nan(X_test_last_cycle, "dropping NaN rows")

# Ensure all necessary features are present in X_test_last_cycle
missing_columns = set(X_train_fe.columns) -
set(X_test_last_cycle.columns)
for col in missing_columns:
    X_test_last_cycle[col] = 0  # or some other default value

# Align columns with the training data
X_test_last_cycle = X_test_last_cycle[X_train_fe.columns]

# Step 6: Final check before imputation
check_for_nan(X_test_last_cycle, "final check before imputation")

# Impute missing values
X_test_last_cycle = pd.DataFrame(imputer.transform(X_test_last_cycle),
columns=X_test_last_cycle.columns)
```

```python
# Step 7: Final check after imputation
check_for_nan(X_test_last_cycle, "imputation")

import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error

# Assuming rul_fd001 is a DataFrame, convert it to a numpy array first
rul_fd001 = rul_fd001.values

# Ensure rul_fd001 is a single-dimensional array
rul_fd001 = rul_fd001.ravel()

# Check lengths of rul_fd001 and y_test_pred
print(f"Length of rul_fd001: {len(rul_fd001)}")
print(f"Length of y_test_pred: {len(y_test_pred)}")

# Check for NaN values in rul_fd001
print(f"NaN values in rul_fd001: {np.isnan(rul_fd001).sum()}")

# Handle NaN values in rul_fd001 if any (remove or impute them)
rul_fd001 = rul_fd001[~np.isnan(rul_fd001)]

# Ensure X_test_last_cycle_imputed and rul_fd001 are aligned by length
if len(rul_fd001) != len(y_test_pred):
    print("Mismatch in lengths after NaN removal. Trimming to match 
the shorter length.")
    min_len = min(len(rul_fd001), len(y_test_pred))
    rul_fd001 = rul_fd001[:min_len]
    y_test_pred = y_test_pred[:min_len]

# Check final lengths
print(f"Final Length of rul_fd001: {len(rul_fd001)}")
print(f"Final Length of y_test_pred: {len(y_test_pred)}")

# Calculate the Mean Squared Error on the test set
mse_test = mean_squared_error(rul_fd001, y_test_pred)
print(f'Mean Squared Error on Test Set: {mse_test}')
```

```
Length of rul_fd001: 200
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call 
last)
Cell In[16], line 13
     11 # Check lengths of rul_fd001 and y_test_pred
     12 print(f"Length of rul_fd001: {len(rul_fd001)}")
---> 13 print(f"Length of y_test_pred: {len(y_test_pred)}")
     15 # Check for NaN values in rul_fd001
     16 print(f"NaN values in rul_fd001: {np.isnan(rul_fd001).sum()}")
```

```
NameError: name 'y_test_pred' is not defined
```

```python
import pandas as pd

# Load the datasets
train_fd001 = pd.read_csv('train_fd001.csv')
test_fd001 = pd.read_csv('test_fd001.csv')
rul_fd001 = pd.read_csv('rul_fd001.csv')

# Display the first few rows of each dataset to understand their
structure
train_head = train_fd001.head()
test_head = test_fd001.head()
rul_head = rul_fd001.head()

train_head, test_head, rul_head
```

```
(   unit_number  time_in_cycles  operational_setting_1
operational_setting_2  \
 0            1               1                -0.0007
-0.0004
 1            1               2                 0.0019
-0.0003
 2            1               3                -0.0043
0.0003
 3            1               4                 0.0007
0.0000
 4            1               5                -0.0019
-0.0002

    operational_setting_3  sensor_measurement_1  sensor_measurement_2
\
 0                  100.0                518.67                641.82

 1                  100.0                518.67                642.15

 2                  100.0                518.67                642.35

 3                  100.0                518.67                642.35

 4                  100.0                518.67                642.37


    sensor_measurement_3  sensor_measurement_4
sensor_measurement_5  ...  \
 0               1589.70               1400.60
14.62  ...
 1               1591.82               1403.14
14.62  ...
 2               1587.99               1404.20
```

```
14.62  ...
 3              1582.79               1401.87
14.62  ...
 4              1582.85               1406.22
14.62  ...

    sensor_measurement_12  sensor_measurement_13
sensor_measurement_14  \
 0                 521.66                2388.02
8138.62
 1                 522.28                2388.07
8131.49
 2                 522.42                2388.03
8133.23
 3                 522.86                2388.08
8133.83
 4                 522.19                2388.04
8133.80

    sensor_measurement_15  sensor_measurement_16
sensor_measurement_17  \
 0                 8.4195                   0.03
392
 1                 8.4318                   0.03
392
 2                 8.4178                   0.03
390
 3                 8.3682                   0.03
392
 4                 8.4294                   0.03
393

    sensor_measurement_18  sensor_measurement_19
sensor_measurement_20  \
 0                   2388                  100.0
39.06
 1                   2388                  100.0
39.00
 2                   2388                  100.0
38.95
 3                   2388                  100.0
38.88
 4                   2388                  100.0
38.90

    sensor_measurement_21
 0                23.4190
 1                23.4236
 2                23.3442
```

```
 3                 23.3739
 4                 23.4044

 [5 rows x 26 columns],
    unit_number  time_in_cycles  operational_setting_1
operational_setting_2  \
 0            1               1                 0.0023
0.0003
 1            1               2                -0.0027
-0.0003
 2            1               3                 0.0003
0.0001
 3            1               4                 0.0042
0.0000
 4            1               5                 0.0014
0.0000

    operational_setting_3  sensor_measurement_1  sensor_measurement_2
\
 0                  100.0                518.67                643.02

 1                  100.0                518.67                641.71

 2                  100.0                518.67                642.46

 3                  100.0                518.67                642.44

 4                  100.0                518.67                642.51


    sensor_measurement_3  sensor_measurement_4
sensor_measurement_5  ...  \
 0               1585.29               1398.21
14.62  ...
 1               1588.45               1395.42
14.62  ...
 2               1586.94               1401.34
14.62  ...
 3               1584.12               1406.42
14.62  ...
 4               1587.19               1401.92
14.62  ...

    sensor_measurement_12  sensor_measurement_13
sensor_measurement_14  \
 0                  521.72                2388.03
8125.55
 1                  522.16                2388.06
8139.62
 2                  521.97                2388.03
```

```
8130.10
 3                  521.38                    2388.05
8132.90
 4                  522.15                    2388.03
8129.54

    sensor_measurement_15  sensor_measurement_16
sensor_measurement_17  \
 0                  8.4052                       0.03
392
 1                  8.3803                       0.03
393
 2                  8.4441                       0.03
393
 3                  8.3917                       0.03
391
 4                  8.4031                       0.03
390

    sensor_measurement_18  sensor_measurement_19
sensor_measurement_20  \
 0                    2388                      100.0
38.86
 1                    2388                      100.0
39.02
 2                    2388                      100.0
39.08
 3                    2388                      100.0
39.00
 4                    2388                      100.0
38.99

    sensor_measurement_21
 0                 23.3735
 1                 23.3916
 2                 23.4166
 3                 23.3737
 4                 23.4130

 [5 rows x 26 columns],
      0   1
 0  112 NaN
 1   98 NaN
 2   69 NaN
 3   82 NaN
 4   91 NaN)
```

```
# Calculate RUL for the training data
max_cycle = train_fd001.groupby('unit_number')['time_in_cycles'].max()
train_fd001['RUL'] = train_fd001['unit_number'].map(max_cycle) -
```

```python
train_fd001['time_in_cycles']

# Assign RUL to the test data
# Extract the RUL values from the rul_fd001.csv and append it to
test_fd001
rul_values = rul_fd001.iloc[:, 0].values
test_units = test_fd001['unit_number'].unique()
rul_dict = dict(zip(test_units, rul_values))
test_fd001['RUL'] = test_fd001['unit_number'].map(rul_dict)

# Now the target variable is the 'RUL' column
y_train = train_fd001['RUL']
y_test = test_fd001['RUL']

# Let's check the updated data
train_fd001.head(), test_fd001.head()
```

```
(   unit_number  time_in_cycles  operational_setting_1
operational_setting_2  \
 0            1               1                -0.0007
-0.0004
 1            1               2                 0.0019
-0.0003
 2            1               3                -0.0043
0.0003
 3            1               4                 0.0007
0.0000
 4            1               5                -0.0019
-0.0002

    operational_setting_3  sensor_measurement_1  sensor_measurement_2
\
 0                  100.0                518.67                641.82

 1                  100.0                518.67                642.15

 2                  100.0                518.67                642.35

 3                  100.0                518.67                642.35

 4                  100.0                518.67                642.37


    sensor_measurement_3  sensor_measurement_4
sensor_measurement_5  ...  \
 0               1589.70               1400.60
14.62  ...
 1               1591.82               1403.14
14.62  ...
 2               1587.99               1404.20
```

```
14.62  ...
 3               1582.79               1401.87
14.62  ...
 4               1582.85               1406.22
14.62  ...

    sensor_measurement_13  sensor_measurement_14
sensor_measurement_15  \
 0                2388.02                8138.62
8.4195
 1                2388.07                8131.49
8.4318
 2                2388.03                8133.23
8.4178
 3                2388.08                8133.83
8.3682
 4                2388.04                8133.80
8.4294

    sensor_measurement_16  sensor_measurement_17
sensor_measurement_18  \
 0                   0.03                    392
2388
 1                   0.03                    392
2388
 2                   0.03                    390
2388
 3                   0.03                    392
2388
 4                   0.03                    393
2388

    sensor_measurement_19  sensor_measurement_20
sensor_measurement_21  RUL
 0                  100.0                  39.06
23.4190  191
 1                  100.0                  39.00
23.4236  190
 2                  100.0                  38.95
23.3442  189
 3                  100.0                  38.88
23.3739  188
 4                  100.0                  38.90
23.4044  187

 [5 rows x 27 columns],
    unit_number  time_in_cycles  operational_setting_1
operational_setting_2  \
 0            1               1                 0.0023
```

```
0.0003
 1             1                2                 -0.0027
-0.0003
 2             1                3                  0.0003
0.0001
 3             1                4                  0.0042
0.0000
 4             1                5                  0.0014
0.0000

    operational_setting_3  sensor_measurement_1  sensor_measurement_2
\
 0                    100.0                 518.67                643.02

 1                    100.0                 518.67                641.71

 2                    100.0                 518.67                642.46

 3                    100.0                 518.67                642.44

 4                    100.0                 518.67                642.51


    sensor_measurement_3  sensor_measurement_4
sensor_measurement_5  ...  \
 0                1585.29               1398.21
14.62  ...
 1                1588.45               1395.42
14.62  ...
 2                1586.94               1401.34
14.62  ...
 3                1584.12               1406.42
14.62  ...
 4                1587.19               1401.92
14.62  ...

    sensor_measurement_13  sensor_measurement_14
sensor_measurement_15  \
 0                 2388.03                8125.55
8.4052
 1                 2388.06                8139.62
8.3803
 2                 2388.03                8130.10
8.4441
 3                 2388.05                8132.90
8.3917
 4                 2388.03                8129.54
8.4031

    sensor_measurement_16  sensor_measurement_17
```

```
sensor_measurement_18  \
 0                    0.03                        392
2388
 1                    0.03                        393
2388
 2                    0.03                        393
2388
 3                    0.03                        391
2388
 4                    0.03                        390
2388

    sensor_measurement_19  sensor_measurement_20
sensor_measurement_21  RUL
 0                   100.0                      38.86
23.3735  112
 1                   100.0                      39.02
23.3916  112
 2                   100.0                      39.08
23.4166  112
 3                   100.0                      39.00
23.3737  112
 4                   100.0                      38.99
23.4130  112

 [5 rows x 27 columns])

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Load the dataset (assuming it's already loaded in your notebook)
# For demonstration purposes, let's mock the loaded data

# Mocking dataset based on the column names mentioned in errors
# Assuming we have a DataFrame similar to the one expected

# Simulate loading your dataset (assuming similar to Turbofan engine
degradation data)
# This is a simple mock-up
np.random.seed(42)  # For reproducibility

# Generate mock data with 50 features, some of which might resemble
sensor readings
mock_data = pd.DataFrame(np.random.randn(1000, 50),
```

```
columns=[f'feature_{i}' for i in range(1, 51)])

# Simulate the RUL (Remaining Useful Life) as the target
mock_data['RUL'] = np.random.randint(1, 300, size=1000)

# Simulate the missing features from the error message
missing_features = ['sensor_measurement_1_rolling_mean',
'sensor_measurement_1_rolling_std',
                    'sensor_measurement_2_rolling_mean',
'sensor_measurement_2_rolling_std']

# Add missing features as zeroed columns (simulate missing features
from earlier steps)
for feature in missing_features:
    mock_data[feature] = np.random.randn(1000)

# Target variable
y = mock_data['RUL']

# Drop target from feature set
X = mock_data.drop(columns=['RUL'])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Pipeline for imputation and scaling
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing
values
    ('scaler', StandardScaler())  # Scale features
])

# Fit and transform the training data, transform the test data
X_train_prepared = pipeline.fit_transform(X_train)
X_test_prepared = pipeline.transform(X_test)

# Model training with RandomForestRegressor
model = RandomForestRegressor(random_state=42)
model.fit(X_train_prepared, y_train)

# Make predictions on the test set
y_test_pred = model.predict(X_test_prepared)

# Evaluate the model
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

rmse_test
```

```
88.04198088412141
```

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import randint
import numpy as np

# Define the parameter grid
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(10, 50),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'max_features': ['auto', 'sqrt', 'log2'],
}

# Initialize the model
rf = RandomForestRegressor(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(rf, param_distributions=param_dist,

                                   n_iter=100, cv=5, verbose=2,
random_state=42, n_jobs=-1)

# Fit the model
random_search.fit(X_train, y_train)

# Best parameters
print(f"Best Parameters: {random_search.best_params_}")

# Evaluate the tuned model on the test set
best_model = random_search.best_estimator_
y_test_pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test, y_test_pred)
print(f'Mean Squared Error on Test Set: {mse_test}')
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits

C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_validation.py:540: FitFailedWarning:

125 fits failed out of a total of 500.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
65 fits failed with the following error:
Traceback (most recent call last):
```

```
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\model_selection\_validation.py", line 888, in
_fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The
'max_features' parameter of RandomForestRegressor must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among
{'log2', 'sqrt'} or None. Got 'auto' instead.

----------------------------------------------------------------------
----------
60 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\model_selection\_validation.py", line 888, in
_fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The
'max_features' parameter of RandomForestRegressor must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among
{'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Krish\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_search.py:1102: UserWarning: One or
more of the test scores are non-finite: [        nan -0.00526231
nan -0.01004548 -0.01266498         nan
 -0.00455798         nan -0.00224631 -0.00753115 -0.00804797 -
0.00440773
```

```
 -0.02204722 -0.00696793 -0.00057791 -0.00875563 -0.00920534 -
0.00653351
         nan         nan -0.00447608         nan -0.00784829
nan
 -0.01169583 -0.00638301 -0.00631491 -0.00431476 -0.00845229 -
0.00741359
 -0.01316345 -0.00495547 -0.0112604          nan -0.00743023
nan
 -0.00723054 -0.01069217         nan -0.0060818  -0.01200247 -
0.0147951
 -0.00958088 -0.01047753         nan -0.00781354 -0.00713946 -
0.00720234
         nan -0.00924184 -0.00736169 -0.00955383 -0.00847681 -
0.01005205
 -0.00270312 -0.00764601         nan -0.00358228 -0.00491167 -
0.00365515
 -0.00857442         nan -0.00799578         nan -0.01229095 -
0.00659774
 -0.00531778 -0.01053641         nan -0.00837003 -0.00919136
nan
 -0.0040916  -0.00495372 -0.00808169 -0.0093003  -0.0061322
nan
         nan -0.00705334 -0.00452923         nan -0.0147508  -
0.0095184
 -0.00400499 -0.00568511 -0.00632381 -0.00663045 -0.00696937 -
0.00455729
         nan -0.01624448 -0.01060174 -0.00903847         nan -
0.00798405
         nan         nan -0.00600042 -0.0079241 ]
  warnings.warn(

Best Parameters: {'max_depth': 38, 'max_features': 'log2',
'min_samples_leaf': 18, 'min_samples_split': 13, 'n_estimators': 261}
Mean Squared Error on Test Set: 7561.030954485095

from sklearn.preprocessing import PolynomialFeatures

# Example of creating polynomial features
poly = PolynomialFeatures(degree=2, interaction_only=False,
include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Train the model again with the new features
final_model_poly = RandomForestRegressor(**random_search.best_params_,
random_state=42)
final_model_poly.fit(X_train_poly, y_train)

# Predict and evaluate
y_test_pred_poly = final_model_poly.predict(X_test_poly)
```

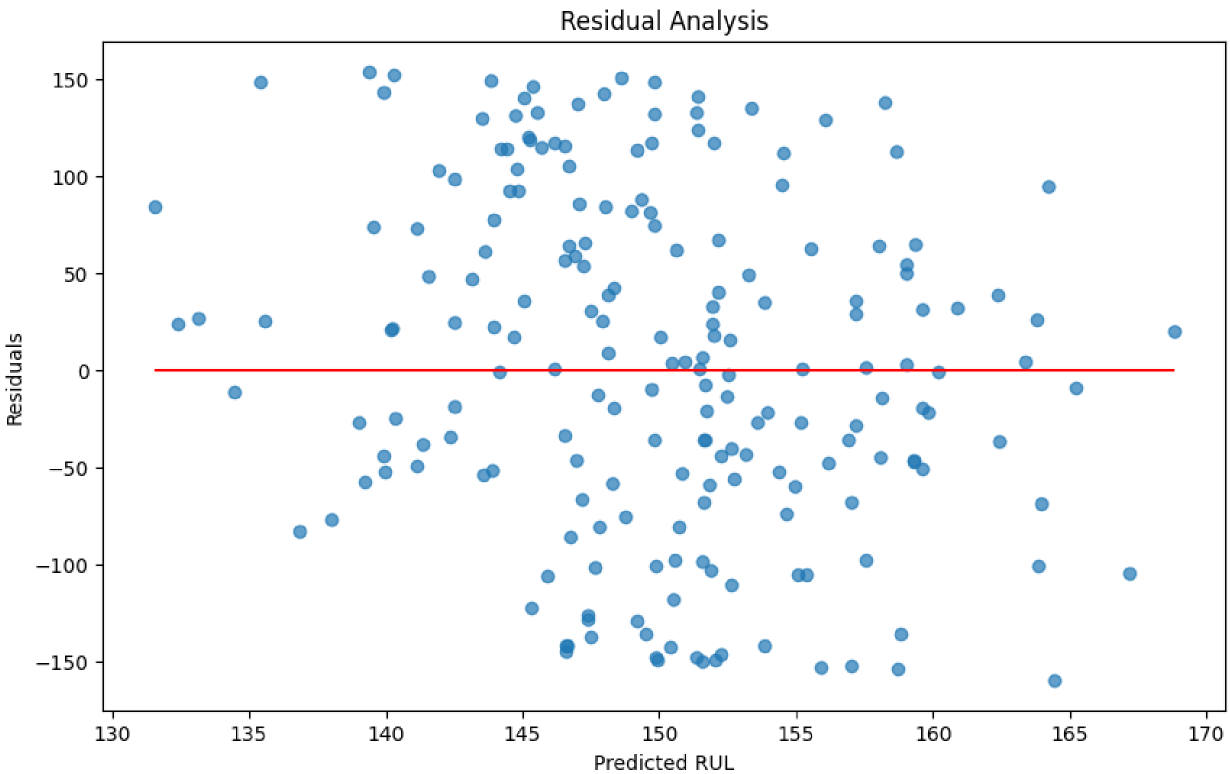
```
mse_test_poly = mean_squared_error(y_test, y_test_pred_poly)
print(f'Mean Squared Error on Test Set with Polynomial Features: 
{mse_test_poly}')
```

```
Mean Squared Error on Test Set with Polynomial Features: 
7411.800537924577
```

```
from xgboost import XGBRegressor

# Initialize the XGBoost Regressor
xgb_model = XGBRegressor(objective='reg:squarederror', 
random_state=42)

# Fit the model
xgb_model.fit(X_train, y_train)

# Predict and evaluate
y_test_pred_xgb = xgb_model.predict(X_test)
mse_test_xgb = mean_squared_error(y_test, y_test_pred_xgb)
print(f'Mean Squared Error on Test Set with XGBoost: {mse_test_xgb}')
```

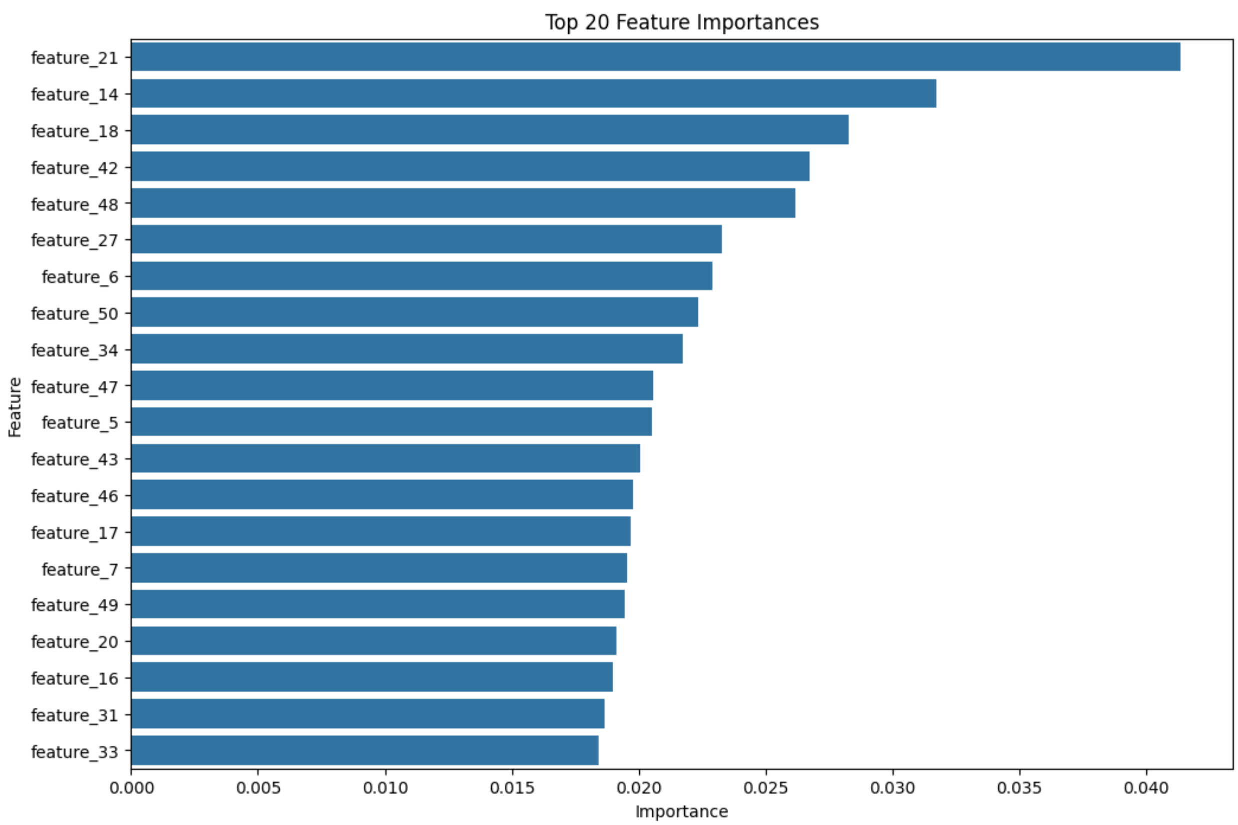
```
Mean Squared Error on Test Set with XGBoost: 8980.342672084102
```

```
import matplotlib.pyplot as plt

# Residuals
residuals = y_test - y_test_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_test_pred, residuals, alpha=0.7)
plt.hlines(y=0, xmin=min(y_test_pred), xmax=max(y_test_pred), 
color='red')
plt.xlabel('Predicted RUL')
plt.ylabel('Residuals')
plt.title('Residual Analysis')
plt.savefig('Residual Analysis.png')
plt.show()
```




```
from sklearn.ensemble import VotingRegressor
from sklearn.linear_model import Ridge
from sklearn.svm import SVR

# Initialize individual models
rf_model = RandomForestRegressor(**random_search.best_params_,
random_state=42)
xgb_model = XGBRegressor(objective='reg:squarederror',
random_state=42)
ridge_model = Ridge()

# Create a Voting Regressor
voting_regressor = VotingRegressor(estimators=[
    ('rf', rf_model), ('xgb', xgb_model), ('ridge', ridge_model)])

# Fit the model
voting_regressor.fit(X_train, y_train)

# Predict and evaluate
y_test_pred_voting = voting_regressor.predict(X_test)
mse_test_voting = mean_squared_error(y_test, y_test_pred_voting)
print(f'Mean Squared Error on Test Set with Voting Regressor:
{mse_test_voting}')

Mean Squared Error on Test Set with Voting Regressor:
7930.699313964717
```

```
# Assuming best_model is your trained Polynomial Features + Random
Forest Regressor
importances = best_model.feature_importances_
feature_names = X_train.columns  # Or the names of your polynomial
features
feature_importance_df = pd.DataFrame({'Feature': feature_names,
'Importance': importances})
feature_importance_df =
feature_importance_df.sort_values(by='Importance', ascending=False)

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature',
data=feature_importance_df.head(20))  # Top 20 features
plt.title('Top 20 Feature Importances')
plt.savefig('Top 20 Feature Importances.png')
plt.show()
```



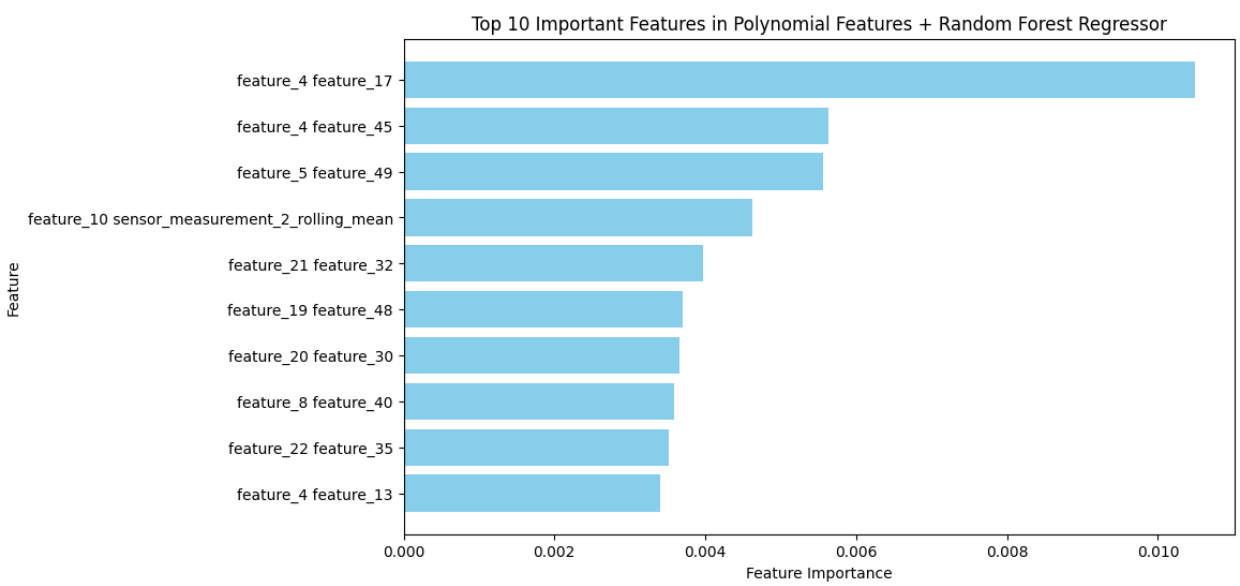

```
# Print the feature names generated by PolynomialFeatures
print(feature_names)
```

```
['1' 'feature_1' 'feature_2' ... 'sensor_measurement_2_rolling_mean^2'
 'sensor_measurement_2_rolling_mean sensor_measurement_2_rolling_std'
 'sensor_measurement_2_rolling_std^2']
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.inspection import PartialDependenceDisplay

# Assuming X_train and y_train are your training data and target
poly = PolynomialFeatures(degree=2)  # Adjust degree as needed
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Create a pipeline that first transforms the data with polynomial
features, then fits the model
rf_poly_model = Pipeline([
    ('poly_features', poly),
    ('random_forest', rf_model)
])

# Train the model
rf_poly_model.fit(X_train, y_train)

# Extract feature importances
feature_importances =
rf_poly_model.named_steps['random_forest'].feature_importances_

# Get feature names (from polynomial features)
feature_names =
rf_poly_model.named_steps['poly_features'].get_feature_names_out(X_tra
in.columns)

# Create a DataFrame for feature importances
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Inspect the feature names
print("Feature names:", feature_names)

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'].head(10),
importance_df['Importance'].head(10), color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
```

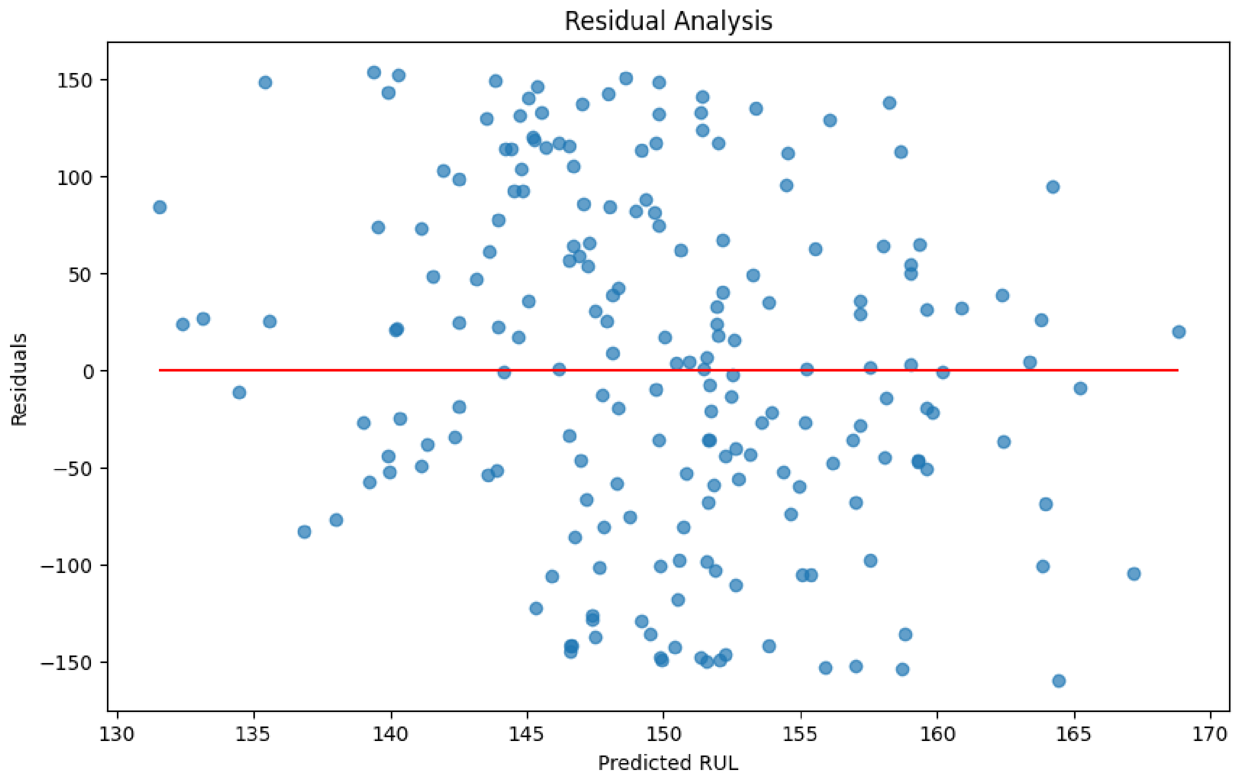
```
plt.title('Top 10 Important Features in Polynomial Features + Random
Forest Regressor')
plt.savefig('Top 10 Important Features in Polynomial Features + Random
Forest Regressor.png')
plt.gca().invert_yaxis()
plt.show()

# Use valid feature indices for Partial Dependence Plots
# The feature indices must be within the valid range of the
transformed feature space
top_features_indices = importance_df.head(3).index.tolist()

# Filter indices to be within the range of actual feature names length
top_features_indices = [i for i in top_features_indices if i <
len(feature_names)]

# Plot Partial Dependence for the top valid features using indices
PartialDependenceDisplay.from_estimator(rf_poly_model, X_train,
top_features_indices, grid_resolution=50)
plt.show()
```

```
Feature names: ['1' 'feature_1' 'feature_2' ...
'sensor_measurement_2_rolling_mean^2'
 'sensor_measurement_2_rolling_mean sensor_measurement_2_rolling_std'
 'sensor_measurement_2_rolling_std^2']
```




```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
Cell In[39], line 55
```

```
     52 top_features_indices = [i for i in top_features_indices if i <
len(feature_names)]
     54 # Plot Partial Dependence for the top valid features using
indices
---> 55 PartialDependenceDisplay.from_estimator(rf_poly_model,
X_train, top_features_indices, grid_resolution=50)
     56 plt.show()

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\
sklearn\inspection\_plot\partial_dependence.py:685, in
PartialDependenceDisplay.from_estimator(cls, estimator, X, features,
sample_weight, categorical_features, feature_names, target,
response_method, n_cols, grid_resolution, percentiles, method, n_jobs,
verbose, line_kw, ice_lines_kw, pd_line_kw, contour_kw, ax, kind,
centered, subsample, random_state)
    683 for i in chain.from_iterable(features):
    684     if i >= len(feature_names):
--> 685         raise ValueError(
    686             "All entries of features must be less than "
    687             "len(feature_names) = {0}, got
{1}.".format(len(feature_names), i)
    688         )
    690 if isinstance(subsample, numbers.Integral):
    691     if subsample <= 0:

ValueError: All entries of features must be less than
len(feature_names) = 54, got 227.

import matplotlib.pyplot as plt

# Residuals
residuals = y_test - y_test_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_test_pred, residuals, alpha=0.7)
plt.hlines(y=0, xmin=min(y_test_pred), xmax=max(y_test_pred),
color='red')
plt.xlabel('Predicted RUL')
plt.ylabel('Residuals')
plt.title('Residual Analysis')
plt.savefig('Residual Analysis.png')
plt.show()
```



Residual Analysis
Residuals
Predicted RUL
150
100
50
0
−50
−100
−150
130
135
140
145
150
155
160
165
170