Predictive Maintenance of Turbofan Jet Engine

Presented by: Shahril, Zhen Huo

Group 14

Introduction

NASA CMAPSS dataset

To predict the remaining useful life (RUL) of each engine in the test dataset

Enhancing the reliability, efficiency, and safety of aircraft operations.

How many times till failure?

Sensors to monitor the engine performance

Identifying Gaps in Available Code

Inadequate Data exploration

Limited preprocessing

Very Basic Feature engineering

Rolling windows

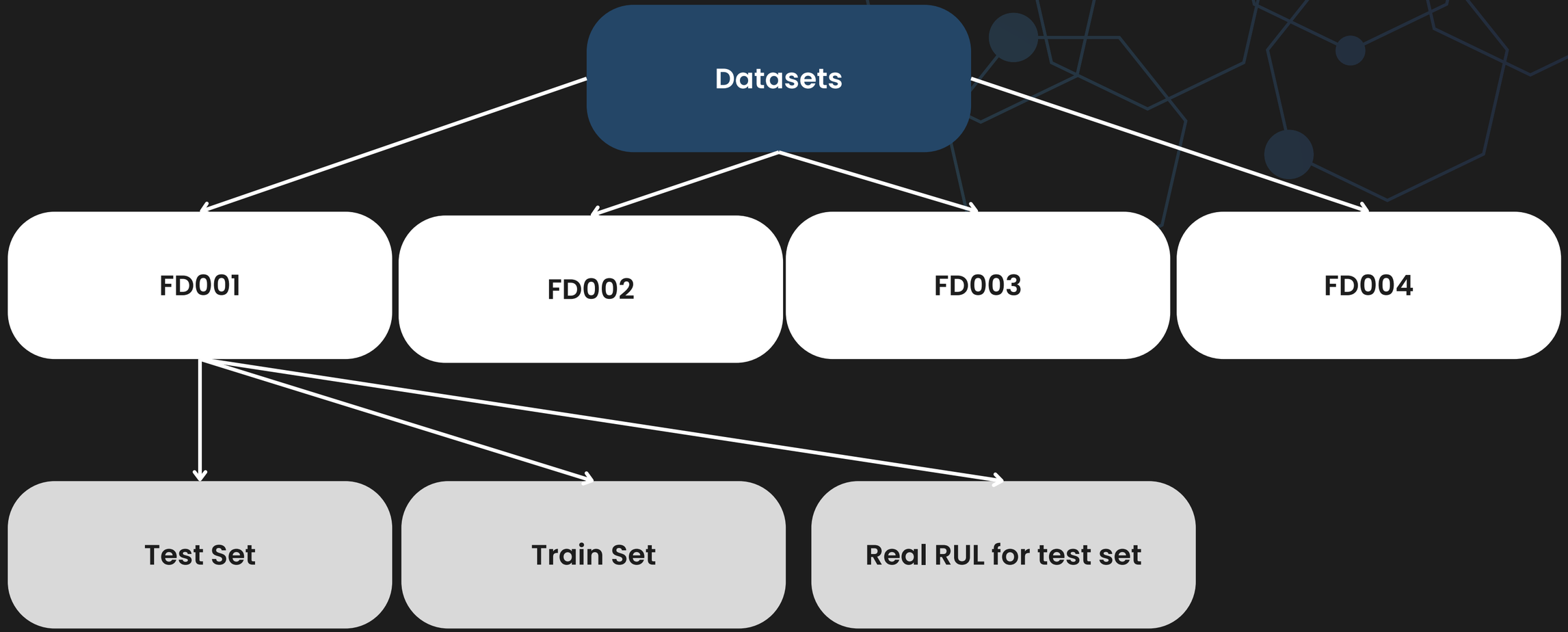
TSFresh

Lack of Hyperparatuning

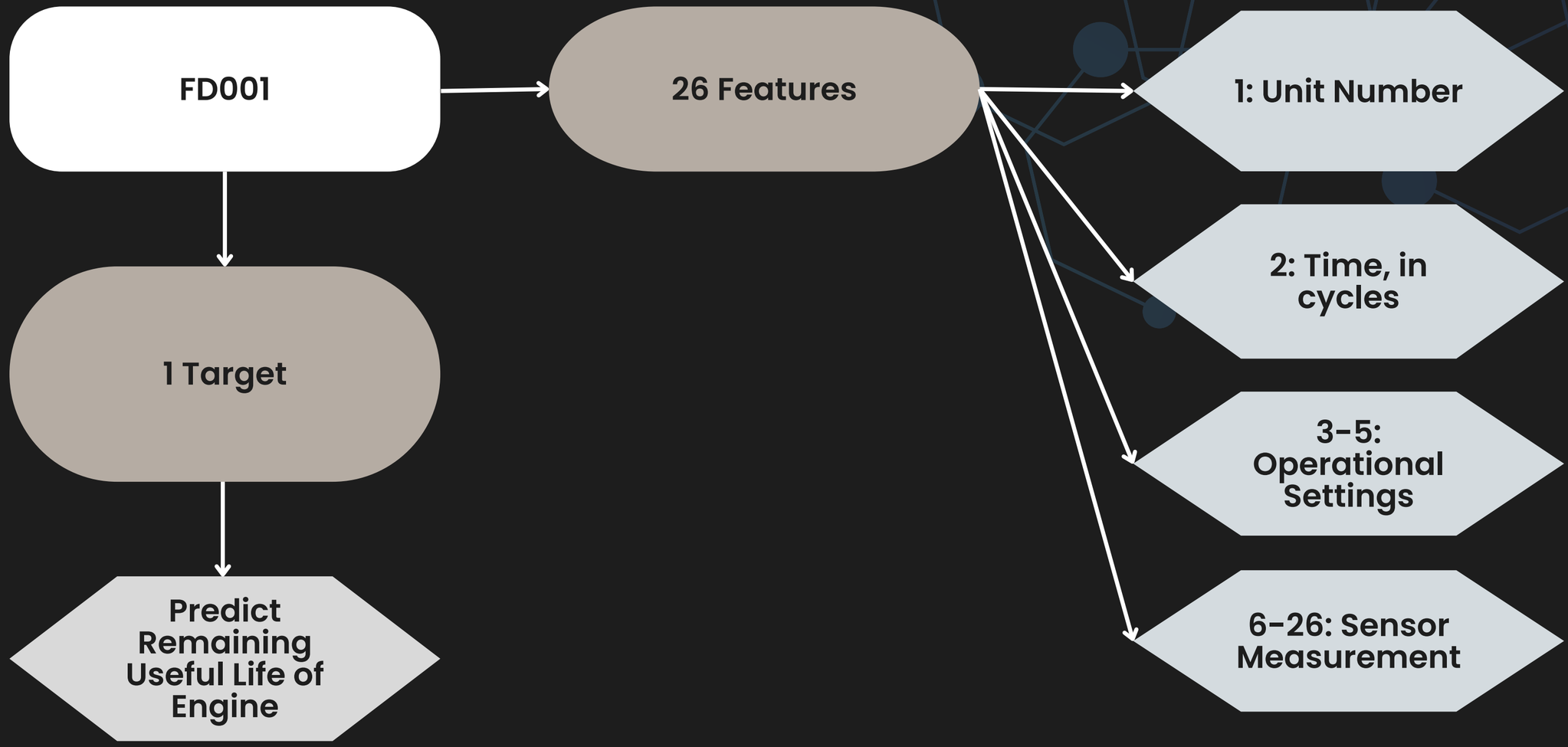
Limited Model exploration

NGBoost

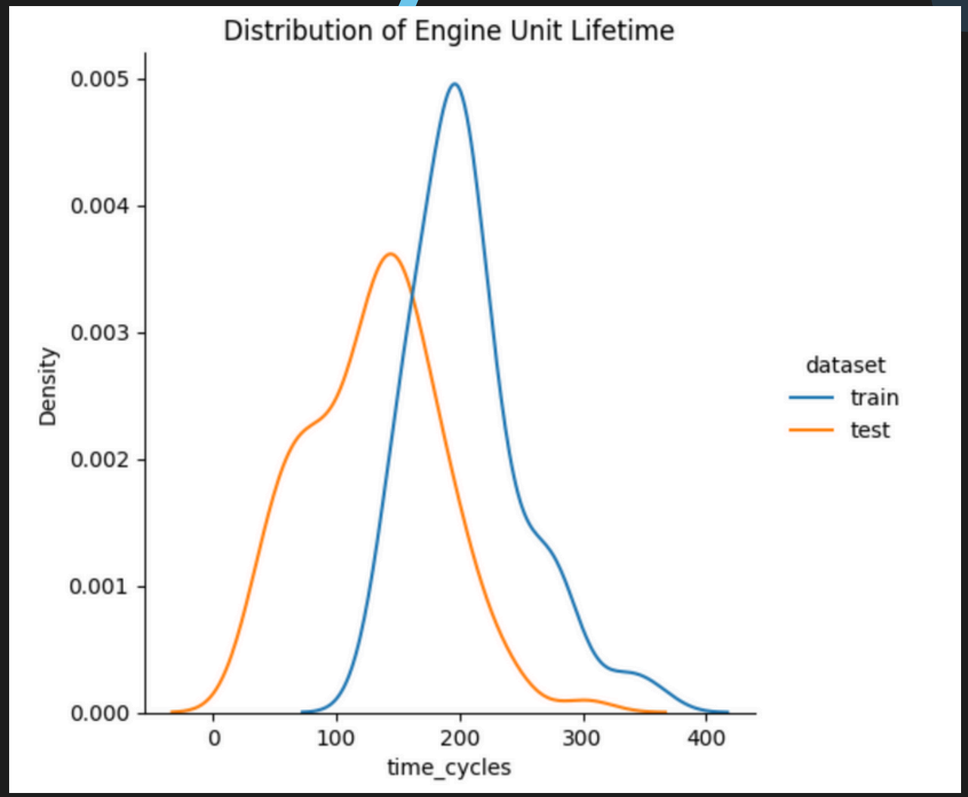
Dataset Description



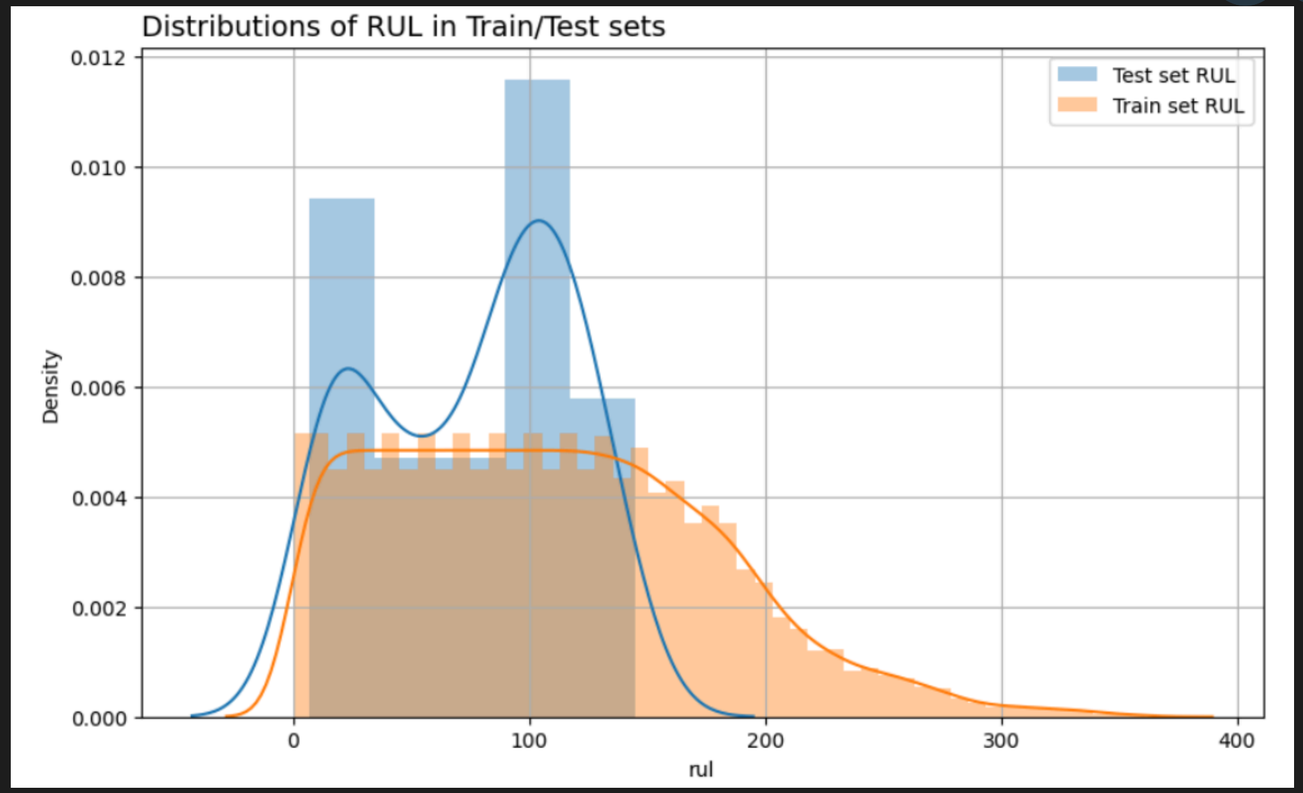
Dataset Description



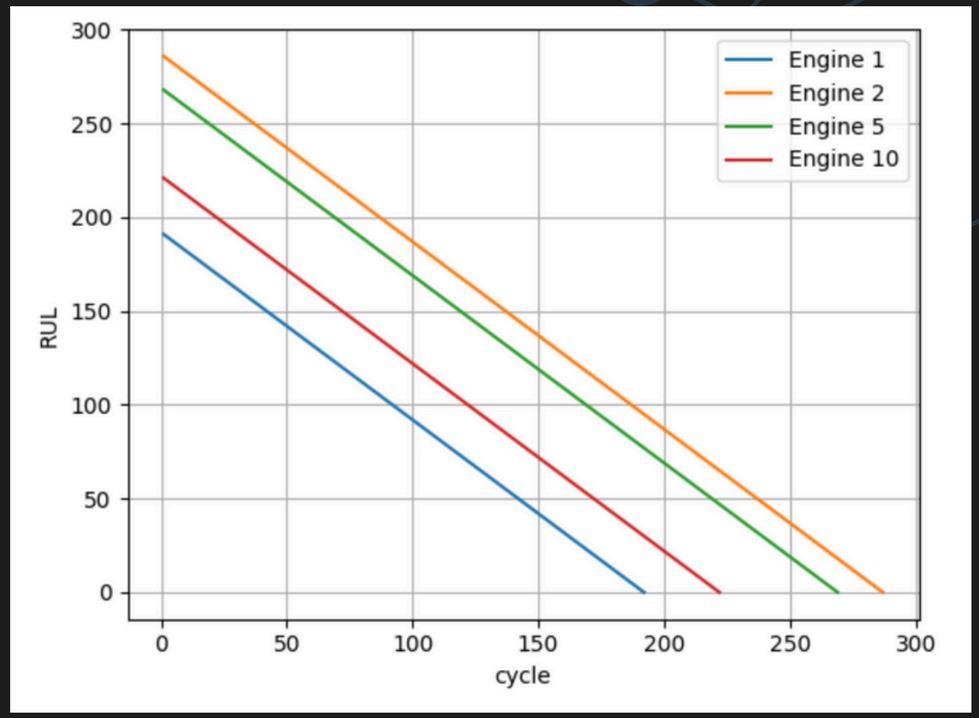
Dataset Description: Exploratory Data Analysis



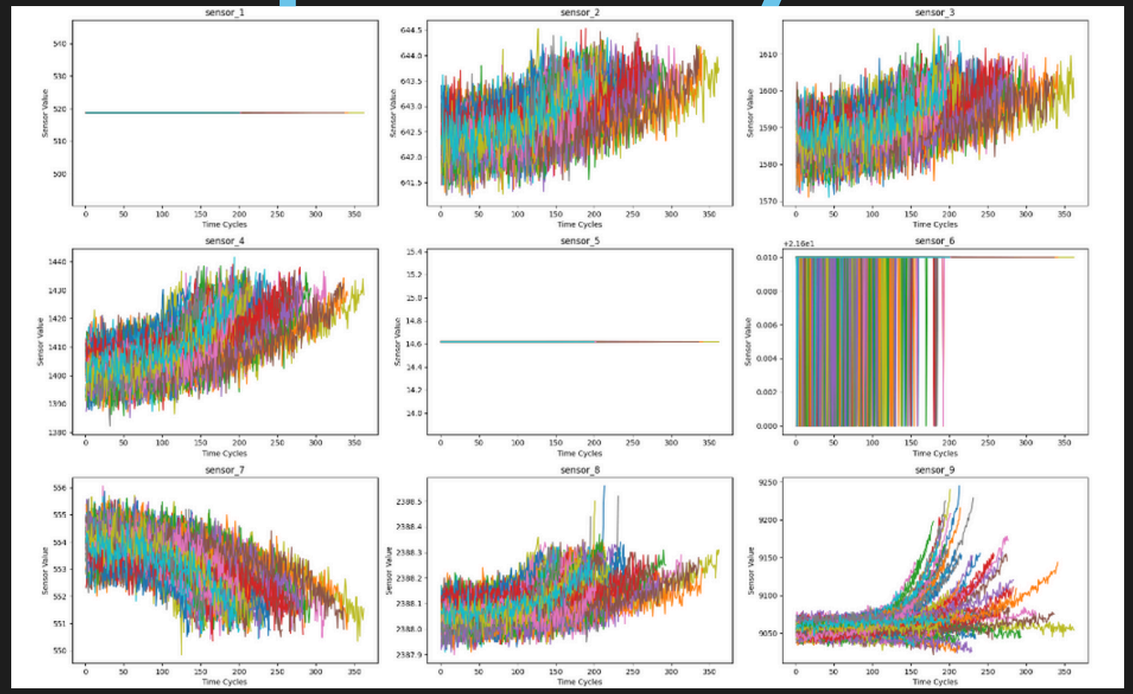
Dataset Description: Exploratory Data Analysis

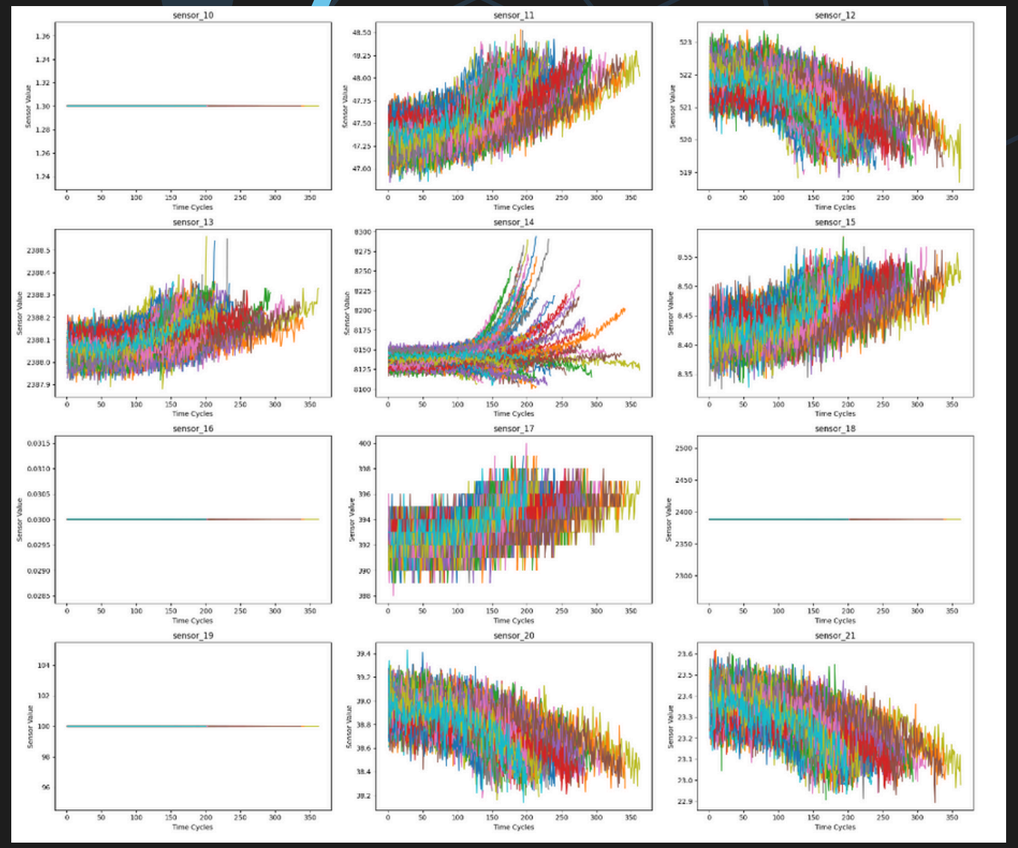


Dataset Description: Exploratory Data Analysis



Dataset Description: Exploratory Data Analysis

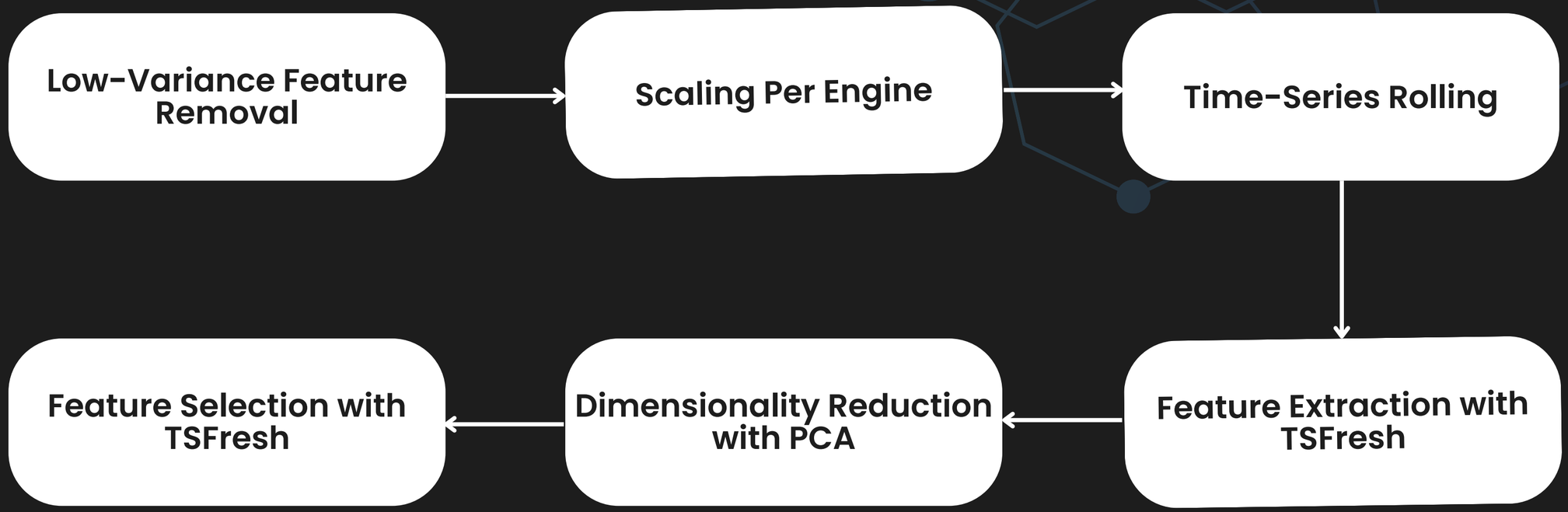




Wear/ Degration patterns

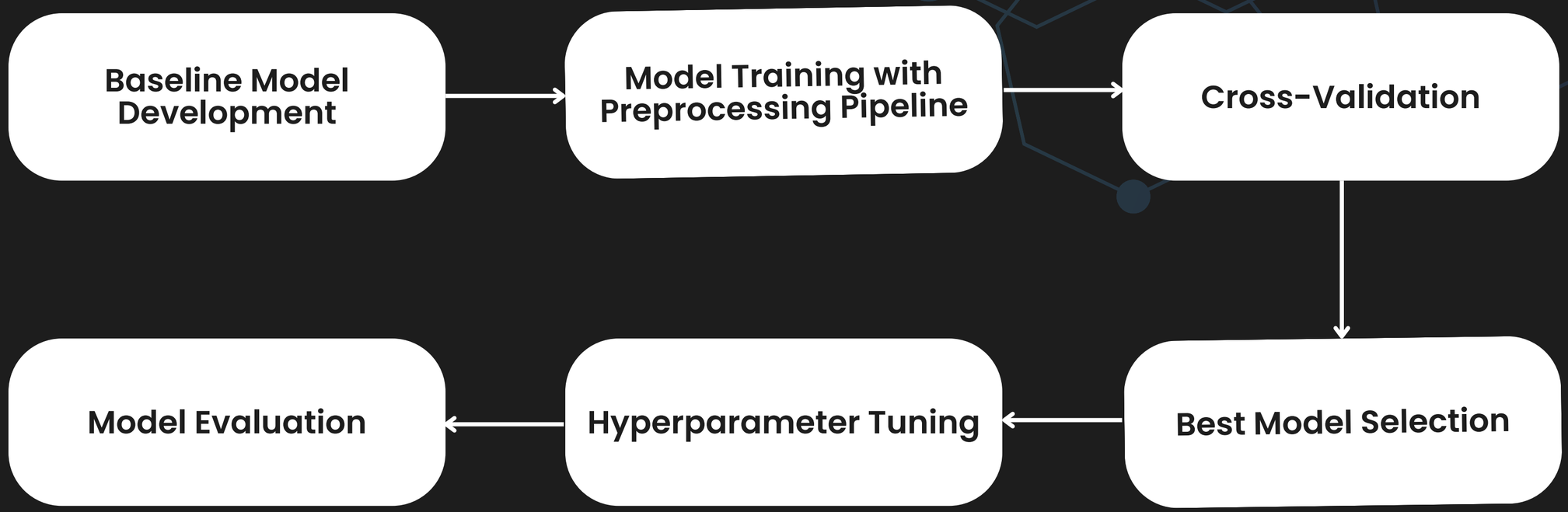
Methodology

1. Building Preprocessing Pipeline



Methodology

2. Model Training and Evaluation



```
rul              1.000000
sensor_11        0.769662
time_cycles      0.755796
sensor_4         0.751705
sensor_12        0.743360
sensor_7         0.727802
sensor_15        0.714847
sensor_21        0.702201
sensor_20        0.699407
sensor_17        0.675436
sensor_2         0.672701
sensor_3         0.649536
sensor_8         0.619891
sensor_13        0.619302
sensor_9         0.455894
sensor_14        0.364142
sensor_6         0.112056
unit             0.033918
op_setting_2     0.006521
op_setting_1     0.005232
op_setting_3          NaN
sensor_1              NaN
sensor_5              NaN
sensor_10             NaN
sensor_16             NaN
sensor_18             NaN
sensor_19             NaN
Name: rul, dtype: float64
```

Low-Variance Feature Removal

```python
class LowVarianceFeaturesRemover(BaseEstimator, TransformerMixin):
    def __init__(self, threshold=0):
        self.threshold = threshold
        self.selector = VarianceThreshold(threshold=threshold)

    def fit(self, X):
        self.selector.fit(X)
        return self

    def transform(self, X):
        X_t = self.selector.transform(X)
        droped_features = X.columns[~self.selector.get_support()]
        print(f"Dropped low variance features: {droped_features.to_list()}")
        return pd.DataFrame(X_t, columns=self.selector.get_feature_names_out())
```

```python
# Apply LowVarianceFeaturesRemover and list out remaining features
train = LowVarianceFeaturesRemover(threshold=0).fit_transform(train)
```

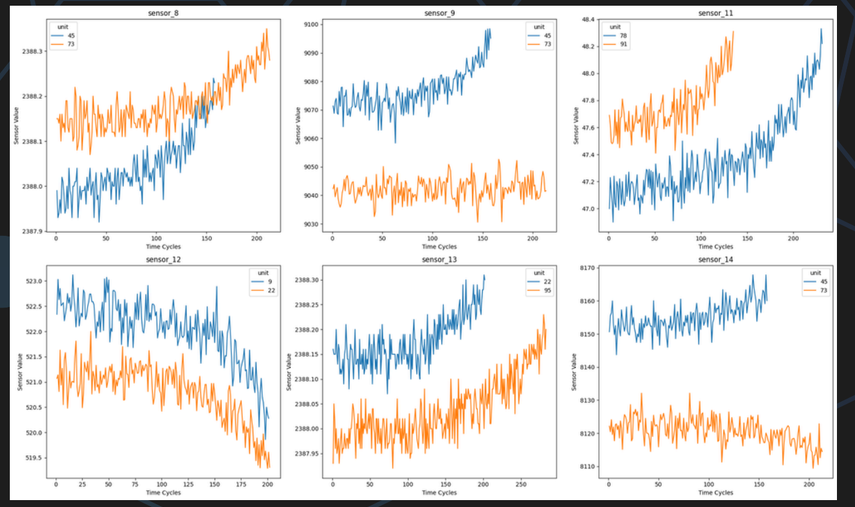
Dropped Features:

- op_setting_3
- sensor_1
- sensor_5
- sensor_10
- sensor_16
- sensor_18
- sensor_19

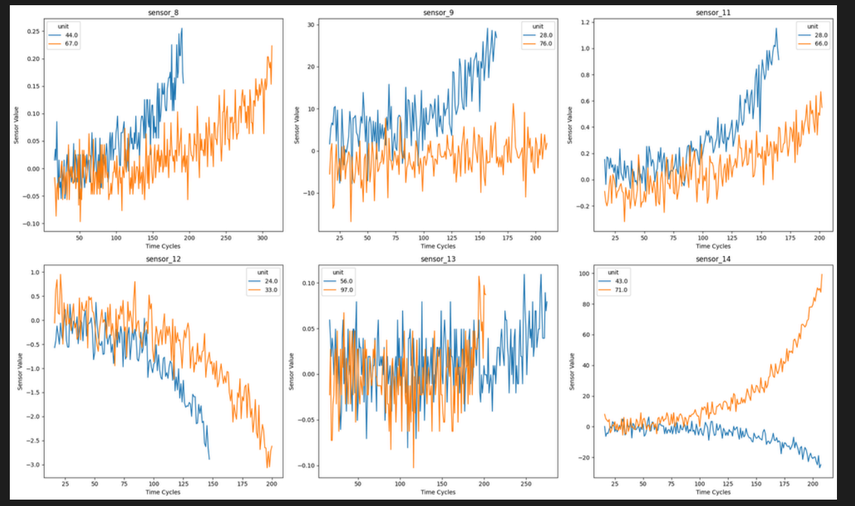
Scaling Per Engine

```python
class ScalePerEngine(BaseEstimator, TransformerMixin):
    def __init__(self, n_first_cycles=20, sensors_columns=SENSOR_COLUMNS):
        self.n_first_cycles = n_first_cycles
        self.sensors_columns = sensors_columns

    def fit(self, X):
        return self

    def transform(self, X):
        self.sensors_columns = [x for x in X.columns if x in self.sensors_columns]

        init_sensors_avg = (
            X[X["time_cycles"] <= self.n_first_cycles]
            .groupby(by=["unit"])[self.sensors_columns]
            .mean()
            .reset_index()
        )

        X_t = X[X["time_cycles"] > self.n_first_cycles].merge(
            init_sensors_avg, on=["unit"], how="left", suffixes=("", "_init_v")
        )

        for SENSOR in self.sensors_columns:
            X_t[SENSOR] = X_t[SENSOR] - X_t["{}_init_v".format(SENSOR)]

        drop_columns = X_t.columns.str.endswith("init_v")
        return X_t[X_t.columns[~drop_columns]]
train = ScalePerEngine(n_first_cycles=15, sensors_columns=SENSOR_COLUMNS).fit_transform(train)
```

Before scaling



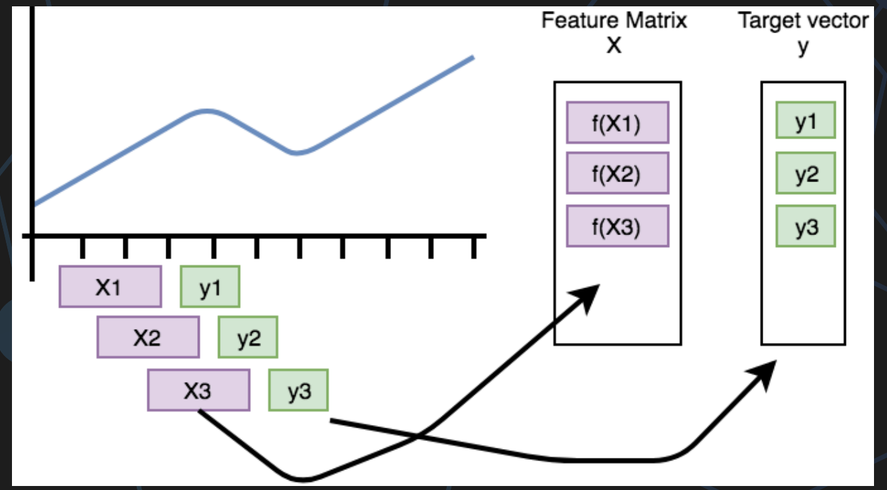
After scaling



Rolling Window

```python
from tsfresh.utilities.dataframe_functions import roll_time_series

class RollTimeSeries(BaseEstimator, TransformerMixin):
    def __init__(self, min_timeshift, max_timeshift, rolling_direction):
        self.min_timeshift = min_timeshift
        self.max_timeshift = max_timeshift
        self.rolling_direction = rolling_direction

    def fit(self, X):
        return self

    def transform(self, X):
        _start = datetime.now()
        print("Start Rolling TS")
        X_t = roll_time_series(
            X,
            column_id="unit",
            column_sort="time_cycles",
            rolling_direction=self.rolling_direction,
            min_timeshift=self.min_timeshift,
            max_timeshift=self.max_timeshift,
        )
        print(f"Done Rolling TS in {datetime.now() - _start}")
        return X_t
```

```python
train = RollTimeSeries(min_timeshift=29, max_timeshift=29, rolling_direction=1).fit_transform(train)
```



sensor_15	sensor_17	sensor_20	sensor_21	rul	id
-0.015947	0.133333	-0.026	0.071733	135.0	(1.0, 45.0)
0.044653	0.133333	-0.186	-0.051367	135.0	(1.0, 45.0)
-0.006747	0.133333	-0.106	0.015433	135.0	(1.0, 45.0)
0.022553	-0.866667	-0.196	-0.036867	135.0	(1.0, 45.0)
0.011453	0.133333	0.034	0.038733	135.0	(1.0, 45.0)

Feature Extraction using TSFresh

```python
tsfresh_calc = {
    "mean_change": None,
    "mean": None,
    "standard_deviation": None,
    "root_mean_square": None,
    "last_location_of_maximum": None,
    "first_location_of_maximum": None,
    "last_location_of_minimum": None,
    "first_location_of_minimum": None,
    "maximum": None,
    "minimum": None,
    "time_reversal_asymmetry_statistic": [{"lag": 1}, {"lag": 2}, {"lag": 3}],
    "c3": [{"lag": 1}, {"lag": 2}, {"lag": 3}],
    "cid_ce": [{"normalize": True}, {"normalize": False}],
    "autocorrelation": [
        {"lag": 0},
        {"lag": 1},
        {"lag": 2},
        {"lag": 3},
    ],
    "partial_autocorrelation": [
        {"lag": 0},
        {"lag": 1},
        {"lag": 2},
        {"lag": 3},
    ],
    "linear_trend": [{"attr": "intercept"}, {"attr": "slope"}, {"attr": "stderr"}],
    "augmented_dickey_fuller": [
        {"attr": "teststat"},
        {"attr": "pvalue"},
        {"attr": "usedlag"},
    ],
    "linear_trend_timewise": [{"attr": "intercept"}, {"attr": "slope"}],
    "lempel_ziv_complexity": [
        {"bins": 2},
        {"bins": 3},
        {"bins": 5},
        {"bins": 10},
        {"bins": 100},
    ],
    "permutation_entropy": [
        {"tau": 1, "dimension": 3},
        {"tau": 1, "dimension": 4},
        {"tau": 1, "dimension": 5},
        {"tau": 1, "dimension": 6},
        {"tau": 1, "dimension": 7},
    ],
    "fft_coefficient": [
        {"coeff": 0, "attr": "abs"},
        {"coeff": 1, "attr": "abs"},
        {"coeff": 2, "attr": "abs"},
        {"coeff": 3, "attr": "abs"},
        {"coeff": 4, "attr": "abs"},
        {"coeff": 5, "attr": "abs"},
        {"coeff": 6, "attr": "abs"},
        {"coeff": 7, "attr": "abs"},
        {"coeff": 8, "attr": "abs"},
        {"coeff": 9, "attr": "abs"},
        {"coeff": 10, "attr": "abs"},
    ],
    "fft_aggregated": [
        {"aggtype": "centroid"},
        {"aggtype": "variance"},
        {"aggtype": "skew"},
        {"aggtype": "kurtosis"},
    ],
}
```

```python
from tsfresh import extract_features


class TSFreshFeaturesExtractor(BaseEstimator, TransformerMixin):
    def __init__(self, calc=tsfresh_calc):
        self.calc = calc

    def _clean_features(self, X):
        old_shape = X.shape
        X_t = X.T.drop_duplicates().T
        print(f"Droped {old_shape[1] - X_t.shape[1]} duplicate features")

        old_shape = X_t.shape
        X_t = X_t.dropna(axis=1)
        print(f"Droped {old_shape[1] - X_t.shape[1]} features with NA values")
        return X_t

    def fit(self, X):
        return self

    def transform(self, X):
        _start = datetime.now()
        print("Start Extracting Features")
        X_t = extract_features(
            X[
                ["id", "time_cycles"]
                + X.columns[X.columns.str.startswith("sensor")].tolist()
            ],
            column_id="id",
            column_sort="time_cycles",
            default_fc_parameters=self.calc,
        )
        print(f"Done Extracting Features in {datetime.now() - _start}")
        X_t = self._clean_features(X_t)
        return X_t
```

```python
train = TSFreshFeaturesExtractor().fit_transform(train)
```

```
Start Extracting Features
Feature Extraction: 100%|██████████| 20/20 [06:16<00:00, 18.83s/it]
Done Extracting Features in 0:06:41.194612
Droped 19 duplicate features
Droped 14 features with NA values

Index(['sensor_2__mean_change', 'sensor_2__mean',
       'sensor_2__standard_deviation', 'sensor_2__root_mean_square',
       'sensor_2__last_location_of_maximum',
       'sensor_2__first_location_of_maximum',
       'sensor_2__last_location_of_minimum',
       'sensor_2__first_location_of_minimum', 'sensor_2__maximum',
       'sensor_2__minimum',
       ...
       'sensor_21__fft_coefficient__attr_"abs"__coeff_5',
       'sensor_21__fft_coefficient__attr_"abs"__coeff_6',
       'sensor_21__fft_coefficient__attr_"abs"__coeff_7',
       'sensor_21__fft_coefficient__attr_"abs"__coeff_8',
       'sensor_21__fft_coefficient__attr_"abs"__coeff_9',
       'sensor_21__fft_coefficient__attr_"abs"__coeff_10',
       'sensor_21__fft_aggregated__aggtype_"centroid"',
       'sensor_21__fft_aggregated__aggtype_"variance"',
       'sensor_21__fft_aggregated__aggtype_"skew"',
       'sensor_21__fft_aggregated__aggtype_"kurtosis"'],
      dtype='object', length=822)
```

Dimensionality Reduction using PCA

```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

class CustomPCA(BaseEstimator, TransformerMixin):
    def __init__(self, n_components=None, random_state=None):
        self.n_components = n_components
        self.random_state = random_state

    def fit(self, X):
        assert "unit" not in X.columns, "columns should be only features"
        self.ftr_columns = X.columns

        self.scaler = StandardScaler()
        X_sc = self.scaler.fit_transform(X[self.ftr_columns].values)

        self.pca = PCA(n_components=self.n_components, random_state=self.random_state)
        self.pca.fit_transform(X_sc)
        return self

    def transform(self, X):
        X_sc = self.scaler.transform(X[self.ftr_columns].values)
        X_pca = self.pca.transform(X_sc)
        return pd.DataFrame(X_pca, index=X.index)
```

```python
train = CustomPCA(n_components=40).fit_transform(train)
```

Feature Selection using TSFresh

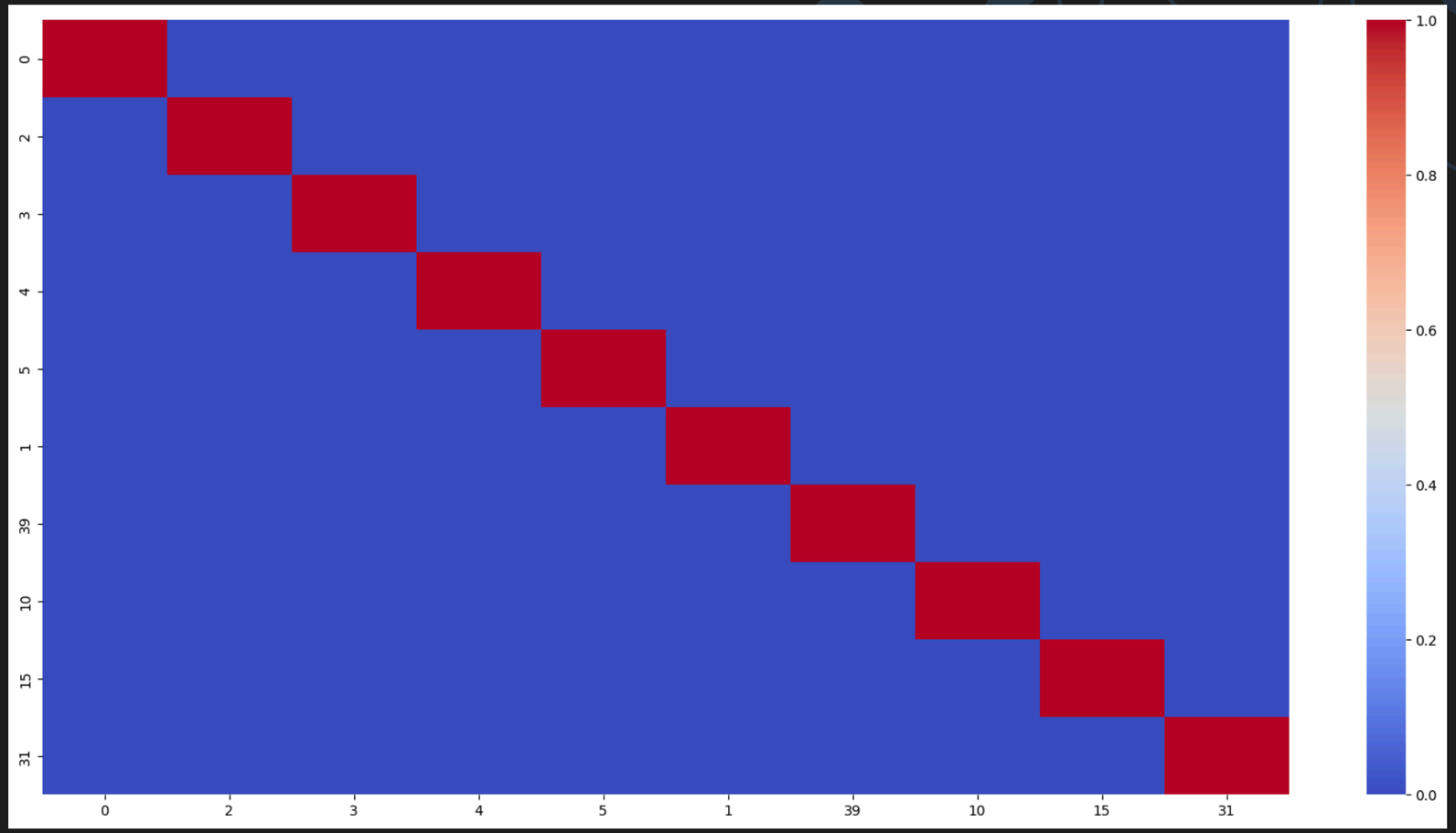
```python
from tsfresh import select_features

class TSFreshFeaturesSelector(BaseEstimator, TransformerMixin):
    def __init__(self, fdr_level=0.001):
        self.fdr_level = fdr_level

    def fit(self, X):
        rul = calculate_RUL(
            X.index.to_frame(name=["unit", "time_cycles"]).reset_index(drop=True),
            upper_threshold=135,
        )

        X_t = select_features(X, rul, fdr_level=self.fdr_level)
        self.selected_ftr = X_t.columns

        print(
            f"Selected {len(self.selected_ftr)} out of {X.shape[1]} features: "
            f"{self.selected_ftr.to_list()}"
        )
        return self

    def transform(self, X):
        return X[self.selected_ftr]
```

```python
train = TSFreshFeaturesSelector(fdr_level=0.001).fit_transform(train)
```

```
Selected 10 out of 40 features: [0, 2, 3, 4, 5, 1, 39, 10, 15, 31]
```

Results

Preprocessing Result



Results

Model Training and Cross Validation

No	Model	RMSE		MAE	
		Train	Validation	Train	Validation
1	Linear Regression (Baseline)	23.80	23.42	19.42	19.32
2	Linear Regression + Preprocessing	15.47	15.62	12.43	12.61
3	SVM + Preprocessing	14.54	15.82	10.71	12.23
4	Decision Tree + Preprocessing	0.00	19.45	0.00	14.97
5	Random Forest + Preprocessing	2.74	14.26	1.81	10.99
6	XGBoost + Preprocessing	4.79	14.55	3.37	11.12
7	NGBoost + Preprocessing	13.11	14.09	10.21	10.82

Results

Hyperparameter Tuning

```
param_grid = {
    'minibatch_frac': [1, 0.8, 0.5],
    'n_estimators': [100, 200, 300, 400],
    'learning_rate': [0.01, 0.05, 0.1],
}

ngb = NGBRegressor(Dist=Poisson)

grid_search = GridSearchCV(
    estimator=ngb,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    cv=CustomGroupKFold(n_splits=5),
    verbose=1
)

grid_search.fit(X_train.values, y_train, groups=train_units_df['unit'])
```

```
grid_search.best_params_
```

```
{'learning_rate': 0.05, 'minibatch_frac': 1, 'n_estimators': 300}
```

```
param_grid = {
    'Base': [
        DecisionTreeRegressor(criterion='friedman_mse',
                              max_depth=4),
        DecisionTreeRegressor(criterion='friedman_mse',
                              max_depth=5,
                              max_features=0.8,
                              min_samples_leaf=50),
    ]
}

ngb = NGBRegressor(Dist=Poisson,
                   n_estimators=grid_search.best_params_['n_estimators'],
                   learning_rate=grid_search.best_params_['learning_rate'],
                   minibatch_frac=grid_search.best_params_['minibatch_frac'])

grid_search = GridSearchCV(
    estimator=ngb,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    cv=CustomGroupKFold(n_splits=5),
    verbose=1
)

grid_search.fit(X_train.values, y_train, groups=train_units_df['unit'])
```

```
# find out the best base learner
grid_search.best_params_
```

```
{'Base': DecisionTreeRegressor(criterion='friedman_mse', max_depth=5, max_features=0.8,
                               min_samples_leaf=50)}
```

Results

Hyperparameter Tuning

```
ngb_base = DecisionTreeRegressor(criterion='friedman_mse', max_depth=5, max_features=0.8, min_samples_leaf=50)
ngb_tuned = NGBRegressor(Dist=Poisson, Base=ngb_base, n_estimators=300, learning_rate=0.05, minibatch_frac=1)

ngb_tuned_cv = evaluate(
    ngb_tuned,
    X=X_train.values,
    y=y_train,
    groups=train_units_df['unit'],
    cv=CustomGroupKFold(n_splits=5),
)
```

```
[test] :: root mean squared error : 13.62 +- 0.62
[train] :: root mean squared error : 10.80 +- 0.14
[test] :: mean absolute error : 10.40 +- 0.45
[train] :: mean absolute error : 7.92 +- 0.11
```

Train RMSE: 13.11 -> 10.80
Train MAE: 10.21 -> 7.92
CV RMSE: 14.09 -> 13.62
CV MAE: 10.82 -> 10.40

Results

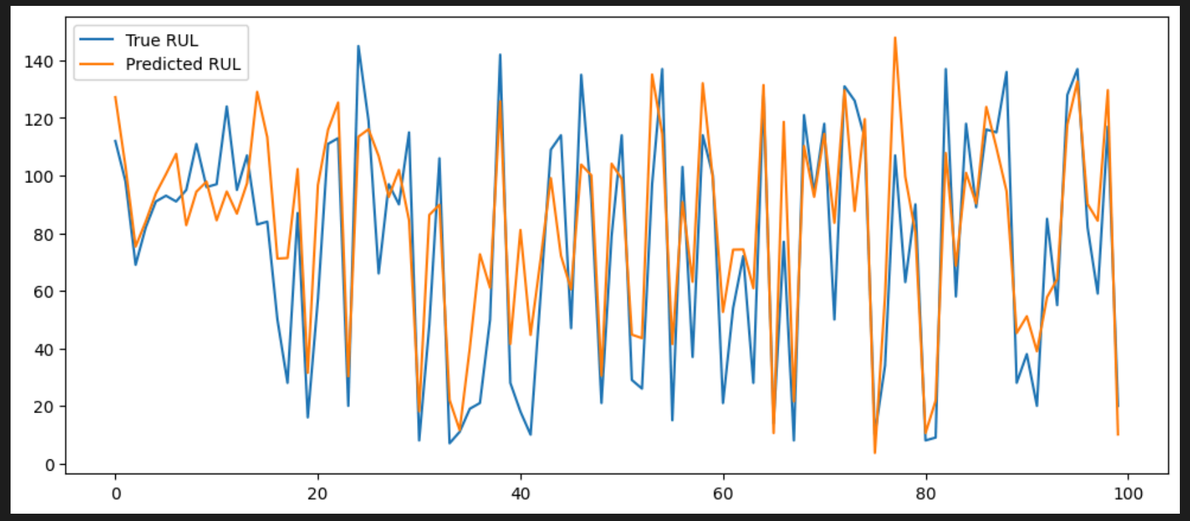
Model Testing

No	Model	Test RMSE
1	Linear Regression (Baseline)	22.37
2	LSTM (Ebrahim, 2022)	15.71
3	Random Forest (Kirstein, 2022)	17.96
4	XGBoost Regressor (Tyagi, 2020)	20.62
5	Tuned NGBoost Regressor (our best model)	13.19

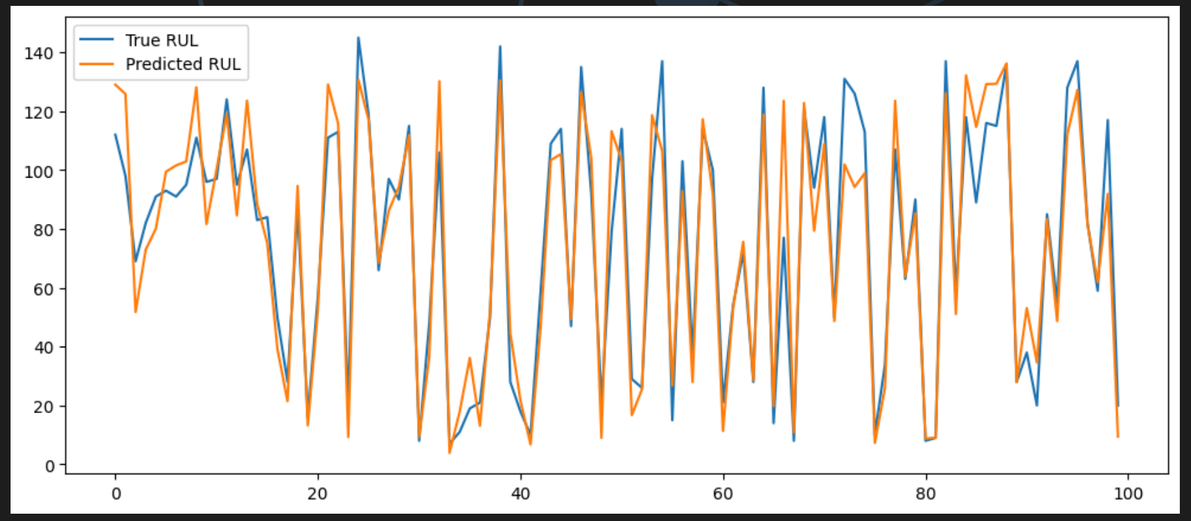
Results

Model Testing

Comparison of Predicted RUL and True RUL of each engine unit



Baseline Model



Tuned NGBoost Regressor

Conclusion

- We have successfully developed and evaluated ML models for predicting the Remaining Useful Life (RUL) of turbofan jet engines.
- Best model: NGBoost Regressor
- After hyperparameter tuning,
  - Test RMSE: 13.19
  - Test MAE: 10.14
- ~40% reduction in both RMSE and MAE when compared to the testing result of Baseline Model.
- With comprehensive EDA, robust preprocessing pipeline, and hyperparameter tuning, our best model outperforms models available in online repositories.

Contribution of Each Member

Shahril:

- Leader of Group 14. Scheduled meetings virtually or physically when needed.
- Performed exploratory data analysis (EDA) and model training on the chosen dataset.
- Responsible for Introduction, Methodology, Results, Discussions, and Conclusion in report writing.

Zhen Huo:

- Developed preprocessing pipeline.
- Prepared GitHub repository template for report writing.
- Responsible for Identifying Gaps in Available Code and Dataset Description in report writing.

THANK YOU!