RENEWIND

Data Analysis Observations & Recommendation

John D Noble

January 2022

Business Problem Overview & Solution Approach

This analysis is intended to use machine learning techniques to classify sensor data find the best one that will help identify failures so that the generator could be repaired before failing/breaking to reduce the maintenance cost.

Current State

- Renewable energy sources play an increasingly important role in the global energy mix, as the effort to reduce the environmental impact of energy production increases.
- Wind energy is one of the most developed technologies worldwide.

- The U.S Department of Energy has put together a guide to achieving operational efficiency using predictive maintenance practices.

a) Predictive maintenance uses sensor information and analysis methods to measure and predict degradation and future component capability

b) The idea is that failure patterns are predictable and if component failure can be predicted accurately and the component is replaced before it fails, the costs of operation and maintenance will be much lower.

What are we trying to solve for?

Can we leverage Machine Learning that can help in shortlisting the sensor data that likely to predict component degradation and failure.

Key Questions We Will Answer!

1. Do we have the proper data?
2. What modeling approaches will yield the best results?
3. Can we find the “right” sensor data and predict with a high degree of certainty which machines may need preventive maintenance before they break.

Financial Implications

The objective is to build various classification models, tune them, and find the best one that will help identify failures so that the generator could be repaired before failing/breaking to reduce the maintenance cost. The different costs associated with maintenance are as follows:

- Replacement cost = $40,000
- Repair cost = $15,000
- Inspection cost = $5,000

Data Description

Data Description

- ReneWind" has collected data of generator failure of wind turbines using sensors.
- They have shared a ciphered version of the data, as the data collected through sensors is confidential (the type of data collected varies with companies).

Data Description

1. The data provided is a transformed version of original data which was collected using sensors.
2. **Train.csv** - To be used for training and tuning of models.
3. **Test.csv** - To be used only for testing the performance of the final best model.
4. Both the datasets consist of 40 predictor variables and 1 target variable

Key Facts

We have a small data set for the analysis:

- Data has 40 predictors, 40000 observations in the training set, and 10000 in the test set.
- The shape of the "test" data
    - There are 10k columns and 41 variables
    - The data types are float64(40), int64(1)
- Missing data
    - Train is missing V1 – 46 / V2 39 entries
    - Test is missing v1 11 / v2 7 entries

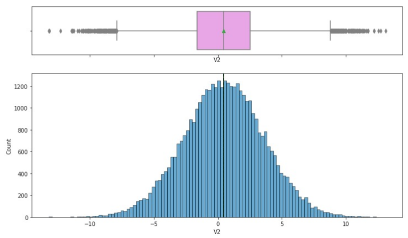
<u>Data Preperation Made Before Modeling</u>

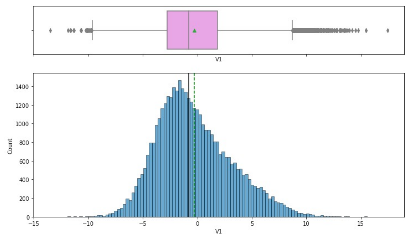
- we will use an imputer strategy to fill the missing data (imputer = SimpleImputer(strategy="median")

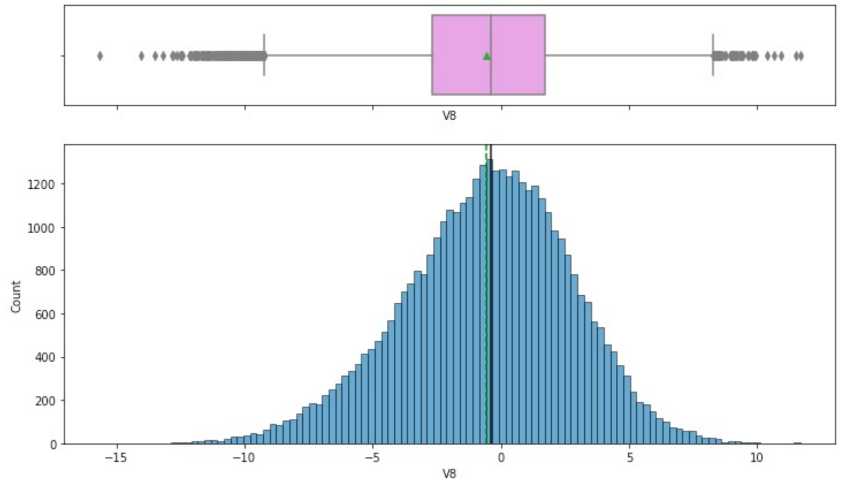
Data Description - EDA

All of the sensor data exhibit normal or slight skewed left/right data with outliers.

However, we are not going to treat the outliers because we believe they are valid data points and removing them would misrepresent the actual performance of the wind turbine components.



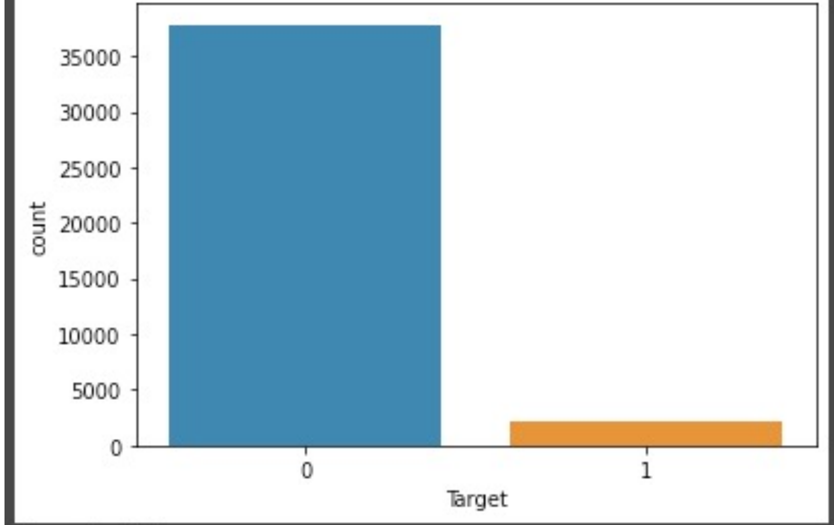




Exploratory Data Analysis (EDA)

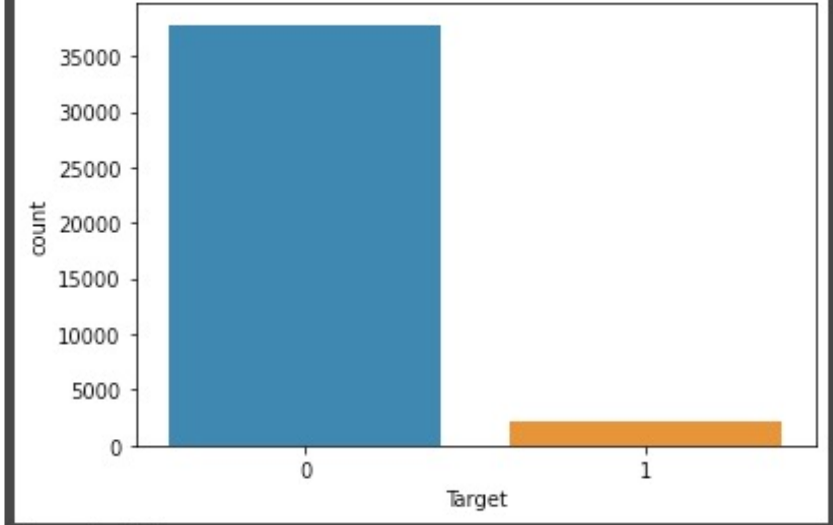
The train and test data sets are imbalanced. To address this we will use the train/validation split of the "train" data and use over/undersampling techniques with each model to see which yields better results.

The target class distribution for the train data



```
0    0.945
1    0.055
Name: Target, dtype: float64
```

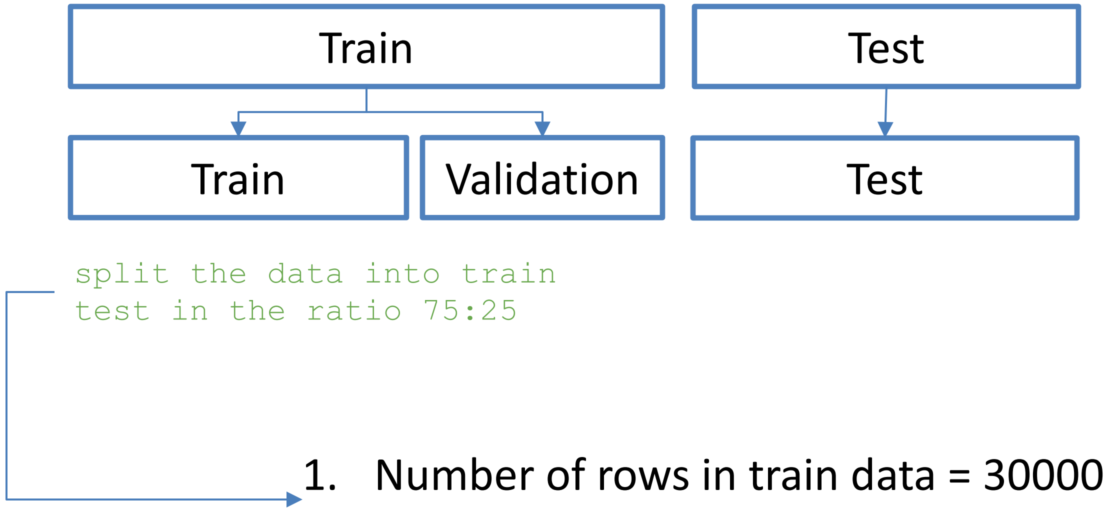
The target class distribution for the test data



```
0    0.945
1    0.055
Name: Target, dtype: float64
```

Data Preprocessing

The train and test data sets are not balanced. To address this we will use the train/validation split of the "train" data and use over/undersampling techniques with each model to see which yields better results



```
split the data into train
test in the ratio 75:25
```

1. Number of rows in train data = 30000
2. Number of rows in validation data = 10000
3. Number of rows in test data = 10000

- *Train V1 (46) and V2 (39) have missing values*
- *Test V1 (11) and V2 (7) have missing values*

How to Avoid Data Leakage

Data leakage happens when a certain part of the data is already seen in the training process.

1. *That's why it is always advised to keep the test dataset away and use it only for final evaluation.*
2. *When we impute the missing values for the entire data and then split the data into train-test then a certain part of the data is leaked in the training process. ***
3. *Hence, the best measure to avoid data leakage is to split the data into three sets.*

CONSIDERATIONS FOR HOW TO APPROACH MODELING

1. **Model evaluation criterion**
   - The nature of predictions made by the classification model will translate as follows:
   - True positives (TP) are failures correctly predicted by the model.
   - False negatives (FN) are real failures in a generator where there is no detection by model.
   - False positives (FP) are failure detections in a generator where there is no failure.

2. **Which metric to optimize?**
   - We would want Recall to be maximized as greater the Recall, the higher the chances of minimizing false negatives.
   - We want to minimize false negatives because if a model predicts that a machine will have no failure when there will be a failure, it will increase the maintenance cost.

3. The different costs associated with maintenance are as follows:
   - Replacement cost = $40,000
   - Repair cost = $15,000
   - Inspection cost = $5,000

Modeling Approaches

Model performance was evaluated on validation data set to assess if model generalizes well or is prone to overfitting/underfitting

We will approach the testing with some well-known and common modeling techniques:

1. BaggingClassifier
2. RandomForestClassifier
3. AdaBoostClassifier
4. GradientBoostingClassifier
5. DecisionTreeClassifier
6. LogisticRegression
7. XGBClassifier

For our model we will use the standard SKLearn method for splitting the data.

NOTE: because the data is imbalanced we are going to split the "train" data into 2 – train and validation for model training. The test data will only be used in the final prediction.

1. Number of rows in train data = 30000
2. Number of rows in validation data = 10000
3. Number of rows in test data = 10000

- "1" in the target variables should be considered as "failure" and "0" will represent "No failure".

Modeling – Ways to Address Imbalance in the Class Data

OverSampling

1. Before OverSampling, counts of label '1': 1640
2. Before OverSampling, counts of label '0': 28360
3. After OverSampling, counts of label '1': 28360
4. After OverSampling, counts of label '0': 28360
5. After OverSampling, the shape of train_X: (56720, 40)
6. After OverSampling, the shape of train_y: (56720,)

UnderSampling

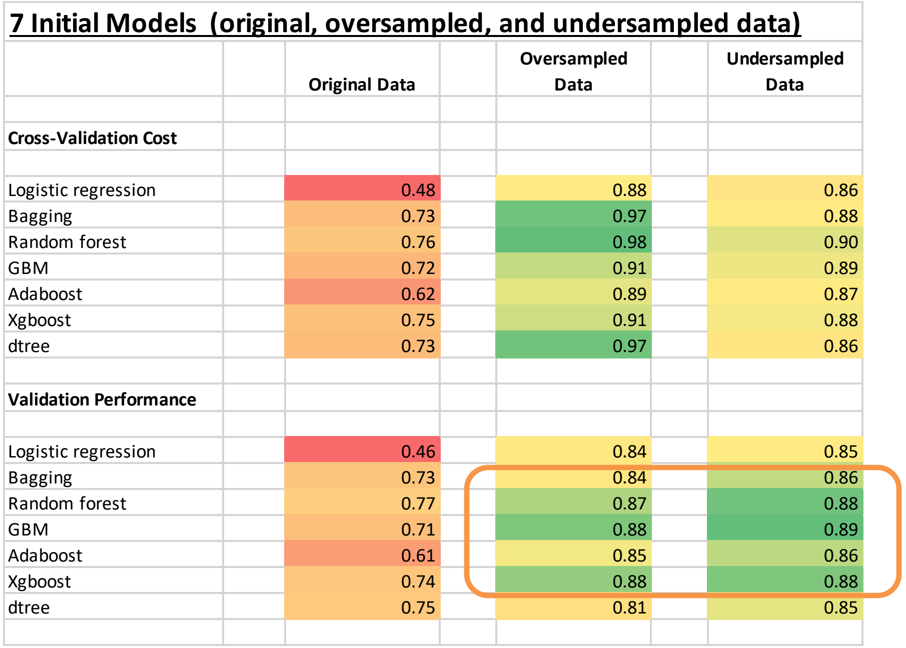
1. Before UnderSampling, counts of label '1': 1640
2. Before UnderSampling, counts of label '0': 28360
3. After UnderSampling, counts of label '1': 1640
4. After UnderSampling, counts of label '0': 1640
5. After UnderSampling, the shape of train_X: (3280, 40)
6. After UnderSampling, the shape of train_y: (3280,)

❖ Class imbalance was handled by
   1. Synthetic Minority Oversampling Technique (SMOTE)
   2. Random Under sampler & the 7 machine learning algorithms were fit again
   3. 5 fold cross validation was performed to calculate an average

Model Performance Summary

7 Separate Models run against the Original, Oversampled, and Undersampled

- For the “train” data set, performance was better on the Oversampled data compared to Undersampled or Original data.
- For the “validation” data set, the same general pattern held with slightly less performance suggesting there was overfitting on the “train” data aka the model learned too much noise and data.
- The “original” data performed poorly in both sets compared to the Over and Undersampling approaches which you would expect with imbalanced data sets.



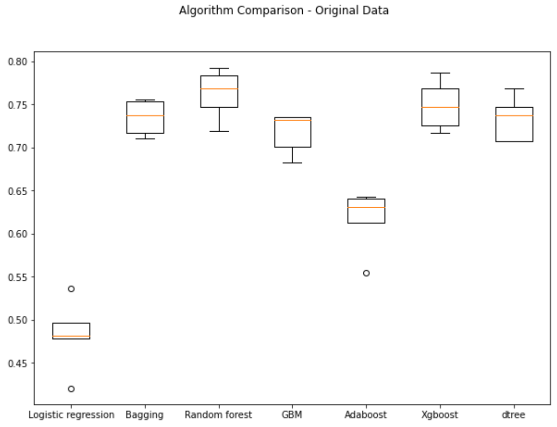
7 Initial Models (original, oversampled, and undersampled data)

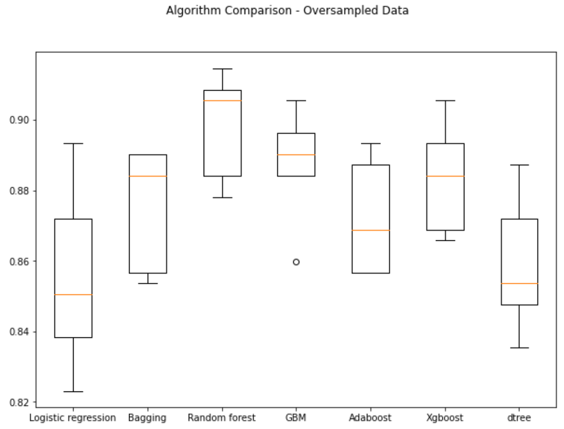
	Original Data	Oversampled Data	Undersampled Data
Cross-Validation Cost			
Logistic regression	0.48	0.88	0.86
Bagging	0.73	0.97	0.88
Random forest	0.76	0.98	0.90
GBM	0.72	0.91	0.89
Adaboost	0.62	0.89	0.87
Xgboost	0.75	0.91	0.88
dtree	0.73	0.97	0.86
Validation Performance			
Logistic regression	0.46	0.84	0.85
Bagging	0.73	0.84	0.86
Random forest	0.77	0.87	0.88
GBM	0.71	0.88	0.89
Adaboost	0.61	0.85	0.86
Xgboost	0.74	0.88	0.88
dtree	0.75	0.81	0.85

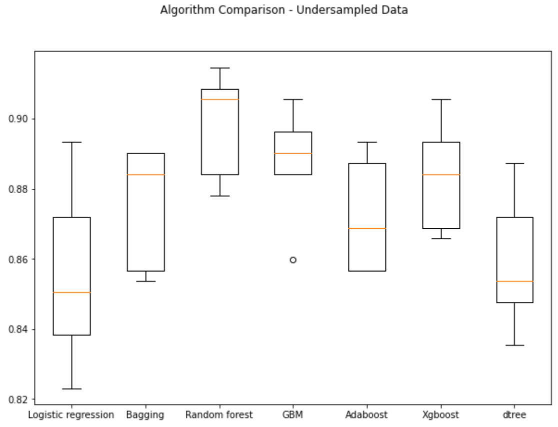
We will continue for with some of these models and **Hyperparameter Tuning.**

Model Performance Summary

Using boxplots, we can see the relative performance differences of each the models from the prior page. Note in a few cases, we can see outliers however as noted earlier we will leave these untreated as they are valid.







Hyperparameter Tuning Results – 4 Models

Which metric to optimize?

•We need to choose the metric which will ensure that the maximum number of generator failures are predicted correctly by the model.
•We would want Recall to be maximized as greater the Recall, the higher the chances of minimizing false negatives.
•We want to minimize false negatives because if a model predicts that a machine will have no failure when there will be a failure, it will increase the maintenance cost.

Model	Tuned Hyperparameter	Train - Recall	Validation - Recall
Tuning Bagging Classifier Using Oversampled Data	Best parameters are {'n_estimators': 50, 'max_samples': 1, 'max_features': 0.7} with CV score=1.0: CPU times: user 10.4 s, sys: 1.22 s, total: 11.6 s Wall time: 39min 47s	1	.85
Tuning Random Forest Using Oversampled Data	Best parameters are {'n_estimators': 250, 'min_samples_leaf': 1, 'max_samples': 0.6, 'max_features': 'sqrt'} with CV score=0.9780324400564175: CPU times: user 1min 10s, sys: 1.62 s, total: 1min 11s Wall time: 30min 37s	1	.87
Tuning Gradient Boosting Using Oversampled Data	Best parameters are {'subsample': 0.7, 'n_estimators': 125, 'max_features': 0.7, 'learning_rate': 1} with CV score=0.9571932299012694: CPU times: user 55.8 s, sys: 705 ms, total: 56.5 s Wall time: 20min 4s	.93	.88
Tuning Xgboost Using Oversampled Data	Best parameters are {'subsample': 0.9, 'scale_pos_weight': 10, 'n_estimators': 250, 'learning_rate': 0.2, 'gamma': 3} with CV score=0.9924541607898449: CPU times: user 41 s, sys: 1.66 s, total: 42.7 s Wall time: 37min 26s	.99	.91

Model Detail – After Hyperparameter Tuning

<u>Hyperparameter Tuning On 4 MODELS</u>

HyperParamter Tuning Perfromance - TRAIN

	Gradient Boosting tuned with oversampled data	XGBoost tuned with oversampled data	Bagging classifier tuned with oversampled data	Random forest tuned with oversampled data
Accuracy	0.95	0.97	1.00	1.00
Recall	0.93	0.99	1.00	1.00
Precision	0.98	0.95	1.00	1.00
F1	0.95	0.97	1.00	1.00

HyperParamter Tuning Perfromance - VAL

	Gradient Boosting tuned with oversampled data	XGBoost tuned with oversampled data	Bagging classifier tuned with oversampled data	Random forest tuned with oversampled data
Accuracy	0.97	0.94	0.99	0.99
Recall	0.88	0.91	0.85	0.87
Precision	0.71	0.46	0.88	0.95
F1	0.79	0.61	0.87	0.91

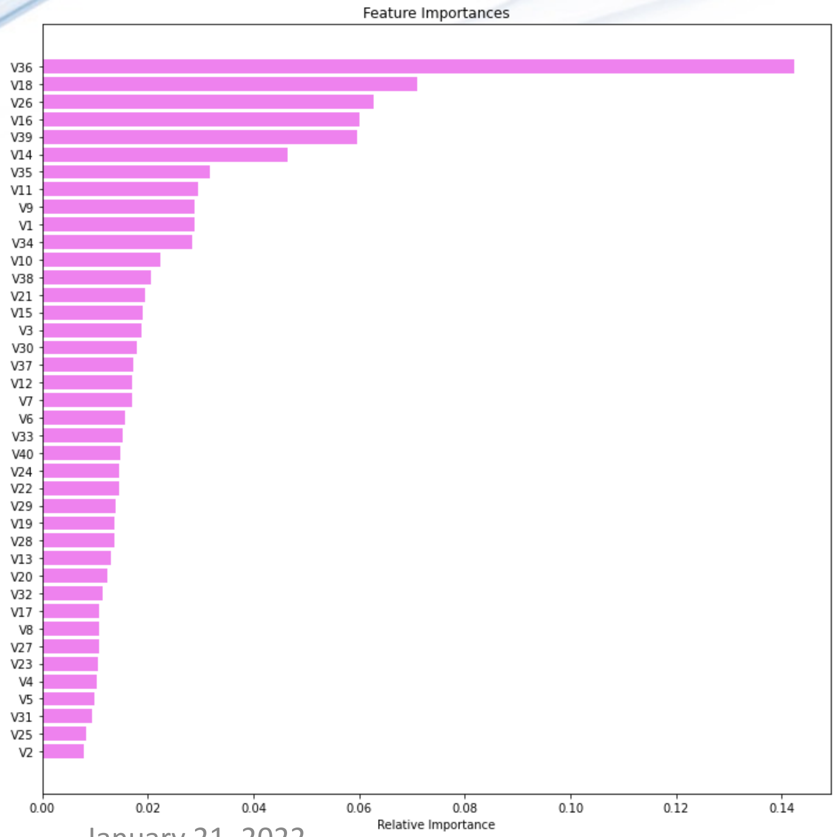
🔼 **XGBoost tuned with oversampled data Works Best!**

Final Model - Test Data

		XGBoost tuned with oversampled data
Accuracy		0.93
Recall		0.89
Precision		0.44
F1		0.58

🔼 **XGBoost tuned with oversampled data Works Best!**

Model Feature Importance



Result Interpretation

This graph shows the top features when it comes to the XGBoost and the impact of those features on successfully predicating whether a sensor

V36, V18, V26, V16, V39, and V14 are the most important features.

Build Final Production Pipeline

Result = The model generalizes well on unseen data!

1) Create Pipeline

```
# Creating new pipeline with best parameters
model = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="median")),
        (
            "XGBClassifier",
            XGBClassifier(
    random_state=1,
    eval_metric="logloss",
    subsample=0.9,
    scale_pos_weight=5,
    n_estimators=250,
    learning_rate=0.2,
    gamma=0,
),
        ),
    ]
)
```

2) Separate target from Train & Test

```
# Separating target variable and other variables
X1 = data.drop(columns="Target")
Y1 = data["Target"]

X_test1 = df_test.drop(columns="Target")
y_test1 = df_test["Target"]
```

3) Impute Missing Values

```
imputer = SimpleImputer(strategy="median")
X1 = imputer.fit_transform(X1)
|
X2 = imputer.fit_transform(X_test1)
```

3) Synthetic Minority Over Sampling Technique

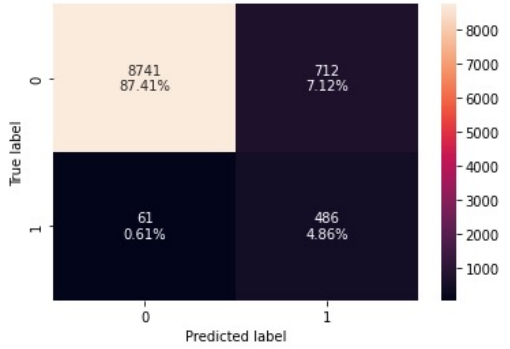
```
# Synthetic Minority Over Sampling Technique
sm = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=1)
X_over1, y_over1 = sm.fit_resample(X1, Y1)
```

4/5/6) Fit, Score, Test Performance

Test performance:

	Accuracy	Recall	Precision	F1
0	0.923	0.888	0.406	0.557

7) Confusion Matrix



Business Insights and Recommendations

1. We know we can build a machine learning model to minimize the total maintenance cost of machinery/processes used for wind energy production
   - The final tuned model (XGBoost) was chosen after building many machine learning algorithms and optimizing those in an imbalanced data set.
   - We fined tuned the model discrete hyper parameters and used a cross validation techniques to ha e confidence in the results.
   - **We production version of the pipeline was created, and found the model generalizes well in production!**
2. We can use this predictive model for any wind generator assuming we know the data about them in advance and with that information predict the likelihood of a malfunction (real or not).
   - This graph shows the top features when it comes to the XGBoost and the impact of those features on successfully predicating whether a sensor
   - V36, V18, V26, V16, V39, and V14 are the most important features.
3. The data is cyphered, so we really don't know the exact nature in a failure of sensor V36 for example. Additional discussions and the proper NDA should be put in place to generate more value added insight … e.g. maybe it will tell us that a certain part manufactured in one vs another has a highe incident in failure rate.
   - This added knowledge can be used to refine the process of collecting more frequent sensor information to be used in improving the machine learning model to further decrease maintenance costs
   - That would lead to changing the part sourcing strategy.
4. We can understand the real cost associated with the model and further work should be done to track the financial results:
   1. True positives (TP) are failures correctly predicted by the model.
   2. False negatives (FN) are real failures in the generator of wind turbine where there is no detection by the model.
   3. False positives (FP) are failure detections in the generator of the wind turbine where actually there is no failure.
   4. Maintenance cost = TP*(Repair cost) + FN*(Replacement cost) + FP*(Inspection cost)