ReneWind – Project 6

Model Tuning

February 17, 2023

Contents / Agenda

Executive Summary

- The model built can be used for ***ReneWind*** to predict occurrence of failures in a generators ***~87% +/- 2%***.

- The model has been **optimized to reduce** the likelihood that real failures in a generator go undetected (i.e. false negatives. It is possible that this may increase the number of inspections, but the **cost of inspections will likely be offset by the reduced number of replacements.**

- Of the forty features analysed, the **most importain factors** influenceing the prediction of failures are **(1) V36** (~10% R.I.) and **(2) V8,** (~10% R.I.) followed by **(3) V39**, **(4) V15**, and **(5) V3.** These ***Top Five*** account for nearly **37%** of the relative importance.

- In addition to the grouping of the ***Top Five*** , the ***Middle Nine*** has similar relative importance within the group, and the ***Remaining 26*** share similar relative importance.

- ***ReneWind*** may want to investigate commonalities within these groups to try to identify more granular information. For example, if one of these groups is all related to the **(1) environmental factors** (temperature, humidity, wind speed, etc.) or **(2) parts of the wind turbine** (gearbox, tower, blades, break, etc.), then it may be useful to isolate those groups and do separate studies.

- ***ReneWind*** would also benefit from **another iteration of data modeling.**

Business Problem Overview – Context

- **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.
- Out of all the renewable energy alternatives, **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.
- **Predictive maintenance** uses **sensor** information and analysis methods to **measure** and **predict degradation** and **future component capability**. **The idea behind predictive maintenance** 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 (**O&M**) will be much **lower**.
- The **sensors** fitted across different machines involved in the process of energy generation **collect data** related to various **(1) environmental factors** (temperature, humidity, wind speed, etc.) and additional features related to various **(2) parts of the wind turbine** (gearbox, tower, blades, break, etc.).

Business Problem Overview – Objective

- ***"ReneWind"*** is a company working on **improving the machinery/processes** involved in the production of **wind energy** using machine learning and 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 has **40 predictor variables**, and **one target variable**. The **target variable values** should be considered to represent **"1" as "failure"** and **"0" as "No failure".**
- The objective is to
  - **(1) build** various **classification models**,
  - **(2) tune** them, and
  - **(3) find the best one** that will help **identify failures** ...so that the generators could be **repaired before failing/breaking** to **reduce overall maintenance cost**. (i.e. costs of **inspection**, **repair**, **replacement**).

Business Problem Overview – Objective – Costs & Predictions

- Objective --- reduce overall maintenance cost (i.e. costs of **inspection**, **repair**, **replacement**).
- The nature of predictions made by the classification model will translate as follows:
  - **True positives** (TP) are failures correctly predicted by the model. These will result in **repair costs**.
  - **False negatives** (FN) are real failures where there is no detection by the model. These will result in **replacement costs**.
  - **False positives** (FP) are detections where there is no failure. These will result in **inspection costs**.
- It is given that the **cost of repairing ($$)** a generator is much less than the **cost of replacing($$$$)** it, and the **cost of the inspection ($)** is less than the **cost of repair ($$)**.
- Again, “1” in the target variables should be considered as “failure” and “0” represents “No failure”.

Model Building – Model evaluation criterion

- **Both the datasets consist of 40 predictor variables and 1 target variable**
- **“1”** in the **target variables** considered as **“failure”** and **“0”** represents **“No failure”.**
- The nature of **predictions** made by the **classification model** will translate as follows :

- **True positives (TP)** are **real failures “1” AND** model predicted correctly **“1”**. --- Result in repair costs.
- **False negatives (FN)** are **real failures “1” BUT** model predicted incorrectly **“0”**. --- Result in replacement costs.
- **False positives (FP)** are **NOT real failures “0” BUT** model predicted incorrectly **“1”**. --- Result in inspection costs.

- It is given that: the cost of inspection is **less than** the cost of repair and that
- cost of repairing a is **much less than** the cost of replacing it.

- cost least < cost a little more << cost most
- inspection < repair << replacing
- FP < TP << FN

Business Problem Overview – Objective – Recall

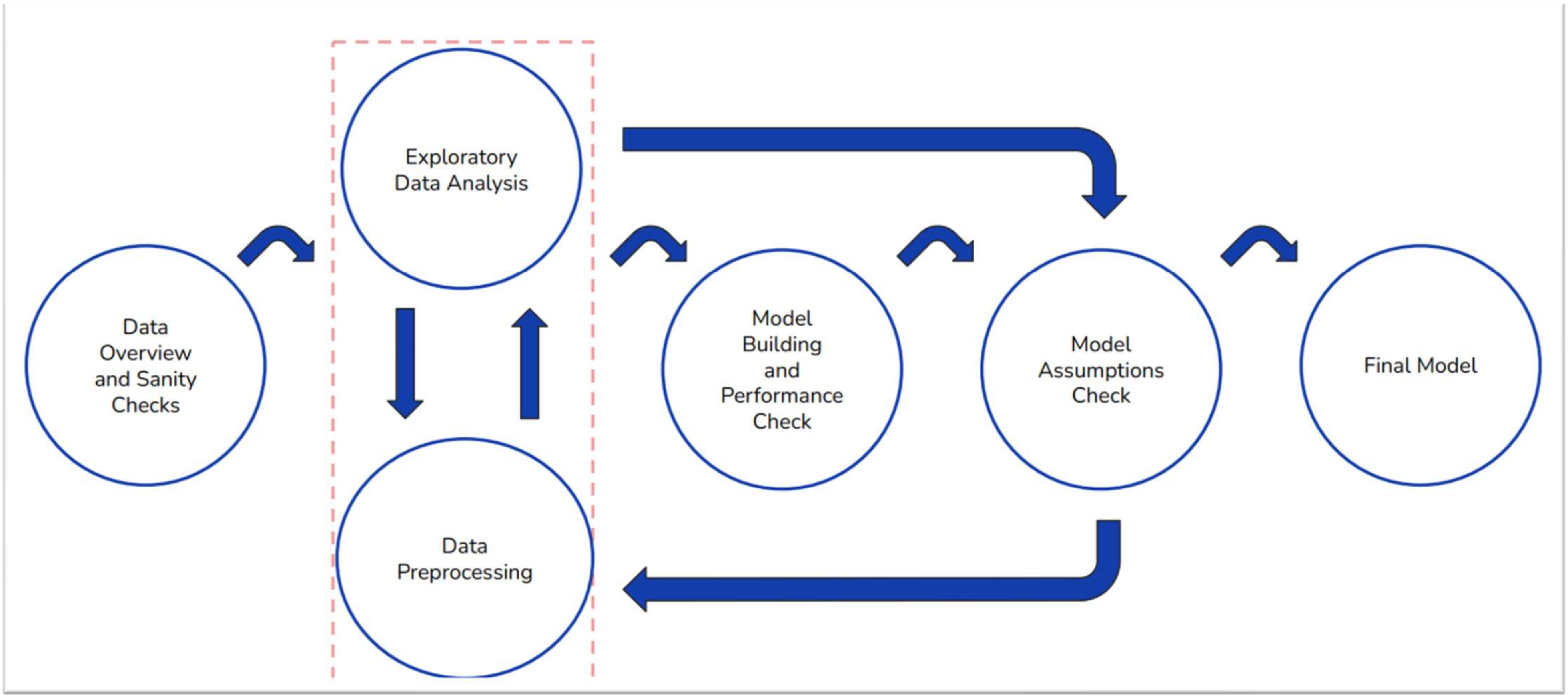
- We need to choose the metric which will **ensure that the maximum number of generator failures are predicted correctly by the model**.
- Optimizing for Recall --- We would want **Recall** to be maximized as greater the Recall, the higher the chances of **minimizing false negatives**.
- We want to **minimize 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.

Business Problem Overview – Data Description

- The data provided is a transformed version of original data which was collected using sensors.
- Train.csv - To be used for **training** and **tuning** of models **via validation sets**.
- Test.csv - To be used only for **testing** the **performance** of the **final best model**.

- Both the datasets consist of **40** predictor variables and **1** target variable
- The total data has **25,000 observations:**
  - **20,000 observations** in the training set
  - **5,000 observations** in the test set.

Business Problem Overview – Solution approach



Business Problem Overview – Solution approach

- **Exploratory Data Analysis and Insights**: - Overview of the data - Univariate analysis
- **Data pre-processing**: - Prepare the data for analysis - Missing value Treatment - Ensure no data leakage
- **Model building**
  - **Model building – <u>Original</u> data** : - Build at least 6 classification models (Using logistic regression, decision trees, random forest, bagging classifier and boosting methods (XGBoost))
  - **Model building - <u>Oversampled</u> data**: Build at least 6 classification models using oversampled train data.
  - **Model building - <u>Undersampled</u> data**: Build at least 6 classification models using undersampled train data.

Business Problem Overview – Solution approach

- **Hyperparameter tuning**: - Choose at least 3 best performing models among all the models built previously (Mention the reason for the choices made) - Tune the chosen models. - Check the performance of the tuned models.
- **Model Performances**: - Compare performances of the tuned models and choose a final model. - Check the performance of the final model on test data.
- **Productionize the model**: - Productionize the final model using pipelines.Model building - Oversampled data: Build at least 6 classification models using oversampled train data.

EDA Results

- Exploratory Data Analysis and Insights (Key results from EDA):
- - Overview of the data
- - Univariate analysis

EDA Results – Overview of the Data

- Checking the **shape** of the dataset. Displaying the first few rows of the dataset.
- Checking the **data types** of the **columns** for the dataset
- Checking for **duplicate** values
- Checking for **missing** values
- **Statistical summary** of the dataset

EDA Results – Overview of the Data

- **Shape Train** shape of the dataset (20,000 rows, 41 columns)
- **Shape Test** shape of the dataset (5,000 rows, 41 columns)
- **Columns** “V1”, “V2”, “V3”...“V40”. The last column is “Target”
- **Data types** of the columns are all numeric data types (40 float & 1 int)
- **Duplicate values.** No.
- **Missing values**. Yes. Both Train and Test have missing values in V1 & V2

  - TRAIN — Missing values for columns V1 and V2. Both showing value of `0.090`. - All other columns show 0.000.
  - TEST — Missing values again for columns V1 and V2. The values are `0.100` and `0.120` respectively. - All other columns show 0.000.

- Train missing 18 values in V1 & V2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      19982 non-null  float64
 1   V2      19982 non-null  float64
 2   V3      20000 non-null  float64
 3   V4      20000 non-null  float64
```

Test missing 5 in V1 and 6 values in V2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      4995 non-null   float64
 1   V2      4994 non-null   float64
 2   V3      5000 non-null   float64
 3   V4      5000 non-null   float64
 4   V5      5000 non-null   float64
```

EDA Results – Overview of the Data – Statistical summary

- **Statistical summary** of the dataset
- Count ...count for V1 and V2 indicates 18 missing values
- Mean, ...positive and negative values, low single digits
- Standard deviation, ...highest for V32 at 5.500
- Minimum, ...all minimums are negative
- Range, ...possible outliers
- Maximum, ...highest max for V32 at 23.633
- **Train**

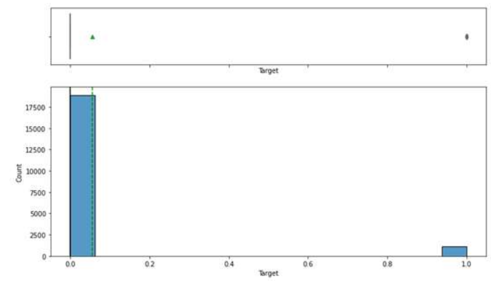
	count	mean	std	min	25%	50%	75%	max
V1	19982.000	-0.272	3.442	-11.876	-2.737	-0.748	1.840	15.493
V2	19982.000	0.440	3.151	-12.320	-1.641	0.472	2.544	13.089
V3	20000.000	2.485	3.389	-10.708	0.207	2.256	4.566	17.091
V4	20000.000	-0.083	3.432	-15.082	-2.348	-0.135	2.131	13.236
V39	20000.000	0.891	1.753	-6.439	-0.272	0.919	2.058	7.760
V40	20000.000	-0.876	3.012	-11.024	-2.940	-0.921	1.120	10.654
Target	20000.000	0.056	0.229	0.000	0.000	0.000	0.000	1.000

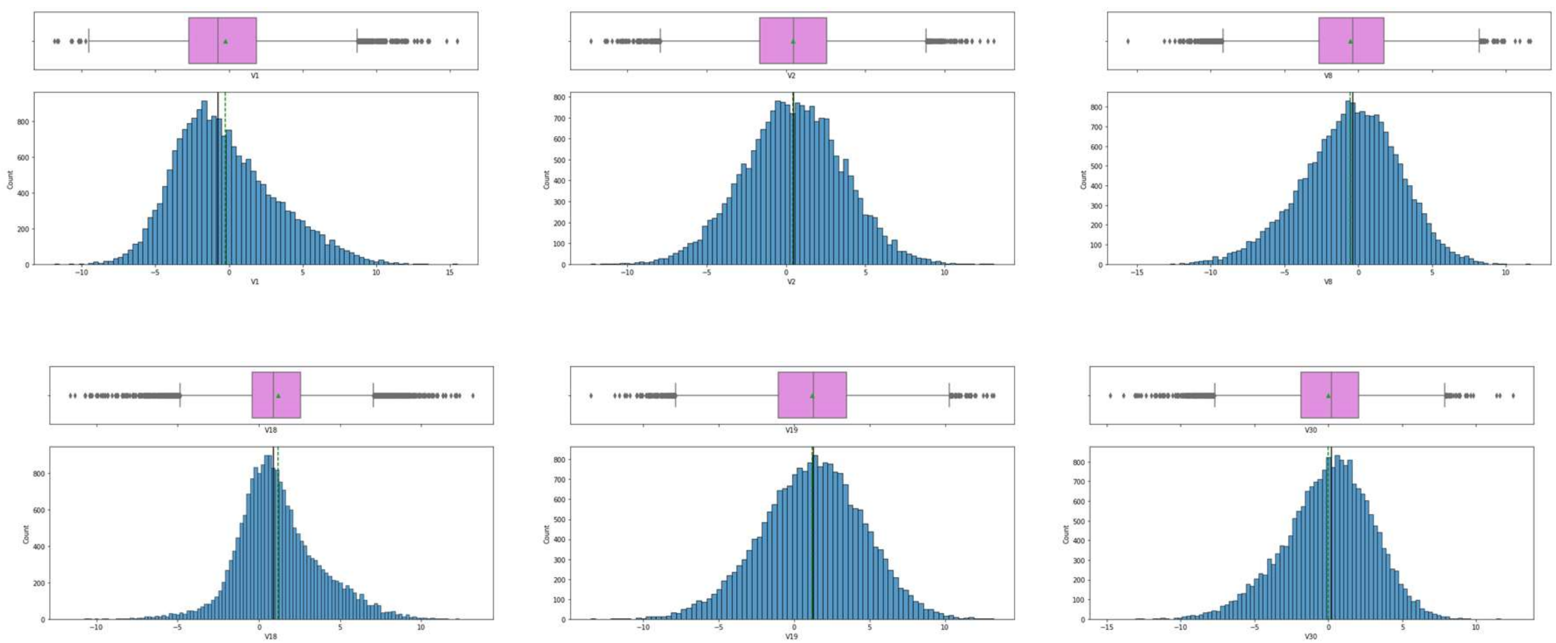
Test

	count	mean	std	min	25%	50%	75%	max
V1	4995.000	-0.278	3.466	-12.382	-2.744	-0.765	1.831	13.504
V2	4994.000	0.398	3.140	-10.716	-1.649	0.427	2.444	14.079
V3	5000.000	2.552	3.327	-9.238	0.315	2.260	4.587	15.315
V4	5000.000	-0.049	3.414	-14.682	-2.293	-0.146	2.166	12.140
V39	5000.000	0.939	1.717	-5.451	-0.208	0.959	2.131	7.182
V40	5000.000	-0.932	2.978	-10.076	-2.987	-1.003	1.080	8.698
Target	5000.000	0.056	0.231	0.000	0.000	0.000	0.000	1.000

EDA Results – Univariate analysis

- Distributions, consistently normal distributions throughout.
- Some skews for columns V1, V8, V10, V18, V30, V37.

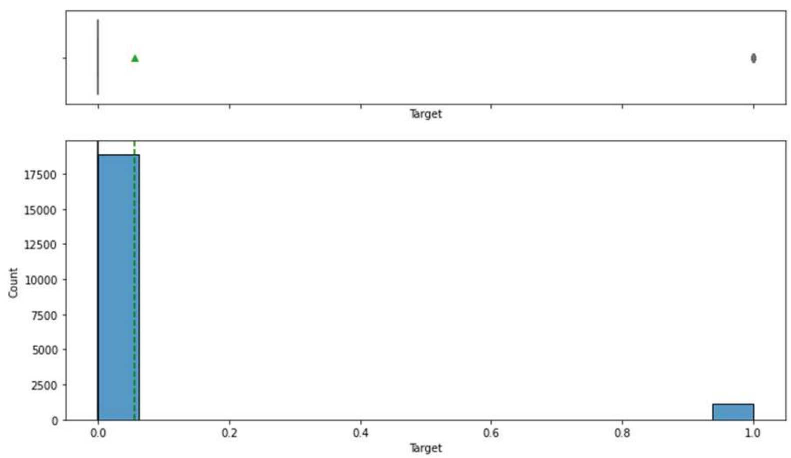


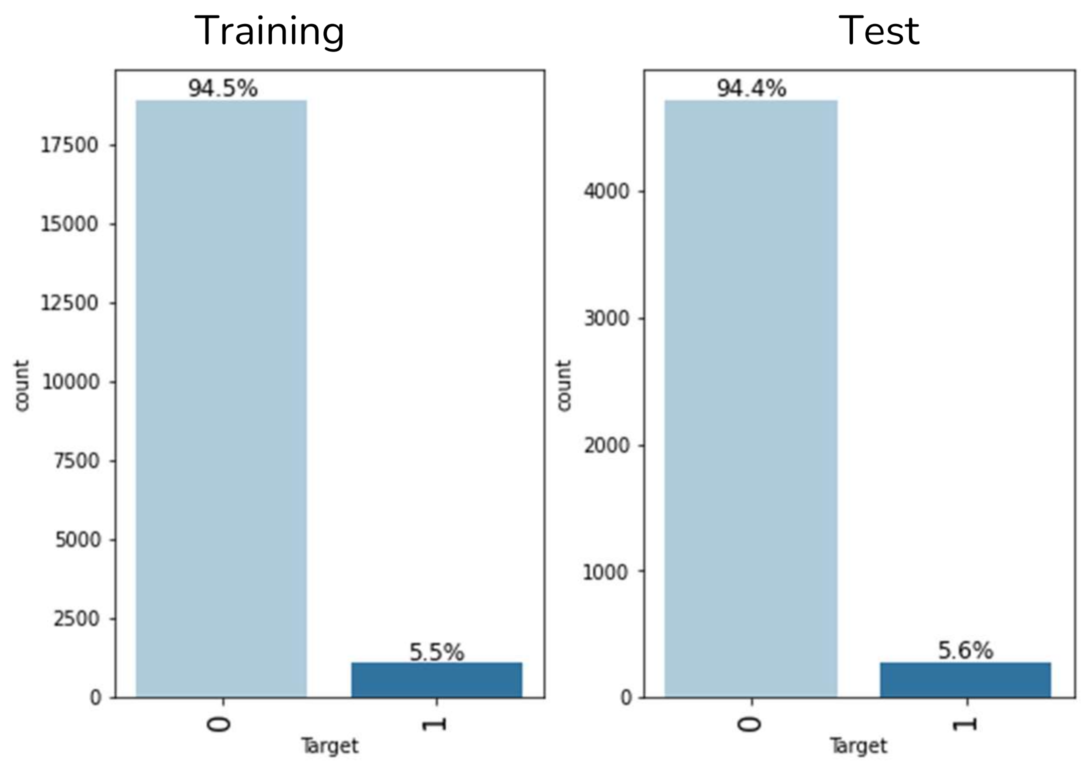


EDA Results – Univariate analysis (“Target”)

- The values are **imbalanced** 95% to 5% in target variable. This is addressed in model building.

- **~95% are “No failure” , “0”** and
- **~5% are “failure” represented by “1”**





Data Preprocessing

- Prepare the data for analysis
- Missing value Treatment
- Ensure no data leakage
- Duplicate value check
- Outlier check (treatment if needed)
- Feature engineering
- Data preparation for modeling

Data Preprocessing – Prepare the data for analysis

- Dividing **train** data into **X** and **y**
- Dividing **test** data into **X_test** and **y_test**
- Since we already have a separate **test** data set, we don't need to divide data into **train**, **validation**, and **test.**
- Divided the **train** data into **train** and **validation** data sets.
- Split **train** dataset into **training** and **validation** set in the ratio 75:25 ... **X_train**, **X_val**, **y_train**, **y_val**

```
# Checking the number of rows and columns in the X_train data
X_train.shape ##  Complete the code to view dimensions of the X_train data
```

```
(15000, 40)
```

```
# Checking the number of rows and columns in the X_val data
X_val.shape ##  Complete the code to view dimensions of the X_val data
```

```
(5000, 40)
```

```
# Checking the number of rows and columns in the X_test data
X_test.shape ##  Complete the code to view dimensions of the X_test data
```

```
(5000, 40)
```

```
# js ... Checking the number of rows and columns in the X_test data
y_test.shape ##  Complete the code to view dimensions of the X_test data
```

```
(5000,)
```

Data Preprocessing – Missing value Treatment – Imputation

- Used **Simple Imputer** to fill missing values with the **median** of train.
- Fit and transform the train data, X_train
- Transform the validation data, X_val
- Transform the test data, X_test

- Confirmed no missing values by checking the count of missing values.

```
[33] # creating an instace of the imputer to be used
     imputer = SimpleImputer(strategy="median")

[34] # Fit and transform the train data
     X_train = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)

     # Transform the validation data
     X_val = pd.DataFrame(imputer.transform(X_val), columns=X_train.columns) ## Complet
     ## X_val = pd.DataFrame(imputer._______(X_val), columns=X_train.columns)
     ## Complete the code to impute missing values in X_val without data leakage

     # Transform the test data
     X_test = pd.DataFrame(imputer.transform(X_test), columns=X_train.columns) ## Compl
     ## X_test = pd.DataFrame(imputer._______(X_test), columns=X_train.columns)
     ## Complete the code to impute missing values in X_test without data leakage
```

Model Building – Model evaluation criterion

- **Which metric to optimize? … … *Recall*.**
    - 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**.

- **Defining scorer** to be used for **cross-validation** and **hyperparameter tuning**
    - We want to reduce false negatives and will try to maximize "Recall".
    - To maximize Recall, we can **use Recall as a scorer** in **cross-validation** and **hyperparameter tuning**.

Model Building Summary (See Appendix for details.)

- **Model building – <u>Original</u> data** : - Build at least 6 classification models (Using logistic regression, decision trees, random forest, bagging classifier and boosting methods (XGBoost))
- **Model building - <u>Oversampled</u> data**: Build at least 6 classification models using oversampled train data.
- **Model building - <u>Undersampled</u> data**: Build at least 6 classification models using undersampled train data.

ORIGINAL DATA	CV	Val	GapDiff	OVERSAMPLED DATA	CV	Val	GapDiff	UNDERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	49%	48%	1%	Logistic regression:	88%	85%	4%	Logistic regression:	87%	85%	2%
Bagging:	72%	73%	-1%	Bagging:	98%	83%	14%	Bagging:	86%	87%	-1%
dtree:	70%	71%	-1%	dtree:	97%	78%	20%	dtree:	86%	84%	2%
Random forest:	72%	73%	0%	Random forest:	98%	85%	13%	Random forest:	90%	89%	1%
GBM:	71%	72%	-2%	GBM:	93%	88%	5%	GBM:	90%	89%	1%
Adaboost:	63%	68%	-5%	Adaboost:	90%	86%	4%	Adaboost:	87%	85%	2%
Xgboost:	74%	76%	-2%	Xgboost:	92%	87%	5%	Xgboost:	90%	89%	1%

Model Performance Summary (original, over, under)

- **ORIGINAL**

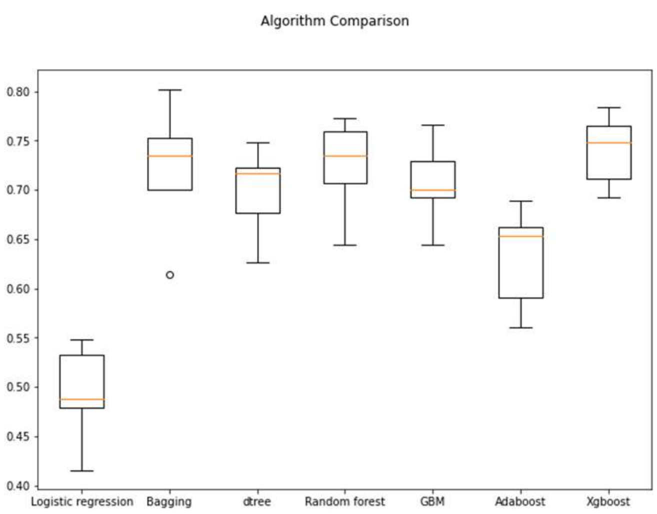
- Baseline wide range
- +/- 5 to7 basis points,
- wide confidence interval

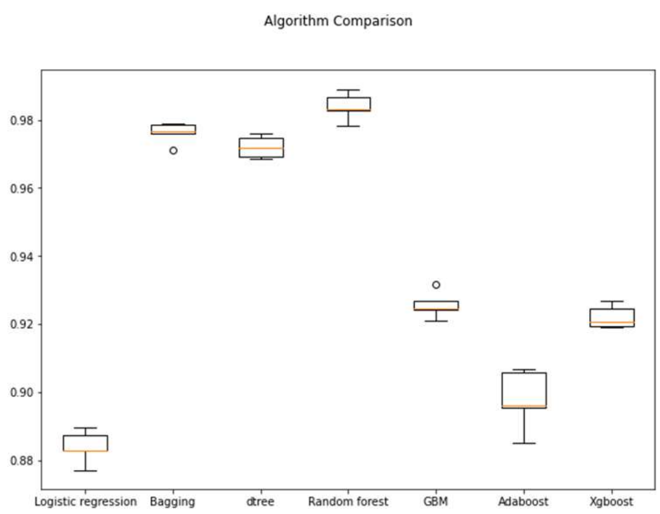
- **OVERSAMPLED**

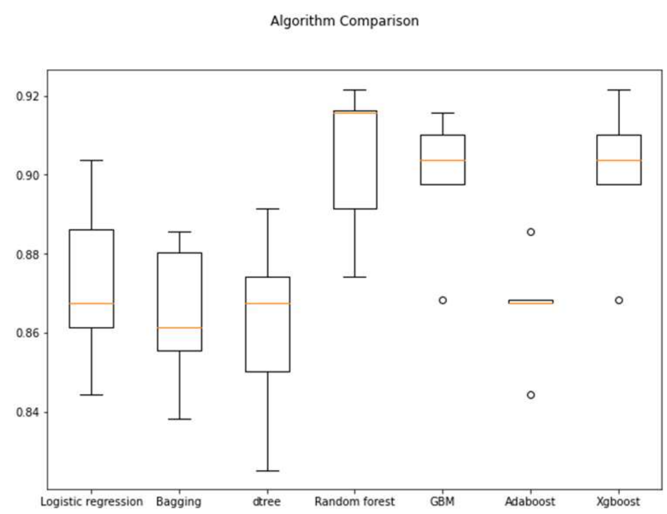
- Narrower range,
- +/- 1 to 3 basis points
  Narrower confidence interval
  or some, better
- Some possible overfitting.

- **UNDERSAMPLED**

- Wider range,
- +/- 1 to 3 basis points wider
  confidence interval







Model Performance Summary – Before Tuning Choose four

- Best models to for next stage, for Hyperparameter
- Tuning **AdaBoost** using **oversampled** data
- Tuning **Random forest** using **undersampled** data
- Tuning **Gradient Boosting** using **oversampled** data
- Tuning **XGBoost** using **oversampled** data

OVERSAMPLED DATA	CV	Val	GapDiff	UNDERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	88%	85%	4%	Logistic regression:	87%	85%	2%
Bagging:	98%	83%	14%	Bagging:	86%	87%	-1%
dtree:	97%	78%	20%	dtree:	86%	84%	2%
Random forest:	98%	85%	13%	Random forest:	90%	89%	1%
GBM:	93%	88%	5%	GBM:	90%	89%	1%
Adaboost:	90%	86%	4%	Adaboost:	87%	85%	2%
Xgboost:	92%	87%	5%	Xgboost:	90%	89%	1%

Model Performance Summary – Hyperparameter Tuning

- **Tuned Models ---** Training Performance VS Validation Performance --- Checking the performance of the tuned models.

Training performance comparison:

	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data	Gradient Boosting tuned with oversampled data	XGBoost tuned with oversampled data
Accuracy	0.992	0.961	0.993	0.978
Recall	0.988	0.933	0.992	1.000
Precision	0.995	0.989	0.994	0.959
F1	0.992	0.960	0.993	0.979

Validation performance comparison:

	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data	Gradient Boosting tuned with oversampled data	XGBoost tuned with oversampled data
Accuracy	0.979	0.938	0.969	0.938
Recall	0.856	0.885	0.856	0.885
Precision	0.791	0.468	0.678	0.470
F1	0.822	0.612	0.757	0.614

Model Performance Summary – Hyperparameter Tuning – Final

- Evaluation --- Comparing the Training performance with the Validation Performance focusing on the Recall score and focusing on the minimizing the gap between training and validation scores. The any gap over 5% will be interpreted as overfitting.

- **Random forest tuned with undersampled data is chosen as the best because:**

  - It has a good training recall score (**0.933**) with an acceptable gap to the validation recall score (**0.885**).

  - The is not overfitting with an acceptable **gap of less than 5%**.

  - **All three other models are overfitting.**

- **Overfitting**
- Tuning **AdaBoost** using **over**sampled data --- Over fit training data set at 0.988 large gap to reach 0.853
- Tuning **Gradient Boosting** using **oversampled** data --- Over fit training data set at 0.992. large gap to reach 0.856
- Tuning **XGBoost** using **oversampled** data --- Recall on the training set is 1.0. Something is likely wrong and need to revisit inputs arguments model parameters and hyperparameters.

Model Performance Summary (final model on test data)

- **Random forest tuned with undersampled data:**
- Recall scores
  - **Training** recall score (**0.933**)
  - **Validation** recall score (**0.885**).
  - **Test** recall score (**0.879**)
- **Test train gap is above 5%.** Indicating the model is somewhat overfitting. Test performed slightly worst than validation by 0.006
- Performance on test data is **NOT generalized yet** with regard to recall and is not ready for production.
- There is still more work to be done to improve the prediction.
- Precision scores tank down to 0.500. This is no better a prediction than a coin.
- Next iteration…recommend **revisiting** the other three classifiers with oversampled data: **GBM**, **Ada**, and **Xgb**. Compare against test data. Oversampled data perferd.

Training performance:

	Accuracy	Recall	Precision	F1
0	0.961	0.933	0.989	0.960

Validation performance:

	Accuracy	Recall	Precision	F1
0	0.938	0.885	0.468	0.612

Test performance of Random forest using undersampled data:

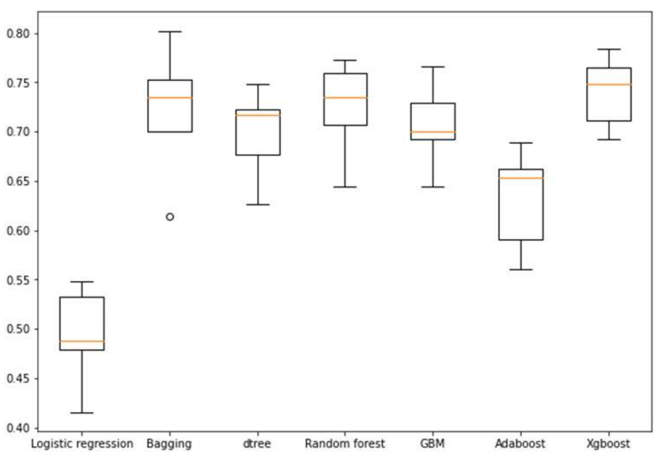
	Accuracy	Recall	Precision	F1
0	0.944	0.879	0.500	0.638

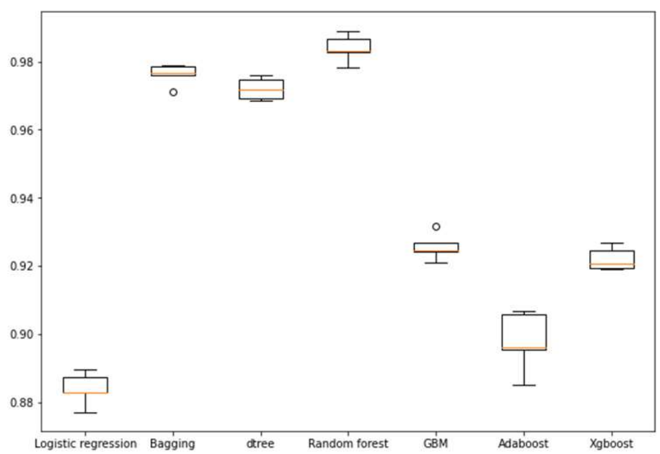
Model Performance Summary (original, over, under)

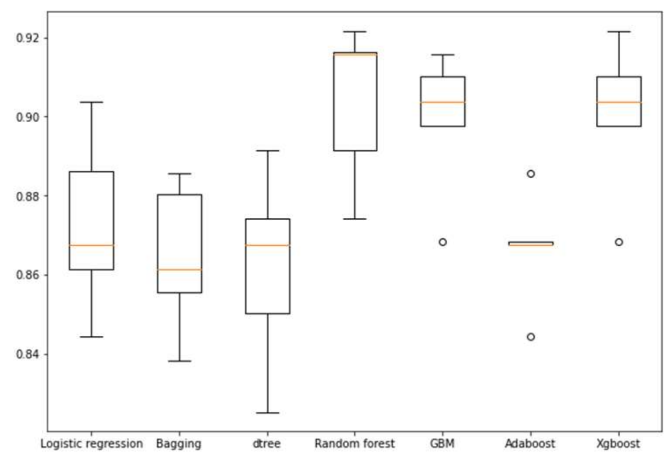
ORIGINAL DATA	CV	Val	GapDiff
Logistic regression:	49%	48%	1%
Bagging:	72%	73%	-1%
dtree:	70%	71%	-1%
Random forest:	72%	73%	0%
GBM:	71%	72%	-2%
Adaboost:	63%	68%	-5%
Xgboost:	74%	76%	-2%

OVERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	88%	85%	4%
Bagging:	98%	83%	14%
dtree:	97%	78%	20%
Random forest:	98%	85%	13%
GBM:	93%	88%	5%
Adaboost:	90%	86%	4%
Xgboost:	92%	87%	5%

UNDERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	87%	85%	2%
Bagging:	86%	87%	-1%
dtree:	86%	84%	2%
Random forest:	90%	89%	1%
GBM:	90%	89%	1%
Adaboost:	87%	85%	2%
Xgboost:	90%	89%	1%







Productionize and test the final model using pipelines

- **Steps** taken to **create a pipeline** for the final model
- Summary of the performance of the **model built with pipeline** on test dataset
- Summary of **most important factors** used by the model built with pipeline for prediction

Productionize and test the final model using pipelines

- Steps taken to **create a pipeline for the final model**
- **Step 1** - Create a pipeline with a simple imputer. This basic structure of the pipeline is as follows:
- **Step 2** - Separate the target variable and other variables into **X1**(independent variables) and **y1**(target) for train data. Repeat the same for the test data, **X_test1, y_test1**.
- **Step 3** - We can't oversample/**undersample** data without doing missing value treatment, so first, we have to **treat missing values** in the train and test sets.
- **Step 4** - **Undersampled** the train data and create necessary variables for them, **X_un1, y_un1**
- Step 5 - **Fit** the model on the **undersampled** train data
- Step 6 - Check the **performance** of the **Pipeline_model** on the test data

Productionize and test the final model using pipelines

- Steps taken to create a pipeline for the final model

```
Pipeline(steps=[('imputer_js_nm',
                 Pipeline(steps=[('imputer',
                                  SimpleImputer(strategy='median'))])),
                ('rft_js',
                 RandomForestClassifier(max_features='sqrt', max_samples=0.5,
                                        min_samples_leaf=2, n_estimators=300,
                                        random_state=1))])
```

- Summary of the performance of the model built with pipeline on test dataset

Test performance from
Pipeline_model_test:

	Accuracy	Recall	Precision	F1
0	0.945	0.872	0.507	0.641

Productionize final model – Feature Importances

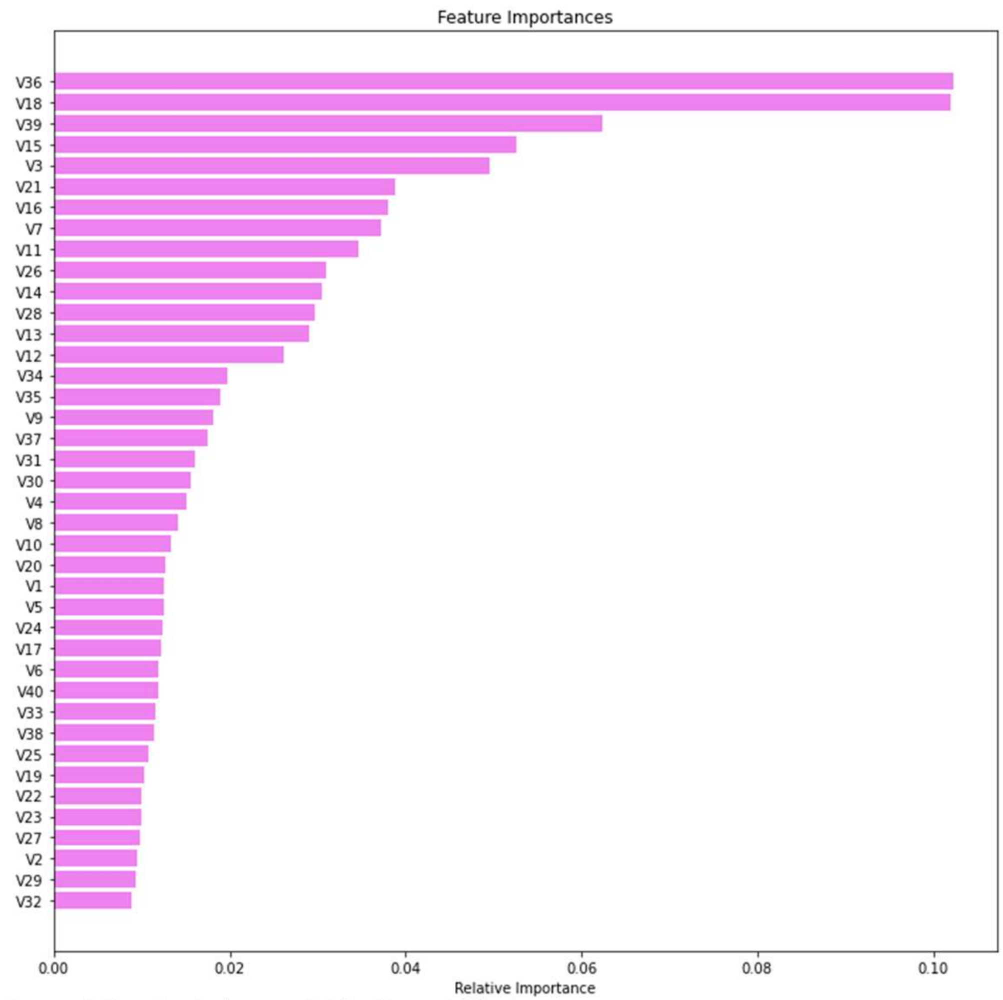
- Summary of most important factors used by the model built with pipeline for prediction.

<u>Top five</u>
- V36 and V8 are the most important feature for prediction. **Relevant Importance (R.I.)** of ~**0.10** followed by…

- **V39**, **V15**, and **V3**
- **R.I.** of range ~**0.05** to ~**0.06**.

<u>Middle Nine</u>
- Next nine features (9)
- **R.I.** of range ~0.03 to ~0.04

<u>Lower All Others (26)</u>
- The remaining 26 features have a Relevant Importance of range ~0.01 to ~0.02

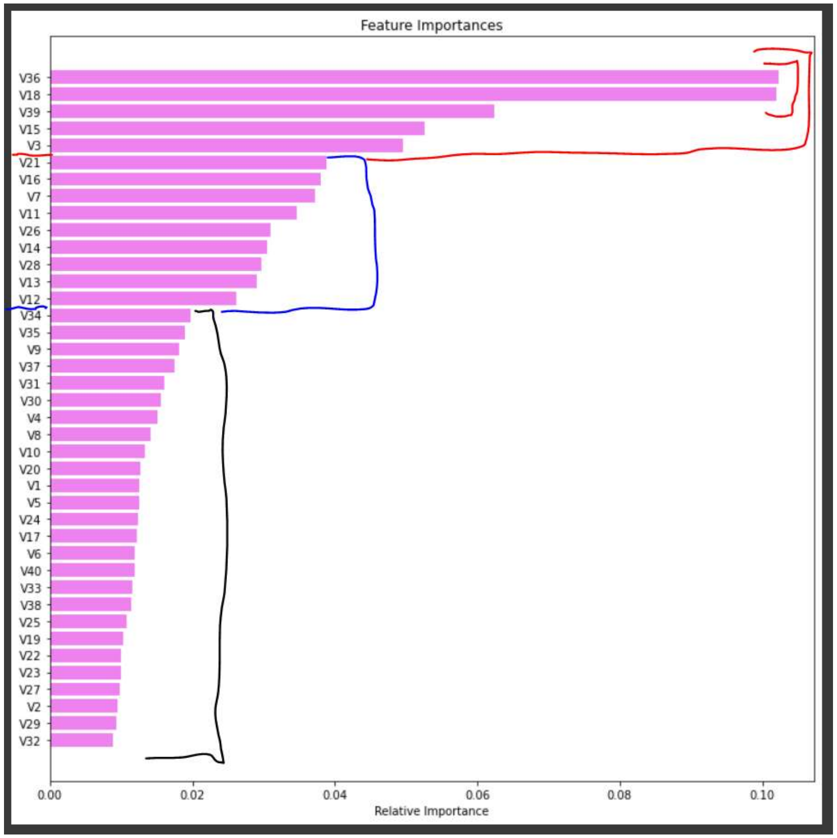


Productionize final model – Feature Importances

Top five(5)

Middle Nine (9)

Lower All Others (26)



APPENDIX

Model Performance Summary - Overview

- **Model building – Original data** : - Build at least 6 classification models (Using logistic regression, decision trees, random forest, bagging classifier and boosting methods (XGBoost))
- **Model building - Oversampled data**: Build at least 6 classification models using oversampled train data.
- **Model building - Undersampled data**: Build at least 6 classification models using undersampled train data.

ORIGINAL DATA	CV	Val	GapDiff	OVERSAMPLED DATA	CV	Val	GapDiff	UNDERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	49%	48%	1%	Logistic regression:	88%	85%	4%	Logistic regression:	87%	85%	2%
Bagging:	72%	73%	-1%	Bagging:	98%	83%	14%	Bagging:	86%	87%	-1%
dtree:	70%	71%	-1%	dtree:	97%	78%	20%	dtree:	86%	84%	2%
Random forest:	72%	73%	0%	Random forest:	98%	85%	13%	Random forest:	90%	89%	1%
GBM:	71%	72%	-2%	GBM:	93%	88%	5%	GBM:	90%	89%	1%
Adaboost:	63%	68%	-5%	Adaboost:	90%	86%	4%	Adaboost:	87%	85%	2%
Xgboost:	74%	76%	-2%	Xgboost:	92%	87%	5%	Xgboost:	90%	89%	1%

Model Performance Summary - Overview

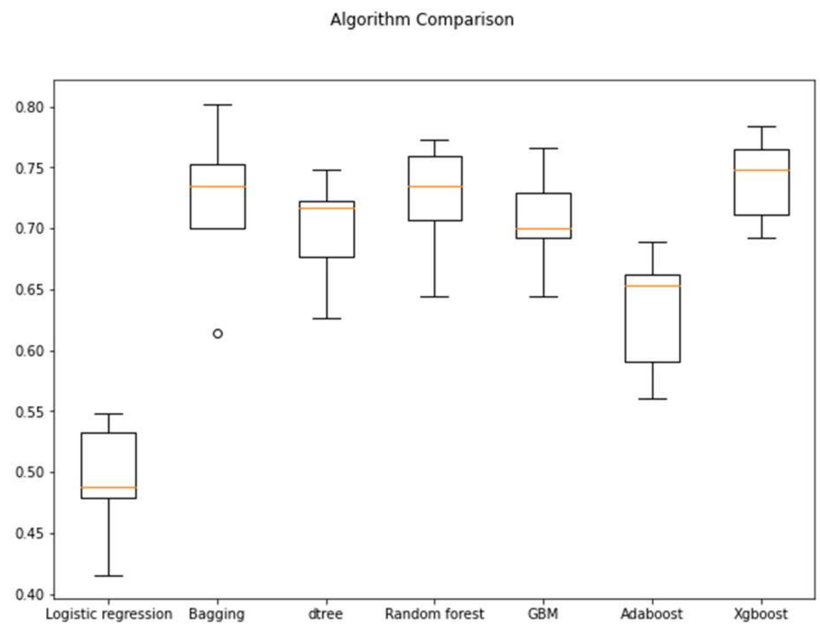
- ****Model Building note****
  - Defined **recall** as **scorer** to be used for **cross-validation** and **hyperprameter** tuning
  - Looped through all models to get the **mean cross-validated score (**5-fold per algorithm).

ORIGINAL DATA	CV	Val	GapDiff	OVERSAMPLED DATA	CV	Val	GapDiff	UNDERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	49%	48%	1%	Logistic regression:	88%	85%	4%	Logistic regression:	87%	85%	2%
Bagging:	72%	73%	-1%	Bagging:	98%	83%	14%	Bagging:	86%	87%	-1%
dtree:	70%	71%	-1%	dtree:	97%	78%	20%	dtree:	86%	84%	2%
Random forest:	72%	73%	0%	Random forest:	98%	85%	13%	Random forest:	90%	89%	1%
GBM:	71%	72%	-2%	GBM:	93%	88%	5%	GBM:	90%	89%	1%
Adaboost:	63%	68%	-5%	Adaboost:	90%	86%	4%	Adaboost:	87%	85%	2%
Xgboost:	74%	76%	-2%	Xgboost:	92%	87%	5%	Xgboost:	90%	89%	1%

Model Performance Summary (original data)

- **Model performance – <u>Original</u> data** : **Model** --- **Bagging** and **Random forest** and **Xgboost** performed the best for the **original** data group of classifiers. Generally no overfitting, lower 70%

ORIGINAL DATA	CV	Val	GapDiff
Logistic regression:	49%	48%	1%
Bagging:	72%	73%	-1%
dtree:	70%	71%	-1%
Random forest:	72%	73%	0%
GBM:	71%	72%	-2%
Adaboost:	63%	68%	-5%
Xgboost:	74%	76%	-2%



- Boxplots to compare algorithm for CV scores of all models to check model performance on original data. --- shows range estimate of 5 fold per algorithm.

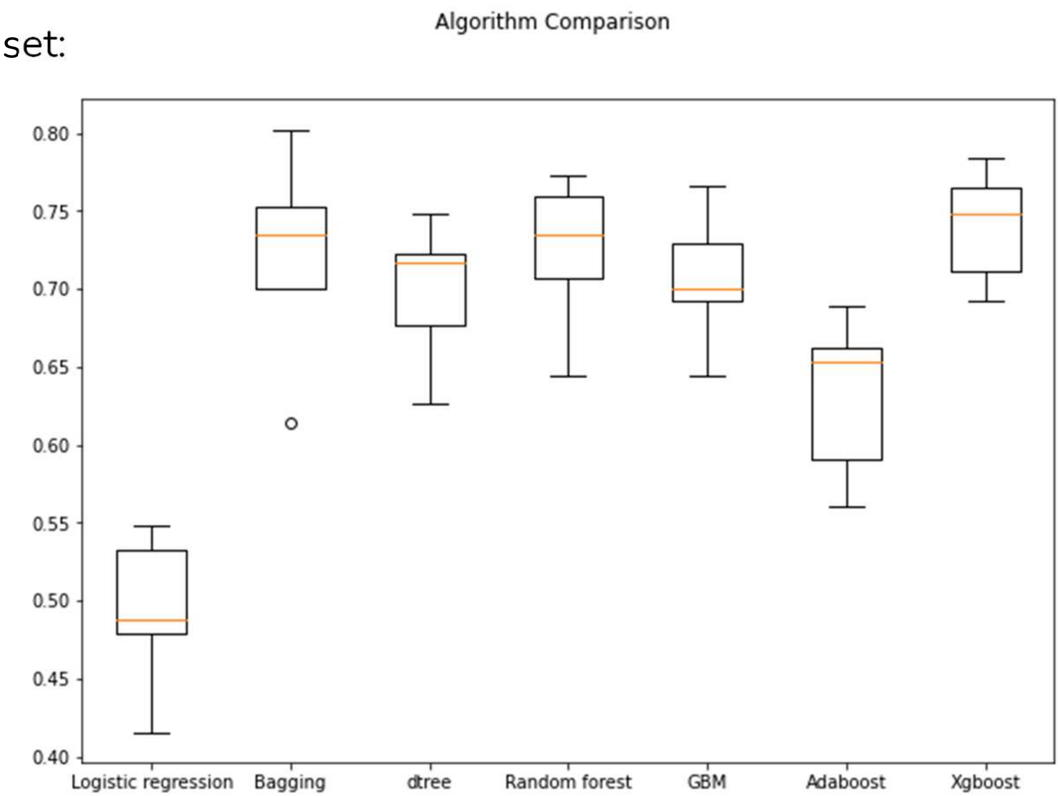
Model Performance Summary (original data)

Cross-Validation performance on **training** dataset:

• Logistic regression:	0.492
• **Bagging**:	**0.721**
• dtree:	0.698
• **Random forest**:	**0.723**
• GBM:	0.706
• Adaboost:	0.630
• **Xgboost**:	**0.740**

Validation Performance:

• Logistic regression:	0.482
• **Bagging**:	**0.730**
• dtree:	0.705
• **Random forest**:	**0.726**
• GBM:	0.723
• Adaboost:	0.676
• **Xgboost**:	**0.762**



- Boxplots to compare algorithm for CV scores of all models to check model performance on original data. --- shows range estimate of 5 fold per algorithm.

Model Performance Summary (oversampled data)

- The **oversampling method** chosen = **SMOTE** (Synthetic Minority Over Sampling Technique)
- **Model performance** --- Significant improvement, increased from low 70s in original data to low 90s in oversampled data. Three (3) models overfit, four (4) models nearly overfit. Best performers **GBM** and **Adaboost** and **Xgboost** performed the best without overfitting group of classifiers.
- Increasing lot of information, losing ~20% of the data
  - Going from 20,000 observations to 28,336

```
Before OverSampling, counts of label '1': 832
Before OverSampling, counts of label '0': 14168

After OverSampling, counts of label '1': 14168
After OverSampling, counts of label '0': 14168

After OverSampling, the shape of train_X: (28336, 40)
After OverSampling, the shape of train_y: (28336,)
```

OVERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	88%	85%	4%
Bagging:	98%	83%	14%
dtree:	97%	78%	20%
Random forest:	98%	85%	13%
GBM:	93%	88%	5%
Adaboost:	90%	86%	4%
Xgboost:	92%	87%	5%

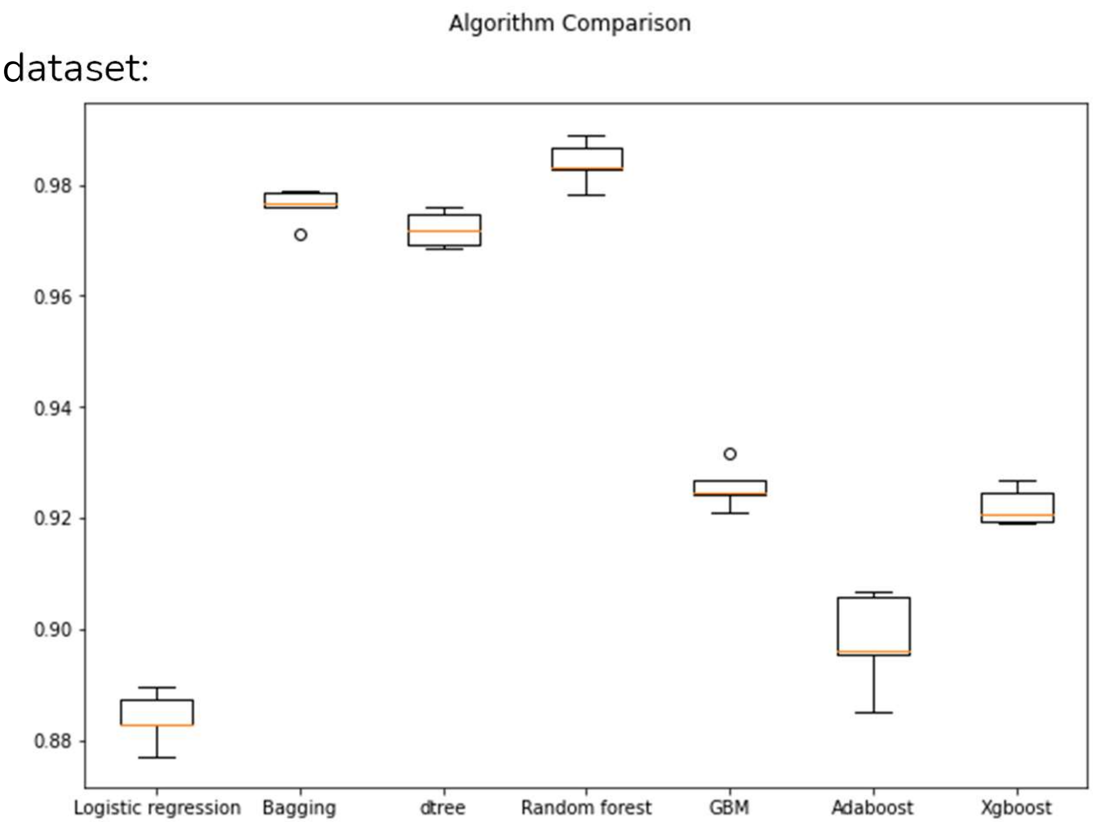
Model Performance Summary (oversampled data)

- **Cross-Validation** performance on **training** dataset:

- Logistic regression: 0.883
- Bagging: 0.976
- dtree: 0.972
- Random forest: 0.983
- **GBM**: **0.925**
- **Adaboost**: **0.897**
- **Xgboost**: **0.922**

- **Validation Performance**:

- Logistic regression: 0.848
- Bagging: 0.834
- dtree: 0.776
- Random forest: 0.848
- **GBM: 0.877**
- **Adaboost: 0.856**
- **Xgboost: 0.874**



- Boxplots to compare algorithm for CV scores of all models to check model performance on original data.

Model Performance Summary (undersampled data)

- The **undersampling method** chosen = **RandomUnderSampler()**
- **Model performance** --- Good improvement, increased from low 70s in original data to high 80s some 90 in oversampled data. None look to be overfit.
- Best performers **Random forest** and **GBM** and **Xgboost** performed the best for **undersampled** data.
- Lost a lot of information, losing ~20% of the data
  - Going from 20,000 observations to 1,654

```
Before UnderSampling, counts of label '1': 832
Before UnderSampling, counts of label '0': 14168

After UnderSampling, counts of label '1': 832
After UnderSampling, counts of label '0': 832

After UnderSampling, the shape of train_X: (1664, 40)
After UnderSampling, the shape of train_y: (1664,)
```

UNDERSAMPLED DATA	CV	Val	GapDiff
Logistic regression:	87%	85%	2%
Bagging:	86%	87%	-1%
dtree:	86%	84%	2%
Random forest:	90%	89%	1%
GBM:	90%	89%	1%
Adaboost:	87%	85%	2%
Xgboost:	90%	89%	1%

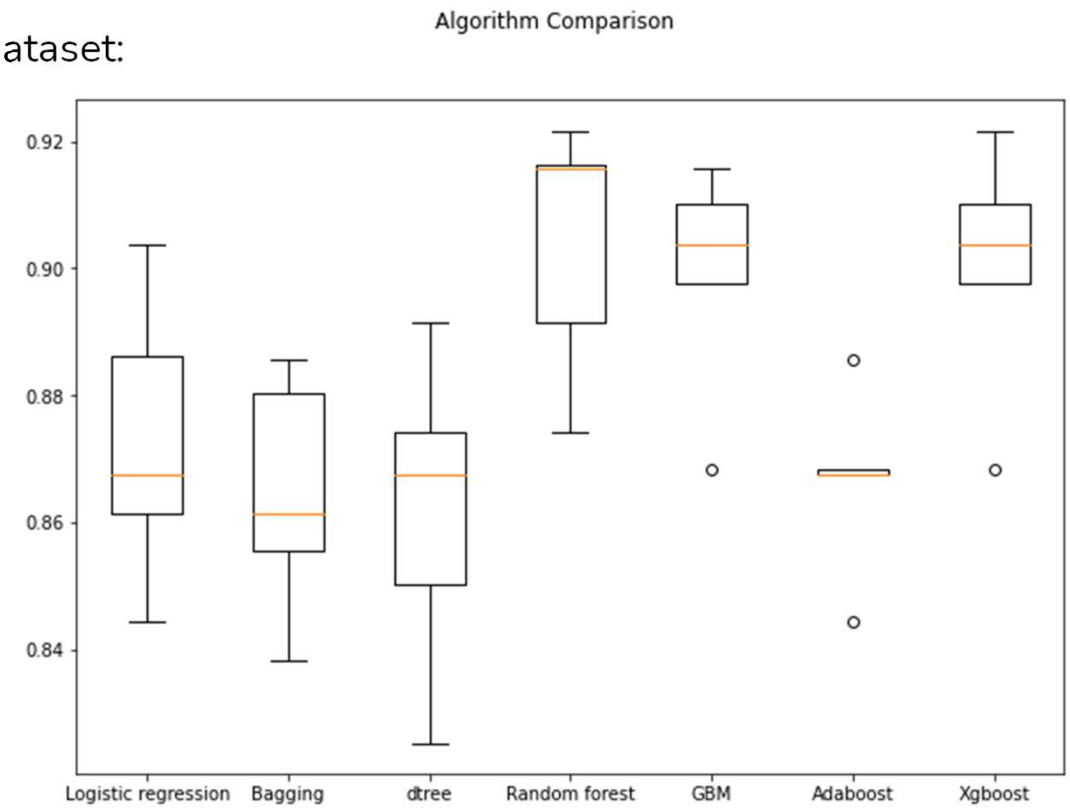
Model Performance Summary (undersampled data)

- **Cross-Validation** performance on **training** dataset:

- Logistic regression: 0.872
- Bagging: 0.864
- dtree: 0.861
- **Random forest: 0.903**
- **GBM: 0.899**
- Adaboost: 0.866
- **Xgboost: 0.900**

- **Validation Performance:**

- Logistic regression: 0.852
- Bagging: 0.870
- dtree: 0.841
- **Random forest: 0.892**
- **GBM: 0.888**
- Adaboost: 0.848
- **Xgboost: 0.888**



- Boxplots to compare algorithm for CV scores of all models to check model performance on original data.

Model Performance Summary – Hyperparameter Tuning

- Tunning **AdaBoost** using **<u>over</u>**sampled data. Best parameters from

```
Best parameters are {'n_estimators': 200, 'learning_rate': 0.2, 'base_estimator': DecisionTreeClassifier(max_depth=3, random_state=1)} with CV score=0.9715559462639259:
CPU times: user 2min 11s, sys: 2.09 s, total: 2min 13s
Wall time: 58min 47s
```

- Tuning **Random forest** using **<u>un</u>**dersampled data

```
Best parameters are {'n_estimators': 300, 'min_samples_leaf': 2, 'max_samples': 0.5, 'max_features': 'sqrt'} with CV score=0.8990116153235697:
CPU times: user 3.41 s, sys: 165 ms, total: 3.57 s
Wall time: 2min 25s
```

- Tuning **Gradient Boosting** using **<u>over</u>**sampled data

```
Best parameters are {'subsample': 0.7, 'n_estimators': 125, 'max_features': 0.5, 'learning_rate': 1} with CV score=0.9723322092856124:
CPU times: user 31.7 s, sys: 985 ms, total: 32.6 s
Wall time: 26min 54s
```

- Tuning **XGBoost** using **<u>over</u>**sampled data

```
Best parameters are {'subsample': 0.8, 'scale_pos_weight': 10, 'n_estimators': 250, 'learning_rate': 0.2, 'gamma': 0} with CV score=0.9952006309347864:
CPU times: user 38.7 s, sys: 1.71 s, total: 40.4 s
Wall time: 52min 27s
```

Hyperparameter Tuning (classifier)

- Tunning **AdaBoost** using **over**sampled data

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3,
                                                         random_state=1),
                   learning_rate=0.2, n_estimators=200)
```

- Tuning **Random forest** using **un**dersampled data

```
RandomForestClassifier(max_features='sqrt', max_samples=0.5, min_samples_leaf=2,
                       n_estimators=300, random_state=1)
```

- Tuning **Gradient Boosting** using **over**sampled data

```
GradientBoostingClassifier(learning_rate=1, max_features=0.5, n_estimators=125,
                           random_state=1, subsample=0.7)
```

- Tuning **XGBoost** using **over**sampled data

```
XGBClassifier(eval_metric='logloss', learning_rate=0.2, n_estimators=250,
              random_state=1, scale_pos_weight=10, subsample=0.8)
```

Hyperparameter Tuning (classifier)

- Tuning **XGBoost** using **<u>over</u>**sampled data … Recall on the training set is 1.0…something went wrong, 100% overfit. Adjusted n_estimators from 250 down to 150 then down to 50. Result was marginally better 0.999 train to 0.906 validation. Precision worst and worse. Need to REVISIT inputs arguments model parameters and hyperparameters.

```
XGBClassifier(eval_metric='logloss', learning_rate=0.2, n_estimators=250,
              random_state=1, scale_pos_weight=10, subsample=0.8)
```

```
Best parameters are {'subsample': 0.8, 'scale_pos_weight': 10, 'n_estimators': 250, 'learning_rate': 0.2, 'gamma': 0} with CV score=0.9952006309347864:
CPU times: user 38.7 s, sys: 1.71 s, total: 40.4 s
Wall time: 52min 27s
```

```
xgb2_train_perf
```

	Accuracy	Recall	Precision	F1
0	0.978	1.000	0.959	0.979

```
xgb2_val_perf
```

	Accuracy	Recall	Precision	F1
0	0.938	0.885	0.470	0.614

Test performance:

	Accuracy	Recall	Precision	F1
0	0.944	0.879	0.500	0.638

```
xgb2_train_perf = model_performance_classification_sklearn(xgb2, X_train_over, y_train_over)
# xgb2_train_perf =  '_______' ## Complete the code to check the performance on oversampled train set
xgb2_train_perf
```

	Accuracy	Recall	Precision	F1
0	0.978	1.000	0.959	0.979

```
xgb2_val_perf = model_performance_classification_sklearn(xgb2, X_val, y_val)
# xgb2_val_perf =  '_______' ## Complete the code to check the performance on validation set
xgb2_val_perf
```

	Accuracy	Recall	Precision	F1
0	0.938	0.885	0.470	0.614

Source Notes

- Sources: GREAT LEARNING
- Project 6: Model Tuning: ReneWind
- **Project FAQs: ReneWind FAQs**
- **Content FAQs:**
- **Week 1: Feature Engineering and Cross Validation**
- **Week 2: ML Pipeline and Hyperparameter Tuning**

Happy Learning !