ReneWind Failure Detection

PGP-DSBA _ ReneWind Project

February 05, 2024

Contents / Agenda

- Executive Summary
- Business Problem Overview
- Solution Approach
- EDA Results
- Data Preprocessing
- Model performance summary for hyperparameter tuning.
- Model building with pipeline
- Appendix

Executive Summary

- Considering recall performance on the validation set and the ability to generalize, the tuned random forest was identified as the best model
- The model trained on under-sampled data generalizes better on accuracy than on recall but worse on precision and F1
- V36 and V18 are by far the most important features of the tuned random forest classifier trained on undersampled data, followed by V39; for each of these 3 variables, an increase reduces the chances of a failure
- About 95% of the cases registered no failure; 5% of failure is obviously in minority but is large enough to impact ReneWind's maintenance cost
- ReneWind should seek means to improve, as much as possible, the values of V18, V36, and V39
- ReneWind might want to consider cost-effective control measures for the less important factors

Business Problem Overview

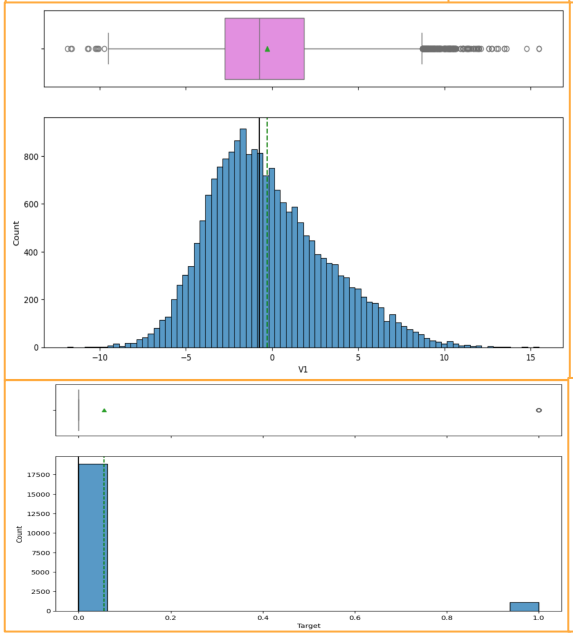
- Wind Energy is part of the increasingly popular renewable energy sources
- In response to the need to reduce maintenance cost associated with this source of energy, the U.S. Department of Energy has published a guide to achieving operational efficiency via predictive maintenance
- ReneWind, a company created to improve Wind Energy production, has collected data on generator failure of wind turbines using sensors fitted across components within the production environment
- The data collected has been ciphered and submitted for analysis so various classification models can be built and assessed – the task that defines the raison-d'etre of this project

Solution Approach

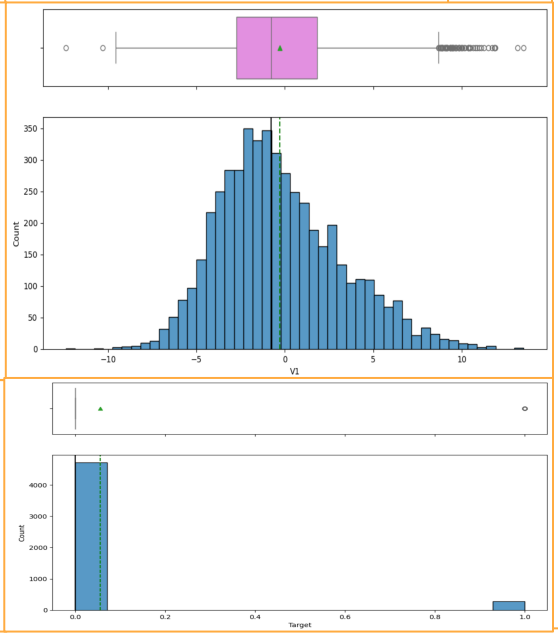
- To accomplish this task, we proceeded as follows:
  - We started out by carrying out Exploratory Data Analysis of the data provided to get an overview of the data and to uncover patterns within and across the different variables
  - We then moved on to Data Pre-Processing, essentially splitting the data into predictor and target variables, training, validation, and test sets, and carrying out imputation of missing values using the column median strategy separately for the training, validation, and test data sets, to avoid data leakage
  - Next, we went on to build and evaluate several models with default parameters as well as with tuned parameters, employing techniques such as oversampling and undersampling to further explore model tuning options
  - Finally, after evaluation on the validation set, one of the models was chosen as the best performing model, a pipeline was used to operationalize the chosen model, and its performance was evaluated on the test dataset

EDA Results: Univariate Analysis

Training Data



Test Data

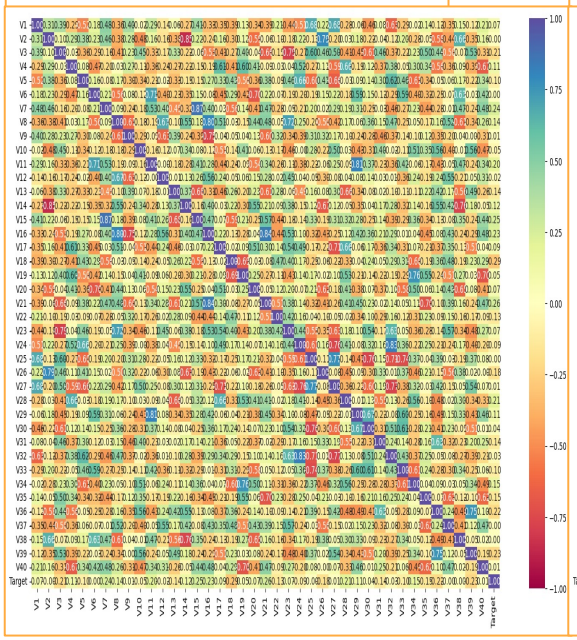


- From the plots of the variables of both the train and test datasets, we have confirmation that all except the target variable have a quasi-normal distribution
- We also have confirmation that cases of failure (Target = 1) constitute a minority class

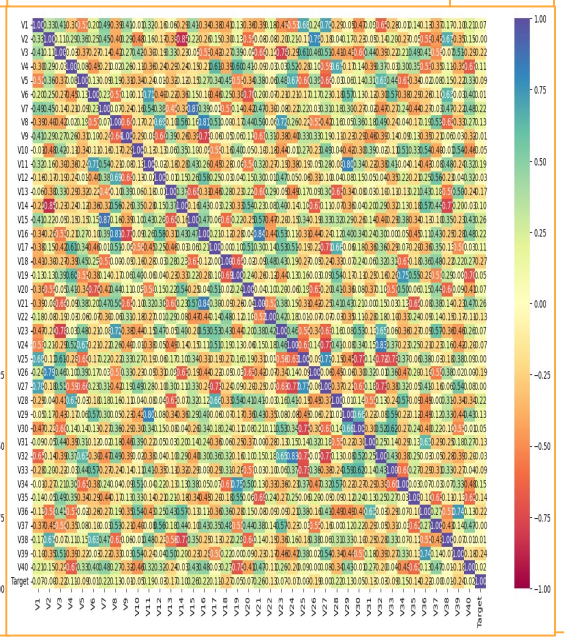
Link to Appendix slide on data background check

EDA Results: Bivariate Analysis: Correlation Check

Training Data



Test Data

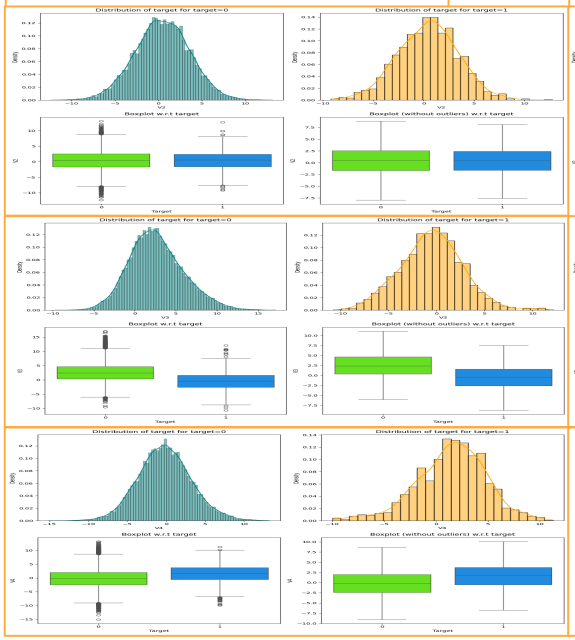


- In both the training and test data sets, significant levels of positive correlation are observed between some variables; for example between V7 and V15, V8 and V16, V11 and V29, V16 and V21, and V24 and V32
- In both sets, significant levels of negative correlation are also observed between some variables; for example between V2 and V14, V3 and V23, V6 and V20, V9 and V16, V14 and V38, V17 and V27, V19 and V40, V24 and V27, V25 and V30, V25 and V32, V25 and V33, and V27 and V32

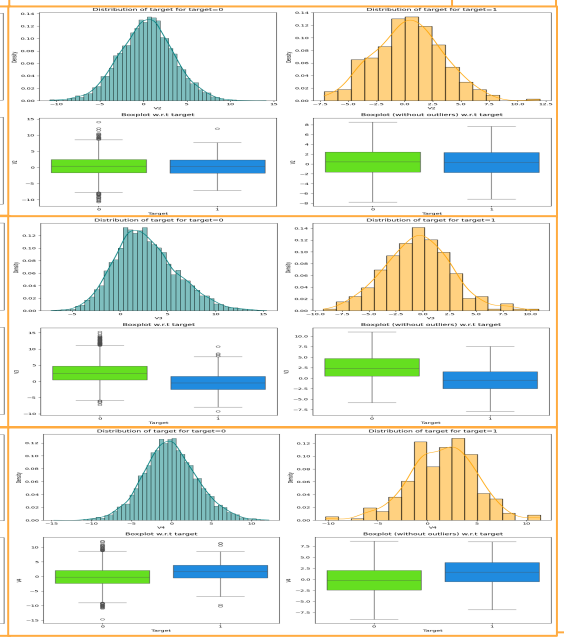
Link to Appendix slide on data background check

EDA Results: Bivariate Analysis: Distribution with respect to Target

Training Data



Test Data



- Both the training and test sets display similar distribution with respect to the Target variable; in both sets, almost all the variables display a quasi-normal distribution for each of the two classes in the Target
- Tend to have higher values for Target=1: V1, V4, V7, V8, V11, V14, V15, V16, V17, V19, V20, V21, V23, V28, V29, V34
- Tend to have higher values for Target=0: V3, V5, V10, V13, V18, V23, V24, V26, V31, V32, V33, V35, V36, V39
- Similar value trends for both Target values: V2, V6, V9, V12, V25, V27, V30, V37, V38, V40

Link to Appendix slide on data background check

Data Preprocessing

Before Data Split and Imputation

```
data.isnull().sum()
```

V1	18
V2	18
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
V29	0
V30	0
V31	0
V32	0
V33	0
V34	0
V35	0
V36	0
V37	0
V38	0
V39	0
V40	0
Target	0

dtype: int64

```
data_test.isnull().sum()
```

V1	5
V2	6
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
V29	0
V30	0
V31	0
V32	0
V33	0
V34	0
V35	0
V36	0
V37	0
V38	0
V39	0
V40	0
Target	0

dtype: int64

After Data Split and Imputation

Train:		Validation:		Test:	
V1	0	V1	0	V1	0
V2	0	V2	0	V2	0
V3	0	V3	0	V3	0
V4	0	V4	0	V4	0
V5	0	V5	0	V5	0
V6	0	V6	0	V6	0
V7	0	V7	0	V7	0
V8	0	V8	0	V8	0
V9	0	V9	0	V9	0
V10	0	V10	0	V10	0
V11	0	V11	0	V11	0
V12	0	V12	0	V12	0
V13	0	V13	0	V13	0
V14	0	V14	0	V14	0
V15	0	V15	0	V15	0
V16	0	V16	0	V16	0
V17	0	V17	0	V17	0
V18	0	V18	0	V18	0
V19	0	V19	0	V19	0
V20	0	V20	0	V20	0
V21	0	V21	0	V21	0
V22	0	V22	0	V22	0
V23	0	V23	0	V23	0
V24	0	V24	0	V24	0
V25	0	V25	0	V25	0
V26	0	V26	0	V26	0
V27	0	V27	0	V27	0
V28	0	V28	0	V28	0
V29	0	V29	0	V29	0
V30	0	V30	0	V30	0
V31	0	V31	0	V31	0
V32	0	V32	0	V32	0
V33	0	V33	0	V33	0
V34	0	V34	0	V34	0
V35	0	V35	0	V35	0
V36	0	V36	0	V36	0
V37	0	V37	0	V37	0
V38	0	V38	0	V38	0
V39	0	V39	0	V39	0
V40	0	V40	0	V40	0
dtype: int64		dtype: int64		dtype: int64	

```
[ ] # let's check for duplicate values in the training data
    data.duplicated().sum() ##  Complete the code to check duplicate entries in the data

    0

[ ] # let's check for duplicate values in the test data
    data_test.duplicated().sum() ##  Complete the code to check duplicate entries in the data

    0
```

```
Train:  (15000, 40)
Validation:  (5000, 40)
```

```
Test:  (5000, 40)
```

- Neither the training data nor the test data has duplicated data
- In the training data, columns V1 and V2 have 18 missing values each
- In the test data, column V1 has 5 missing values whereas column V2 has 6 missing values
- The X dataframe of the training dataset has 15,000 rows and that of the validation dataset has 5,000
- Both have 40 columns since the target variable has been excluded
- The X dataframe of the test dataset has 5,000 rows and 40 columns since the target variable has been excluded
- After simple imputation using the column median strategy, the missing values have been eradicated from the training, validation, and test datasets

Model Performance Summary: AdaBoost Tuning using oversampled data

Training Data

	Accuracy	Recall	Precision	F1
0	0.992	0.988	0.995	0.992

Validation Data

	Accuracy	Recall	Precision	F1
0	0.979	0.856	0.791	0.822

- The performance of AdaBoost Classifier with the best randomly chosen hyperparameters trained on oversampled data has a similar validation recall as that with default hyperparameters trained on the same
- The model with the best randomly chosen hyperparameters is overfitting the oversampled data.

Link to Appendix slide on model assumptions

Model Performance Summary: Random Forest Tuning using undersampled data

Training Data

	Accuracy	Recall	Precision	F1
0	0.961	0.933	0.989	0.960

Validation Data

	Accuracy	Recall	Precision	F1
0	0.938	0.885	0.468	0.612

- The performance of the RandomForest Classifier with the best randomly chosen hyperparameters trained on undersampled data has a slightly worse validation recall than that with default hyperparameters trained on the same
- In addition, it generalizes worse than the model with default hyperparameters trained on undersampled data.

Link to Appendix slide on model assumptions

Model Performance Summary: Gradient Boosting Tuning using oversampled data

Training Data

	Accuracy	Recall	Precision	F1
0	0.993	0.992	0.994	0.993

Validation Data

	Accuracy	Recall	Precision	F1
0	0.969	0.856	0.678	0.757

- The performance of the Gradient Boosting Classifier with the best randomly chosen hyperparameters trained on oversampled data has a slightly worse validation recall than that with default hyperparameters trained on the same
- The model with the best randomly chosen hyperparameters is overfitting the oversampled data

Link to Appendix slide on model assumptions

Model Performance Summary: XGBoost Tuning using oversampled data

Training Data

	Accuracy	Recall	Precision	F1
0	0.994	1.000	0.988	0.994

Validation Data

	Accuracy	Recall	Precision	F1
0	0.967	0.892	0.648	0.750

- The performance of the XGBoost Classifier with the best randomly chosen hyperparameters trained on oversampled data has a better validation recall than that with default parameters trained on the same
- However, the model still overfits the oversampled data

Link to Appendix slide on model assumptions

Model Performance Summary: Model performance comparison and choosing the final model

Training performance comparison:

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

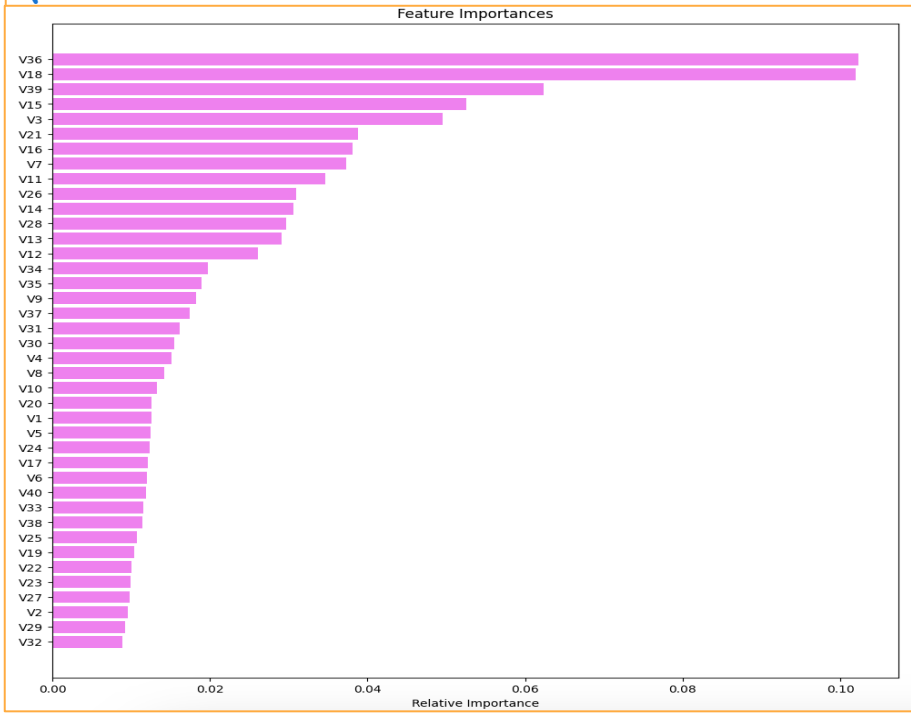
Validation performance comparison:

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data	XGBoost tuned with oversampled data
Accuracy	0.969	0.979	0.938	0.967
Recall	0.856	0.856	0.885	0.892
Precision	0.678	0.791	0.468	0.648
F1	0.757	0.822	0.612	0.750

- Accuracy: The tuned AdaBoost Classifier trained on oversampled data has the best validation score and also generalizes the best
- Recall: The tuned XGBoost Classifier trained on oversampled data has the best validation score but the tuned Random Forest Classifier trained on undersampled data, which has the second best validation score, generalizes better
- Precision: The tuned AdaBoost Classifier trained on oversampled data has the best validation score and also generalizes the best
- F1: The tune AdaBoost Classifier trained on oversample data has the best validation score and also generalizes the best
- Considering that recall is the most important measure for this project, the tuned Random Forest Classifier trained on undersampled data will be retained as the best model since it generalizes best and has the second best validation score

Link to Appendix slide on model assumptions

Model Performance Summary: Performance and Feature Importance of Best Model (Tuned Random Forest Classifier trained on undersampled data)



Test Data

	Accuracy	Recall	Precision	F1
0	0.944	0.879	0.500	0.638

- The tuned random forest classifier trained on undersampled data generalizes better on accuracy than on recall but worse on precision and F1
- V36 and V18 are by far the most important features of the tuned random forest classifier trained on undersampled data, followed by V39

Link to Appendix slide on model assumptions

Productionize and test the final model using pipelines

Training Data

	Accuracy	Recall	Precision	F1
0	0.962	0.929	0.995	0.961

Test Data

	Accuracy	Recall	Precision	F1
0	0.945	0.872	0.507	0.641

- A simple pipeline has been used, one that has one single step: running the model
- The result is very similar to that of the "unpiped" model: The pipeline model trained on undersampled data generalizes better on accuracy than on recall but worse on precision and F1
- As discussed earlier, V36 and V18 are by far the most important features of the tuned random forest classifier trained on undersampled data, followed by V39

Link to Appendix slide on model assumptions

APPENDIX

Data Background Check (1/3)

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

```
(20000, 41)
```

- The training data has 20,000 rows and 41 columns

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

```
(5000, 41)
```

```
# let's view the first 5 rows of the test data
data_test.head() ## Complete the code to view last 5 rows of the data
```

	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
0	2.592	-0.512	-1.023	7.339	-2.242	0.155	2.054	-2.772	1.851	-1.789	-0.277	-1.255	-3.833	-1.505	1.597	2.291	-5.411	0.870	0.574	4.157	1.428	-10.511	0.455	-1.448	0
1	3.367	2.418	1.762	-3.242	-3.193	1.857	-1.708	0.633	-0.588	0.084	3.014	-0.182	0.224	0.865	-1.782	-2.475	2.494	0.315	2.059	0.684	-0.485	5.128	1.721	-1.488	0
2	3.652	2.396	0.601	1.794	-2.120	0.482	-0.841	1.790	1.874	0.364	-0.169	-0.484	-2.119	-2.157	2.907	-1.319	-2.997	0.460	0.620	5.632	1.324	-1.752	1.808	1.676	0
3	0.772	-0.934	0.509	1.211	-3.260	0.105	-0.659	1.498	1.100	4.143	-0.248	-1.137	-5.356	-4.546	3.809	3.518	-3.074	-0.284	0.955	3.029	-1.367	-3.412	0.906	-2.451	0
4	2.494	-0.808	2.945	0.367	-5.789	4.598	4.450	3.225	0.397	0.248	-2.362	1.079	-0.473	2.243	-3.591	1.774	-1.502	-2.227	4.777	-6.560	-0.806	-0.276	-3.858	-0.538	0

```
# let's view the last 5 rows of the test data
data_test.tail() ## Complete the code to view last 5 rows of the data
```

	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
4995	2.403	3.792	0.487	-2.028	1.778	3.668	11.375	-1.977	2.252	-7.319	1.907	-3.734	-0.012	2.120	9.979	0.063	0.217	3.036	2.109	-0.557	1.939	0.513	-2.694	0
4996	3.698	-0.846	-0.222	-3.845	0.736	0.926	3.278	-2.277	4.458	-6.543	-1.348	-1.779	0.352	-0.214	4.424	2.604	-2.152	0.917	2.157	0.467	0.470	2.197	-2.377	0
4997	8.149	-9.199	-3.872	-0.296	1.468	2.884	2.792	-1.136	1.198	-4.342	-2.869	4.124	4.197	3.471	3.792	7.482	-10.061	-0.387	1.849	1.818	-1.246	-1.261	7.475	0
4998	2.536	-0.792	4.398	-4.073	-0.038	-2.371	-1.542	2.908	3.215	-0.169	-1.541	-4.724	-5.525	1.668	-4.100	-5.949	0.550	-1.574	6.824	2.139	-4.036	3.436	0.579	0
4999	0.226	4.963	6.732	-6.306	3.271	1.897	3.271	-0.637	-0.925	-6.759	2.990	-0.814	3.499	-8.435	2.370	-1.082	0.791	4.952	-7.441	-0.070	-0.916	-2.291	-5.363	0

```
# let's view the first 5 rows of the training data
data.head() ## Complete the code to view top 5 rows of the data
```

	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
0	-3.046	2.914	2.270	4.395	-2.388	0.646	-1.191	3.133	0.665	-2.511	-0.037	0.726	-3.982	-1.073	1.667	3.060	-1.690	2.846	2.235	6.667	0.444	-2.369	2.951	-3.480	0
1	0.605	0.908	0.757	-5.834	-3.065	1.597	-1.757	1.766	-0.267	3.625	1.500	-0.586	0.783	-0.201	0.025	-1.795	3.033	-2.468	1.895	-2.298	-1.731	5.909	-0.386	0.616	0
2	-3.818	3.689	3.311	1.059	-2.143	1.650	-1.661	1.680	-0.451	-4.551	3.739	1.134	-2.034	0.841	-1.600	-0.257	0.804	4.086	2.292	5.361	0.352	2.940	3.839	-4.309	0
3	1.774	-1.282	1.582	-1.952	-3.517	-1.206	-5.628	-1.818	2.124	5.295	4.748	-2.309	-3.963	-6.029	4.949	-3.584	-2.577	1.364	0.623	5.550	-1.527	0.139	3.101	-1.277	0
4	-2.909	3.734	-6.313	-5.380	-0.887	2.062	9.446	4.490	-3.945	4.582	-8.780	-3.383	5.107	6.788	2.044	8.266	6.629	-10.069	1.223	-3.230	1.687	-2.164	-3.645	6.510	0

```
# let's view the last 5 rows of the training data
data.tail() ## Complete the code to view top 5 rows of the data
```

	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
19995	1.100	-1.199	-1.065	-0.365	3.131	-3.948	-3.578	-8.139	-1.937	-1.328	-0.403	-1.735	9.996	6.955	-3.938	-8.274	5.745	0.589	-0.650	-3.043	2.216	0.609	0.178	2.928	1
19996	1.695	1.720	3.672	-1.210	-8.222	2.121	-5.492	1.452	1.450	3.885	1.077	-0.384	-0.839	-0.748	-1.089	-4.159	1.181	-0.742	5.369	-0.893	-1.659	3.660	0.820	-1.987	0
19997	1.115	5.672	-3.924	2.133	-4.502	2.777	5.728	1.620	-1.700	-0.042	-2.923	-2.760	-2.254	2.552	0.982	7.112	1.476	-3.954	1.856	5.029	2.083	-6.409	1.477	-0.874	0
19998	-1.365	1.155	3.877	3.524	-7.015	-0.132	-3.446	-4.801	-0.876	-3.812	5.422	-3.732	0.609	5.256	1.915	0.403	3.164	3.752	8.530	8.451	0.204	-7.130	4.249	-6.112	0
19999	3.708	2.893	2.329	1.458	-6.429	1.818	0.806	7.766	0.331	5.257	-4.867	-0.819	-5.667	-2.861	4.674	6.821	-1.969	-1.349	3.952	5.450	-0.455	-2.202	1.678	-1.974	0

- The training data has 20,000 rows and 41 columns
- The test data has 5,000 rows and 41 columns
- From the few rows displayed, it is already apparent that the positive cases (target = 1) constitute a minority class

Data Background Check (2/3)

```
data.info()
```

```
<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
 4   V5      20000 non-null  float64
 5   V6      20000 non-null  float64
 6   V7      20000 non-null  float64
 7   V8      20000 non-null  float64
 8   V9      20000 non-null  float64
 9   V10     20000 non-null  float64
 10  V11     20000 non-null  float64
 11  V12     20000 non-null  float64
 12  V13     20000 non-null  float64
 13  V14     20000 non-null  float64
 14  V15     20000 non-null  float64
 15  V16     20000 non-null  float64
 16  V17     20000 non-null  float64
 17  V18     20000 non-null  float64
 18  V19     20000 non-null  float64
 19  V20     20000 non-null  float64
 20  V21     20000 non-null  float64
 21  V22     20000 non-null  float64
 22  V23     20000 non-null  float64
 23  V24     20000 non-null  float64
 24  V25     20000 non-null  float64
 25  V26     20000 non-null  float64
 26  V27     20000 non-null  float64
 27  V28     20000 non-null  float64
 28  V29     20000 non-null  float64
 29  V30     20000 non-null  float64
 30  V31     20000 non-null  float64
 31  V32     20000 non-null  float64
 32  V33     20000 non-null  float64
 33  V34     20000 non-null  float64
 34  V35     20000 non-null  float64
 35  V36     20000 non-null  float64
 36  V37     20000 non-null  float64
 37  V38     20000 non-null  float64
 38  V39     20000 non-null  float64
 39  V40     20000 non-null  float64
 40  Target  20000 non-null  int64
dtypes: float64(40), int64(1)
memory usage: 6.3 MB
```

```
data_test.info()
```

```
<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
 5   V6      5000 non-null   float64
 6   V7      5000 non-null   float64
 7   V8      5000 non-null   float64
 8   V9      5000 non-null   float64
 9   V10     5000 non-null   float64
 10  V11     5000 non-null   float64
 11  V12     5000 non-null   float64
 12  V13     5000 non-null   float64
 13  V14     5000 non-null   float64
 14  V15     5000 non-null   float64
 15  V16     5000 non-null   float64
 16  V17     5000 non-null   float64
 17  V18     5000 non-null   float64
 18  V19     5000 non-null   float64
 19  V20     5000 non-null   float64
 20  V21     5000 non-null   float64
 21  V22     5000 non-null   float64
 22  V23     5000 non-null   float64
 23  V24     5000 non-null   float64
 24  V25     5000 non-null   float64
 25  V26     5000 non-null   float64
 26  V27     5000 non-null   float64
 27  V28     5000 non-null   float64
 28  V29     5000 non-null   float64
 29  V30     5000 non-null   float64
 30  V31     5000 non-null   float64
 31  V32     5000 non-null   float64
 32  V33     5000 non-null   float64
 33  V34     5000 non-null   float64
 34  V35     5000 non-null   float64
 35  V36     5000 non-null   float64
 36  V37     5000 non-null   float64
 37  V38     5000 non-null   float64
 38  V39     5000 non-null   float64
 39  V40     5000 non-null   float64
 40  Target  5000 non-null   int64
dtypes: float64(40), int64(1)
memory usage: 1.6 MB
```

```
data.isnull().sum()
```

```
V1        18
V2        18
V3         0
V4         0
V5         0
V6         0
V7         0
V8         0
V9         0
V10        0
V11        0
V12        0
V13        0
V14        0
V15        0
V16        0
V17        0
V18        0
V19        0
V20        0
V21        0
V22        0
V23        0
V24        0
V25        0
V26        0
V27        0
V28        0
V29        0
V30        0
V31        0
V32        0
V33        0
V34        0
V35        0
V36        0
V37        0
V38        0
V39        0
V40        0
Target     0
dtype: int64
```

```
data_test.isnull().sum()
```

```
V1        5
V2        6
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
V29       0
V30       0
V31       0
V32       0
V33       0
V34       0
V35       0
V36       0
V37       0
V38       0
V39       0
V40       0
Target    0
dtype: int64
```

```
[ ] # let's check for duplicate values in the training data
    data.duplicated().sum() ##  Complete the code to check duplicate entries in the data
```

```
0
```

```
[ ] # let's check for duplicate values in the test data
    data_test.duplicated().sum() ##  Complete the code to check duplicate entries in the data
```

```
0
```

- The training and test data sets have only numerical columns (40 float and 1 int)
- Columns V1 and V2 contain some missing values
- Neither the training data nor the test data has duplicated data
- In the training data, columns V1 and V2 have 18 missing values each
- In the test data, column V1 has 5 missing values whereas column V2 has 6 missing values

Data Background Check (3/3)

```
data.describe().T ##  Complete the code to print the statitical
```

	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
V5	20000.000	-0.054	2.105	-8.603	-1.536	-0.102	1.340	8.134
V6	20000.000	-0.995	2.041	-10.227	-2.347	-1.001	0.380	6.976
V7	20000.000	-0.879	1.762	-7.950	-2.031	-0.917	0.224	8.006
V8	20000.000	-0.548	3.296	-15.658	-2.643	-0.389	1.723	11.679
V9	20000.000	-0.017	2.161	-8.596	-1.495	-0.068	1.409	8.138
V10	20000.000	-0.013	2.193	-9.854	-1.411	0.101	1.477	8.108
V11	20000.000	-1.895	3.124	-14.832	-3.922	-1.921	0.119	11.826
V12	20000.000	1.605	2.930	-12.948	-0.397	1.508	3.571	15.081
V13	20000.000	1.580	2.875	-13.228	-0.224	1.637	3.460	15.420
V14	20000.000	-0.951	1.790	-7.739	-2.171	-0.957	0.271	5.671
V15	20000.000	-2.415	3.355	-16.417	-4.415	-2.383	-0.359	12.246
V16	20000.000	-2.925	4.222	-20.374	-5.634	-2.683	-0.095	13.583
V17	20000.000	-0.134	3.345	-14.091	-2.216	-0.015	2.069	16.756
V18	20000.000	1.189	2.592	-11.644	-0.404	0.883	2.572	13.180
V19	20000.000	1.182	3.397	-13.492	-1.050	1.279	3.493	13.238
V20	20000.000	0.024	3.669	-13.923	-2.433	0.033	2.512	16.052
V21	20000.000	-3.611	3.568	-17.956	-5.930	-3.533	-1.266	13.840
V22	20000.000	0.952	1.652	-10.122	-0.118	0.975	2.026	7.410
V23	20000.000	-0.366	4.032	-14.866	-3.099	-0.262	2.452	14.459
V24	20000.000	1.134	3.912	-16.387	-1.468	0.969	3.546	17.163
V25	20000.000	-0.002	2.017	-8.228	-1.365	0.025	1.397	8.223
V26	20000.000	1.874	3.435	-11.834	-0.338	1.951	4.130	16.836
V27	20000.000	-0.612	4.369	-14.905	-3.652	-0.885	2.189	17.560
V28	20000.000	-0.883	1.918	-9.269	-2.171	-0.891	0.376	6.528
V29	20000.000	-0.986	2.684	-12.579	-2.787	-1.176	0.630	10.722
V30	20000.000	-0.016	3.005	-14.796	-1.867	0.184	2.036	12.506
V31	20000.000	0.487	3.461	-13.723	-1.818	0.490	2.731	17.255
V32	20000.000	0.304	5.500	-19.877	-3.420	0.052	3.762	23.633
V33	20000.000	0.050	3.575	-16.898	-2.243	-0.066	2.255	16.692
V34	20000.000	-0.463	3.184	-17.985	-2.137	-0.255	1.437	14.358
V35	20000.000	2.230	2.937	-15.350	0.336	2.099	4.064	15.291
V36	20000.000	1.515	3.801	-14.833	-0.944	1.567	3.984	19.330
V37	20000.000	0.011	1.788	-5.478	-1.256	-0.128	1.176	7.467
V38	20000.000	-0.344	3.948	-17.375	-2.988	-0.317	2.279	15.290
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

```
data_test.describe().T ##  Complete the code to print the statitical
```

	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
V5	5000.000	-0.080	2.111	-7.712	-1.615	-0.132	1.341	7.673
V6	5000.000	-1.042	2.005	-8.924	-2.369	-1.049	0.308	5.068
V7	5000.000	-0.908	1.769	-8.124	-2.054	-0.940	0.212	7.616
V8	5000.000	-0.575	3.332	-12.253	-2.642	-0.358	1.713	10.415
V9	5000.000	0.030	2.174	-6.785	-1.456	-0.080	1.450	8.851
V10	5000.000	0.019	2.145	-8.171	-1.353	0.166	1.511	6.599
V11	5000.000	-2.009	3.112	-13.152	-4.050	-2.043	0.044	9.956
V12	5000.000	1.576	2.907	-8.164	-0.450	1.488	3.563	12.984
V13	5000.000	1.622	2.883	-11.548	-0.126	1.719	3.465	12.620
V14	5000.000	-0.921	1.803	-7.814	-2.111	-0.896	0.272	5.734
V15	5000.000	-2.452	3.387	-15.286	-4.479	-2.417	-0.433	11.673
V16	5000.000	-3.019	4.264	-20.986	-5.648	-2.774	-0.178	13.976
V17	5000.000	-0.104	3.337	-13.418	-2.228	0.047	2.112	19.777
V18	5000.000	1.196	2.586	-12.214	-0.409	0.881	2.604	13.642
V19	5000.000	1.210	3.385	-14.170	-1.026	1.296	3.526	12.428
V20	5000.000	0.138	3.657	-13.720	-2.325	0.193	2.540	13.871
V21	5000.000	-3.664	3.578	-16.341	-5.944	-3.663	-1.330	11.047
V22	5000.000	0.962	1.640	-6.740	-0.048	0.986	2.029	7.505
V23	5000.000	-0.422	4.057	-14.422	-3.163	-0.279	2.426	13.181
V24	5000.000	1.089	3.968	-12.316	-1.623	0.913	3.537	17.806
V25	5000.000	0.061	2.010	-6.770	-1.298	0.077	1.428	6.557
V26	5000.000	1.847	3.400	-11.414	-0.242	1.917	4.156	17.528
V27	5000.000	-0.552	4.403	-13.177	-3.663	-0.872	2.247	17.290
V28	5000.000	-0.868	1.926	-7.933	-2.160	-0.931	0.421	7.416
V29	5000.000	-1.096	2.655	-9.988	-2.861	-1.341	0.522	14.039
V30	5000.000	-0.119	3.023	-12.438	-1.997	0.112	1.946	10.315
V31	5000.000	0.469	3.446	-11.263	-1.822	0.486	2.779	12.559
V32	5000.000	0.233	5.586	-17.244	-3.556	-0.077	3.752	26.539
V33	5000.000	-0.080	3.539	-14.904	-2.348	-0.160	2.099	13.324
V34	5000.000	-0.393	3.166	-14.700	-2.010	-0.172	1.465	12.146
V35	5000.000	2.211	2.948	-12.261	0.322	2.112	4.032	13.489
V36	5000.000	1.595	3.775	-12.736	-0.866	1.703	4.104	17.116
V37	5000.000	0.023	1.785	-5.079	-1.241	-0.110	1.238	6.810
V38	5000.000	-0.406	3.969	-15.335	-2.984	-0.381	2.288	13.065
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

- From the statistical summary of both datasets, it can be observed that the mean and median of most, if not all, of the variables are close and almost midway the respective extreme values, suggesting that, for these columns, the data is not very skewed

Model Performance Summary (original data)

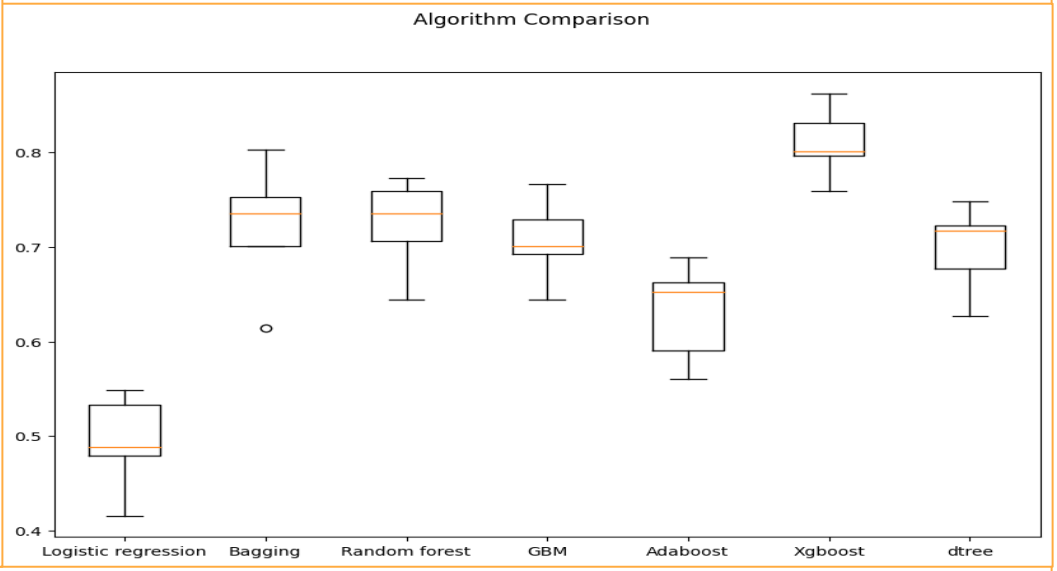
```
Cross-Validation performance on training dataset:

Logistic regression: 0.4927566553639709
Bagging: 0.7210807301060529
Random forest: 0.7235192266070268
GBM: 0.7066661857008874
Adaboost: 0.6309140754635308
Xgboost: 0.8100497799581561
dtree: 0.6982829521679532

Validation Performance:

Logistic regression: 0.48201438848920863
Bagging: 0.7302158273381295
Random forest: 0.7266187050359713
GBM: 0.7230215827338129
Adaboost: 0.6762589928057554
Xgboost: 0.8309352517985612
dtree: 0.7050359712230215
```

Graphic Display of Cross-Validation Performance



- The three best performing models on the original data are XGBoost, Bagging, and Random Forest

Link to Appendix slide on model assumptions

Model Performance Summary (oversampled data)

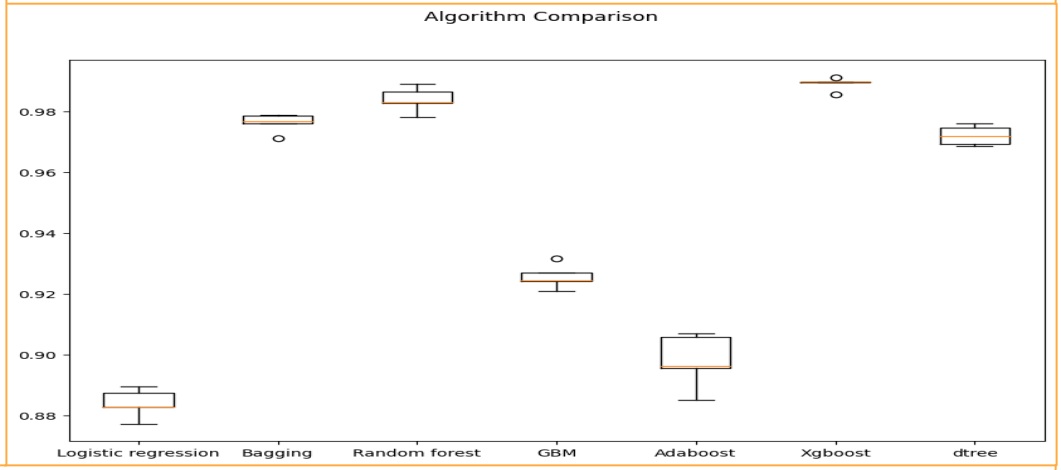
```
Cross-Validation performance on training dataset:

Logistic regression: 0.883963699328486
Bagging: 0.9762141471581656
Random forest: 0.9839075260047615
GBM: 0.9256068151319724
Adaboost: 0.8978689011775473
Xgboost: 0.9891305241357218
dtree: 0.9720494245534969

Validation Performance:

Logistic regression: 0.8489208633093526
Bagging: 0.8345323741007195
Random forest: 0.8489208633093526
GBM: 0.8776978417266187
Adaboost: 0.8561151079136691
Xgboost: 0.8669064748201439
dtree: 0.7769784172661871
```

Graphic Display of Cross-Validation Performance



- The models with the three best cross-validation performance on the oversampled training data are XGBoost, Random Forest, and Bagging
- Among the models trained on the oversampled training data, those with the three best performance on the validation dataset are GBM, XGBoost, and AdaBoost

Link to Appendix slide on model assumptions

Model Performance Summary (undersampled data)

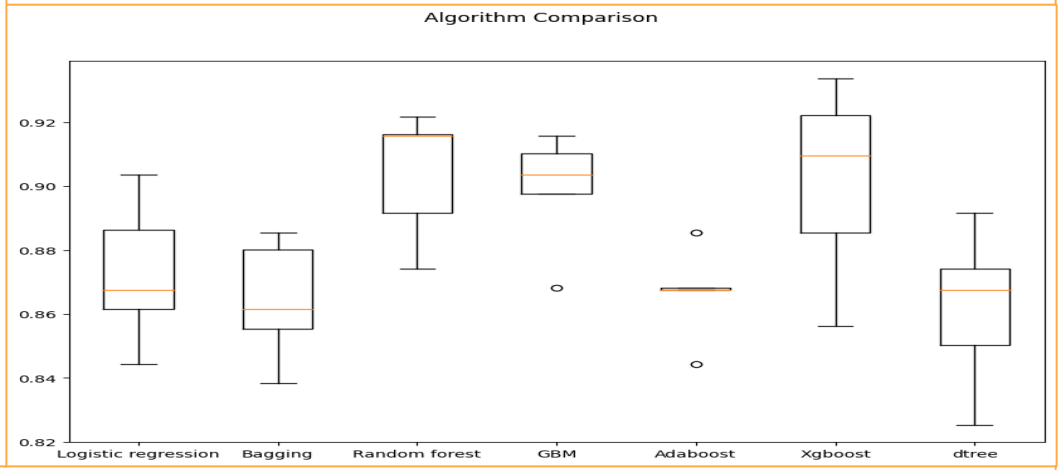
```
Cross-Validation performance on training dataset:

Logistic regression: 0.8726138085275232
Bagging: 0.8641945025611427
Random forest: 0.9038669648654498
GBM: 0.8990621167303946
Adaboost: 0.8666113556020489
Xgboost: 0.9014717552846114
dtree: 0.8617776495202367

Validation Performance:

Logistic regression: 0.8525179856115108
Bagging: 0.8705035971223022
Random forest: 0.8920863309352518
GBM: 0.8884892086330936
Adaboost: 0.8489208633093526
Xgboost: 0.89568345323741
dtree: 0.841726618705036
```

Graphic Display of Cross-Validation Performance



- The models with the three best cross-validation performance on the undersampled training data are Random Forest, XGBoost, and GBM
- Among the models trained on the undersampled training data, those with the three best performance on the validation dataset are XGBoost, Random Forest, and GBM (same set of cross-validation best performing models)

Link to Appendix slide on model assumptions

Model Assumption (Model evaluation criterion)

Interpretations of predictions made by each classification model:

- 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.

Most important metric:

- 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.

Great Learning

Happy Learning !