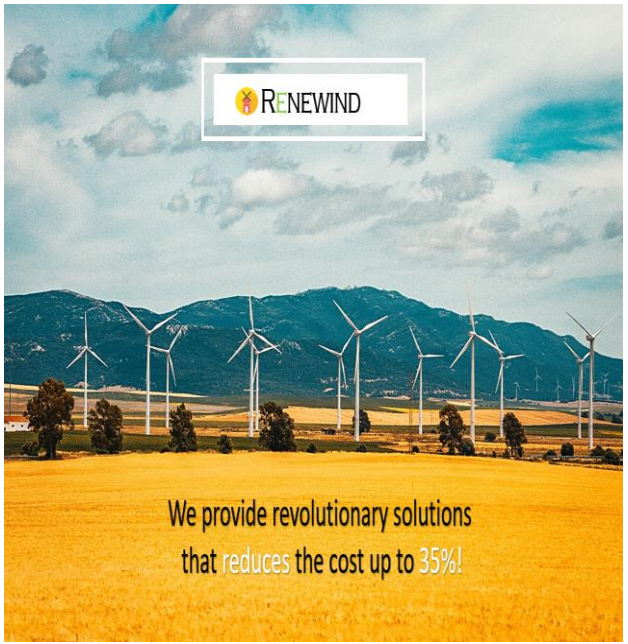
ReneWind

Data based prediction model : Reduce Maintenance cost

Background

1. Predictive maintenance uses sensor information and analysis methods to measure and predict degradation and future component capability.
2. 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 will be much lower.
3. The sensors fitted across different machines involved in the process of energy generation collect data related to various environmental factors (temperature, humidity, wind speed, etc.) and additional features related to various parts of the wind turbine (gearbox, tower, blades, break, etc.).
4. “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.



Business Problem Overview and Solution Approach

"ReneWind" has shared a ciphered version of the data, Data has 40 predictors, 40000 observations in the training set and 10000 in the test set.

- **Objective**

  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 and the overall maintenance cost of the generators can be brought down.

"1" in the target variables should be considered as "failure" and "0" will represent "No failure".
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 the generator of wind turbine where there is no detection by the model.
False positives (FP) are failure detections in the generator of the wind turbine where actually there is no failure.

So, the maintenance cost associated with the model would be:
Maintenance cost = TP*(Repair cost) + FN*(Replacement cost) + FP*(Inspection cost)

Business Problem Overview and Solution Approach

So, the maintenance cost associated with the model would be:

Maintenance cost = TP*(Repair cost) + FN*(Replacement cost) + FP*(Inspection cost)

where,

Replacement cost = $40,000

Repair cost = $15,000

Inspection cost = $5,000

Here the objective is to reduce the maintenance cost so, we want a metric that could reduce the maintenance cost.

The minimum possible maintenance cost = **Actual failures*(Repair cost) = (TP + FN)*(Repair cost)**

And the maintenance cost associated with model = TP*(Repair cost) + FN*(Replacement cost) + FP*(Inspection cost)
So, we will try to maximize the ratio of minimum possible maintenance cost and the maintenance cost associated with the model.

The value of this ratio will lie **between 0 and 1**, the ratio will be 1 only when the maintenance cost associated with the model will be equal to the minimum possible maintenance cost.

Data Overview

Variable	Description
40 Columns	ciphered version of the data, as the data collected through sensors is confidential

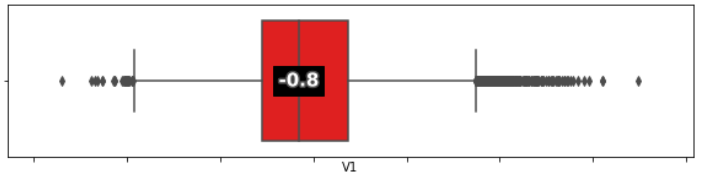
Observations	Variables
Training set 40000	40
Test set **10000**	**40**

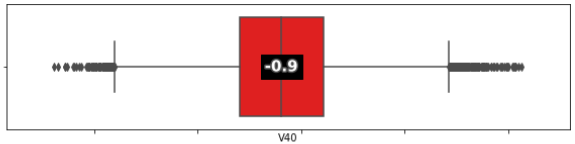
Missing values

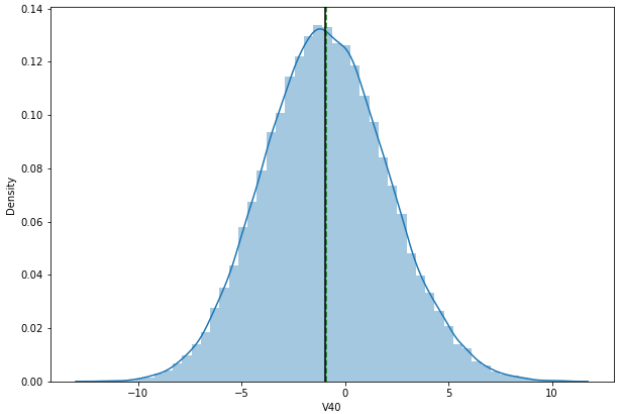
The data set has missing values and, in this case, the mean imputer is used to impute missing values

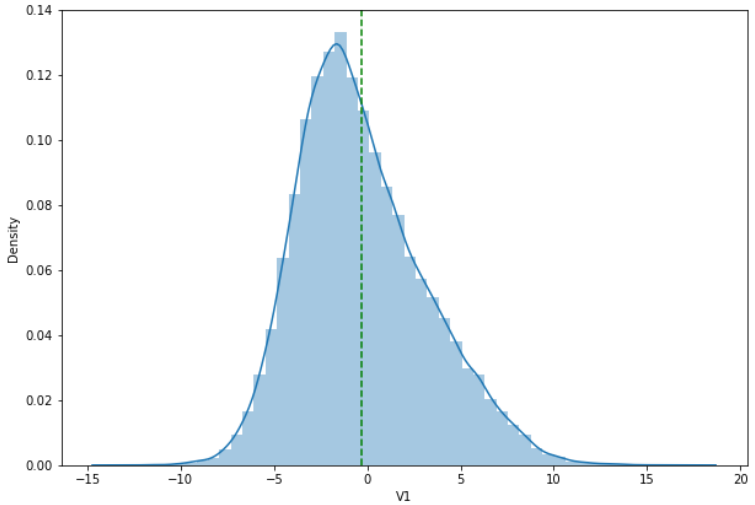
Exploratory Data Analysis

- Most of the variables exhibit almost near normal distribution.
- The distribution of V1 and V40 as an example.



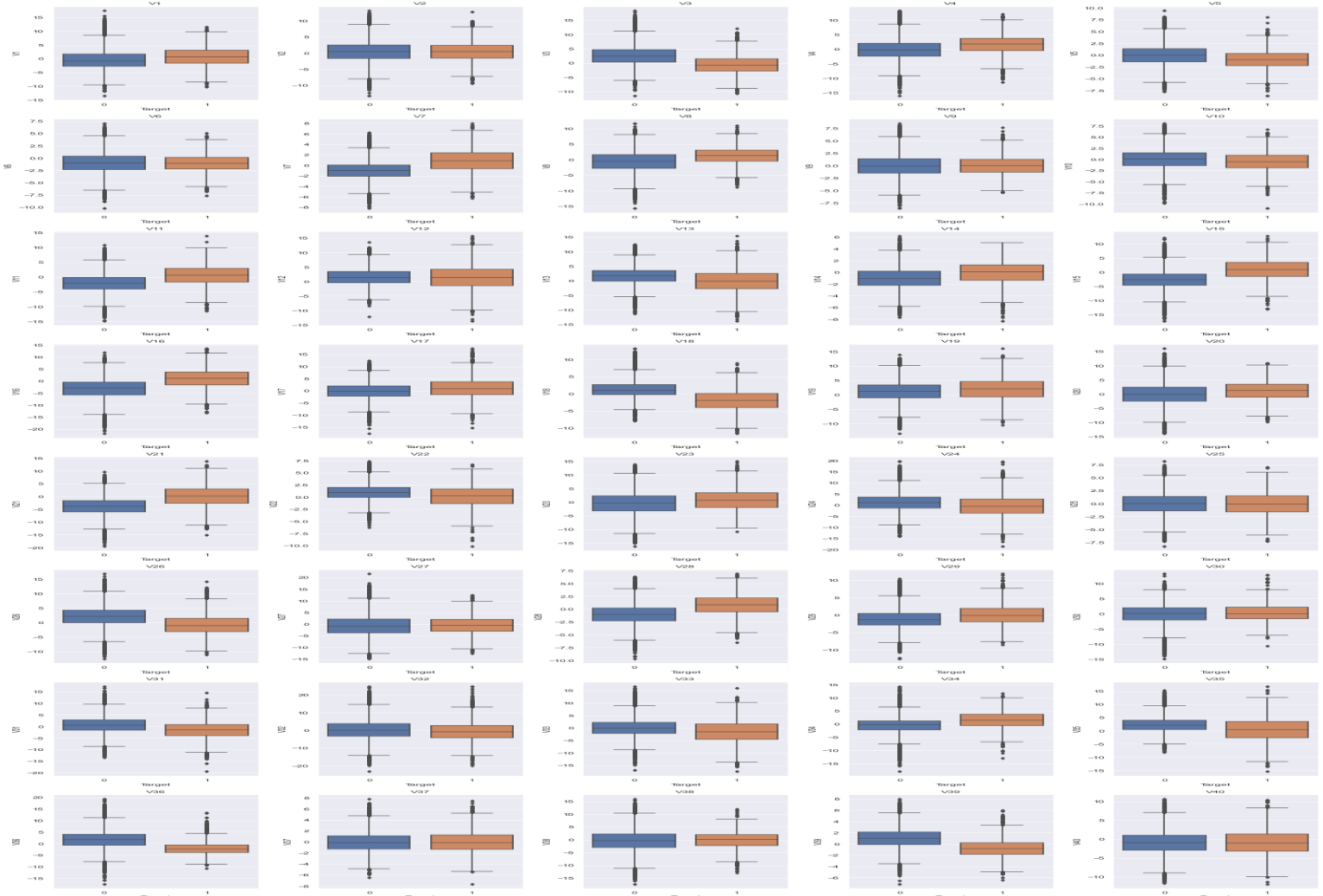






EDA

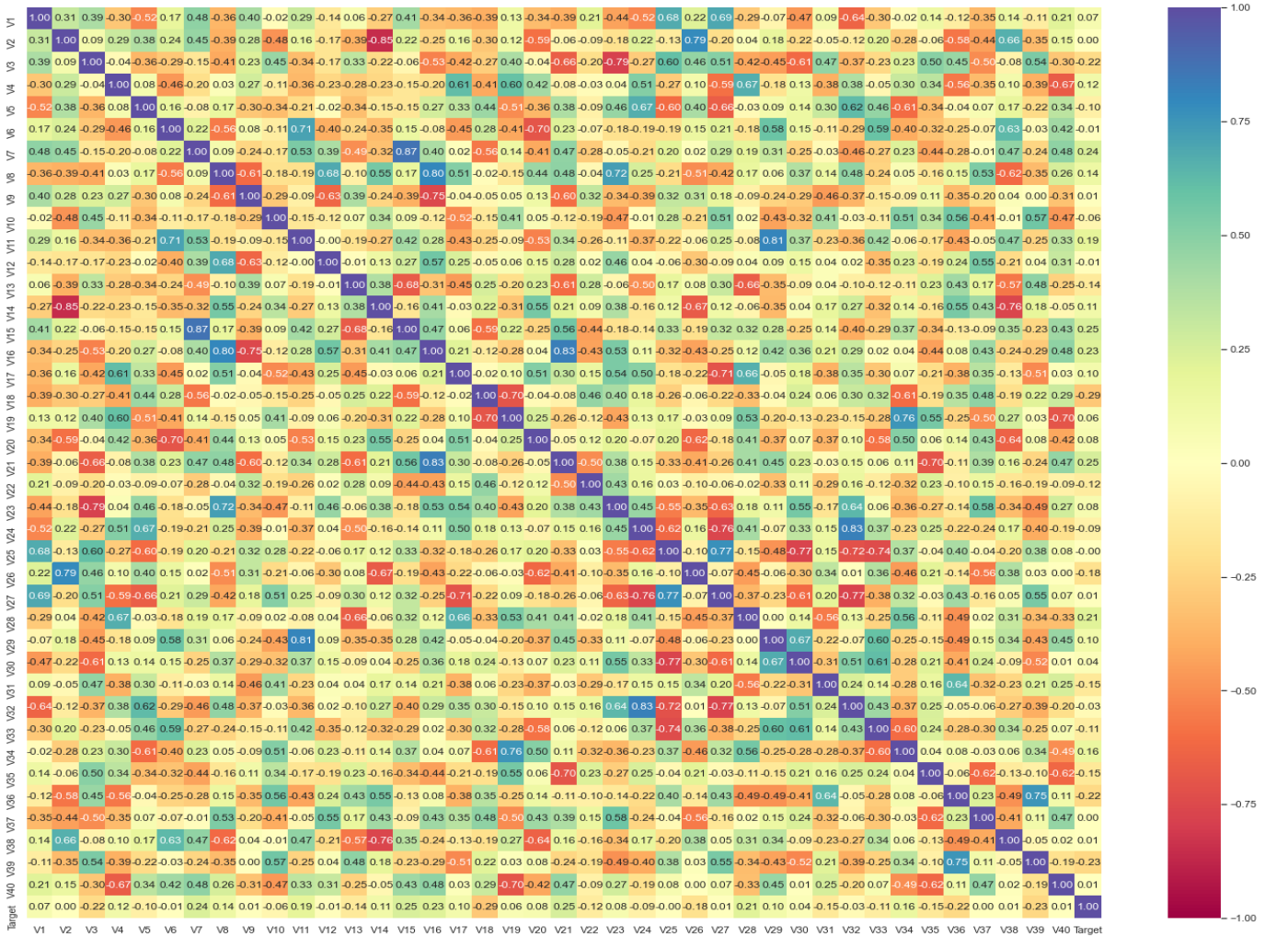
1. All the 40 variable have a near normal distribution.
2. There are outliers.
3. No specific treatment for the outliers as they are part of the natural data.



EDA

1. There is no specific correlation of dependent variables w.r.t to Target variable.

2. There are some variable that have strong positive and negative correlation.



Model Building – The data set is imbalanced

```
print("Number of rows in train data =", X_train.shape[0])
print("Number of rows in validation data =", X_val.shape[0])
```

```
Number of rows in train data = 28000
Number of rows in validation data = 12000
```

```
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
print("Percentage of classes in test set:")
print(y_val.value_counts(normalize=True))
```

```
Percentage of classes in training set:
0    0.944893
1    0.055107
Name: Target, dtype: float64
Percentage of classes in test set:
0    0.946333
1    0.053667
Name: Target, dtype: float64
```

Model Building – With Original Data

Cross-Validation Performance: cv=10

On Training Data

LogisticRegression:	52.35892533471536
DecisionTree:	65.10054473324213
Random forest:	71.53861239850332
Bagging:	69.43885217772146
Adaboost:	58.79589095944772
GBM:	68.1172554037155
Xgboost:	77.45266458451022

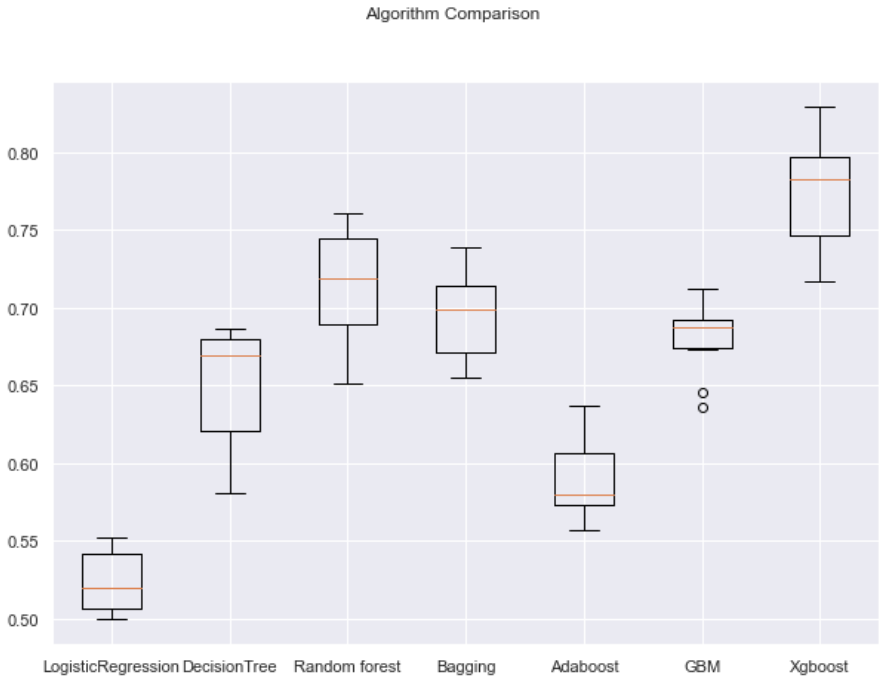
Performance on Validation data:

LogisticRegression:	53.047775947281714
DecisionTree:	67.2
Random forest:	72.93318233295584
Bagging:	70.38251366120218
Adaboost:	60.2996254681648
GBM:	68.58359957401491
Xgboost:	77.52808988764045

Model Building – Algorithm Comparison – With original data

1. XGBOOST is giving the highest cross-validated recall followed by RandomForest.

2. RandomForest and XGBoost can be potential candidates for performance tuning.



Model Building – With Over Sampled Data

Applying:
Synthetic Minority Over Sampling Technique

Model Building with Over sampled data

- Before Over Sampling the data, the class distribution is

```
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
```

```
Percentage of classes in training set:
0    0.944893
1    0.055107
Name: Target, dtype: float64
```

```
# Synthetic Minority Over Sampling Technique
sm = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=1)
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)
```

- After Over Sampling the data, the class distribution is

```
print("Percentage of classes in training set:")
print(y_train_over.value_counts(normalize=True))
```

```
Percentage of classes in training set:
0    0.5
1    0.5
Name: Target, dtype: float64
```

```
X_train_over.shape
```

```
(52914, 40)
```

Model Building – With Over Sampled Data

Cross-Validation Performance: cv=10

On Training Data

LogisticRegression:	79.40984786587808
DecisionTree:	94.01861217925251
Random forest:	97.1952035999941
Bagging:	95.75355501921703
Adaboost:	83.19322841604645
GBM:	87.28991668797381
Xgboost:	97.58309648832963

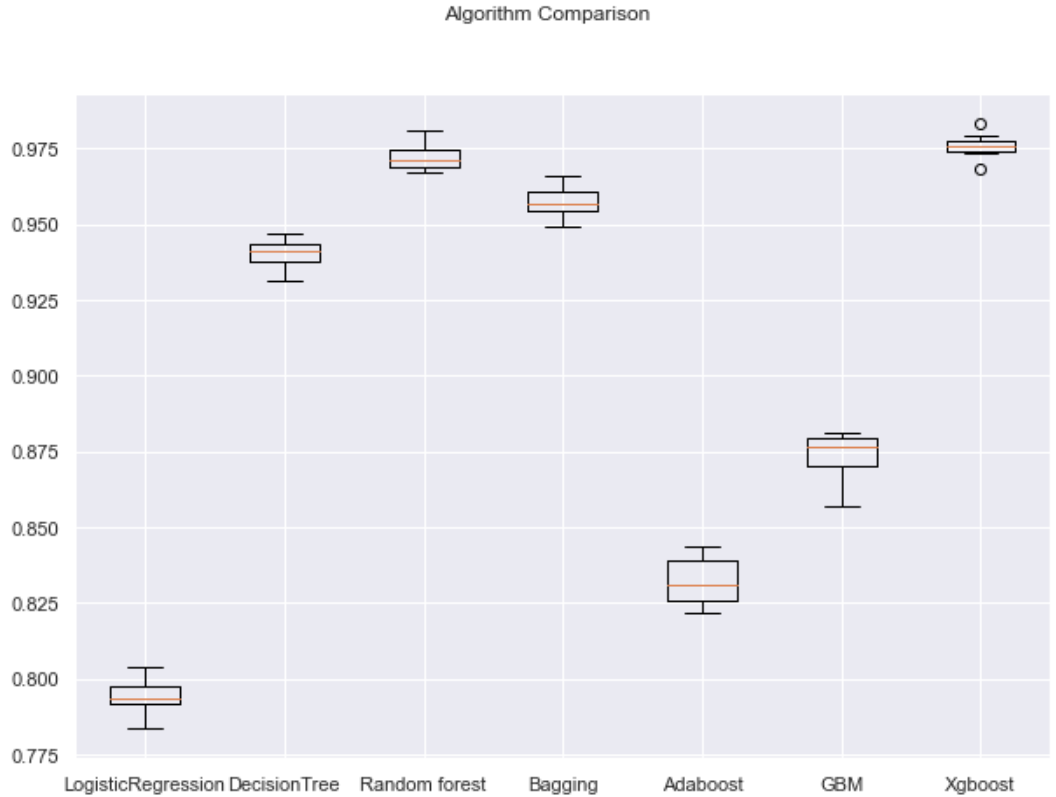
Performance on Validation data:

LogisticRegression:	50.536228093120585
DecisionTree:	63.240589198036
Random forest:	80.836820083682
Bagging:	74.88372093023256
Adaboost:	57.79240203410111
GBM:	73.20954907161804
Xgboost:	80.66805845511482

Model Building – Algorithm Comparison – With Over Sampled Data

1. XGBOOST is giving the highest cross-validated recall followed by RandomForest.

2. RandomForest and XGBoost can be potential candidates for performance tuning.



Model Building – With Over Under Sampled Data

Applying:
Random under sampler for under sampling the data

Model Building with Under sampled data

- Before Under Sampling the data, the class distribution is

```
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
```

```
Percentage of classes in training set:
0    0.944893
1    0.055107
Name: Target, dtype: float64
```

```
# Random undersampler for under sampling the data
rus = RandomUnderSampler(random_state=1, sampling_strategy=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

- After Under Sampling the data, the class distribution is

```
print("Percentage of classes in training set:")
print(y_train_un.value_counts(normalize=True))
```

```
Percentage of classes in training set:
0    0.5
1    0.5
Name: Target, dtype: float64
```

Model Building – With Under Sampled Data

Cross-Validation Performance: cv=10

On Training Data

LogisticRegression:	77.96837541564223
DecisionTree:	77.62112457810912
Random forest:	84.84438354374095
Bagging:	81.74829074330015
Adaboost:	80.08772346318565
GBM:	83.40938371506861
Xgboost:	84.87637818095986

Performance on Validation data:

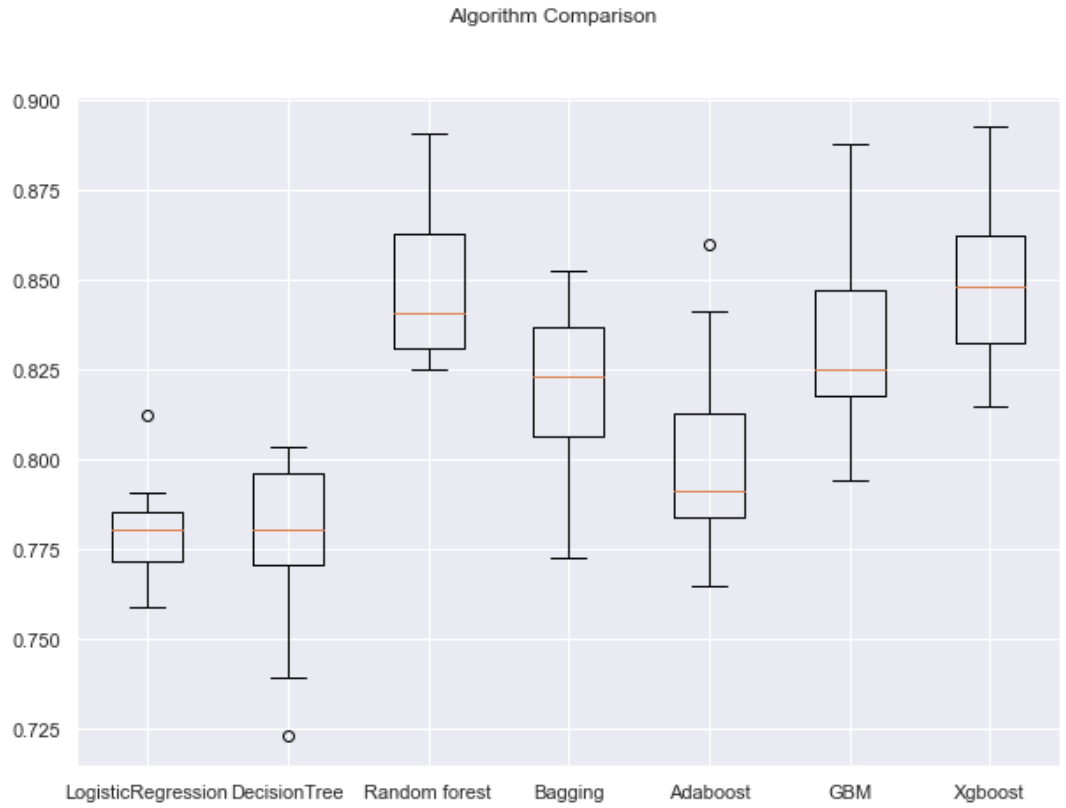
LogisticRegression:	48.80020207123011
DecisionTree:	45.967166309778726
Random forest:	71.92851824274014
Bagging:	67.01352757544224
Adaboost:	53.28185328185329
GBM:	65.33648968549205
Xgboost:	75.1458576429405

There is an overfitting on almost all the models.

Model Building – Algorithm Comparison – With Under Sampled Data

1. XGBOOST is giving the highest cross-validated recall followed by RandomForest.

2. RandomForest and XGBoost can be potential candidates for performance tuning.



Model Selection

- Lets select a mix of models from original Data, over Sampled Data and UnderSampled Data
- 1 'Random forest_original_data'
- 2 'GBM_oversampled'
- 3 'Xgboost_undersampled'
- The reason is of the models from original Data, over Sampled Data and UnderSampled Data the above three have generalized behaviour (The difference between training and validation is less)

- We will also select other models that have good performance and have metric near to .78 before tuning
- so the final list of model selected for tuning

Model	Training	Validation
* 1-Random forest_original_data	71.53861239850332	72.93318233295584
* 2-GBM_oversampled	87.28991668797381	73.20954907161804
* 3-Xgboost_undersampled	84.87637818095986	75.1458576429405
* 4-Xgboost with original data	77.45266458451022	77.52808988764045
* 5-Random forest with oversampled data	97.1952035999941	80.836820083682
* 6-Xgboost with over sampled data	97.58309648832963	80.66805845511482

1- Random Forest - With Original Data - Tuning

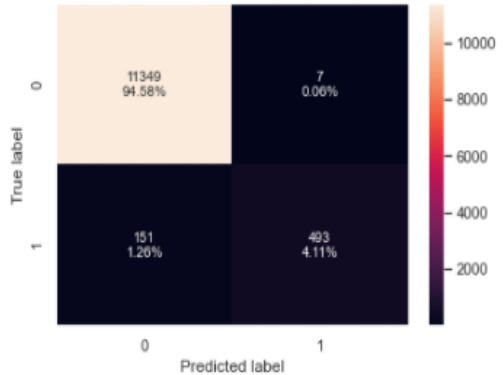
```
Best parameters are {'max_features': 'sqrt', 'max_samples': 0.5000000000000001, 'min_samples_leaf':
1, 'n_estimators': 150} with CV score=0.6920011507337778:
```

Training performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.99325	0.879456	0.997794	0.934895	0.832255

Validation performance:

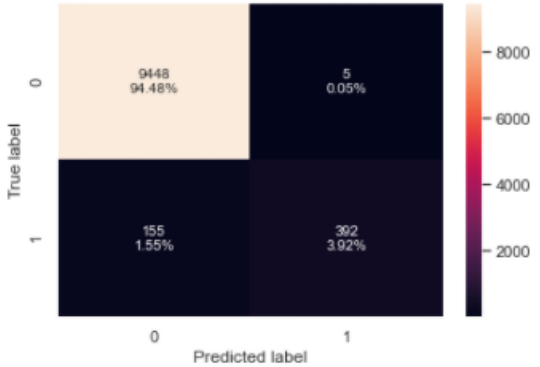
	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.986833	0.765528	0.986	0.861888	0.717149



Test performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.984	0.716636	0.987406	0.830508	0.677819

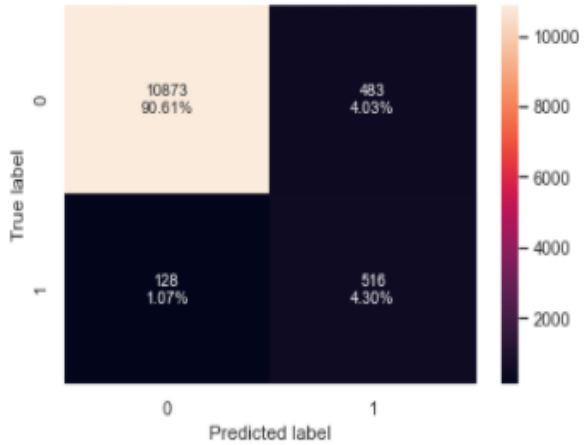
```
#Creating confusion matrix
confusion_matrix_sklearn(rf_estimator_tuned, X_test, y_test)
```



2-Gradient Boost Classifier - With Oversampled Data - Tuning

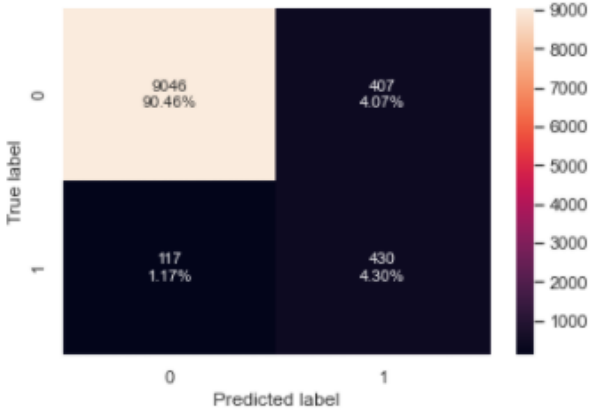
```
Best parameters are {'init': DecisionTreeClassifier(random_state=1), 'learning_rate': 0.2, 'max_features': 0.5, 'n_estimators':
75, 'subsample': 0.5} with CV score=0.9375687323384628:
```

```
Training performance:
    Accuracy  Recall  Precision   F1  Minimum_Vs_Model_cost
0       1.0     1.0        1.0  1.0                   1.0
Validation performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0  0.949083  0.801242   0.516517  0.628119              0.632406
```




```
Test performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0    0.9476  0.786106    0.51374  0.621387              0.623243
```

```
#Creating confusion matrix
confusion_matrix_sklearn(gbc_tuned, X_test, y_test)
```



3- Xgboost Classifier - With Under Sampled Data - Tuning

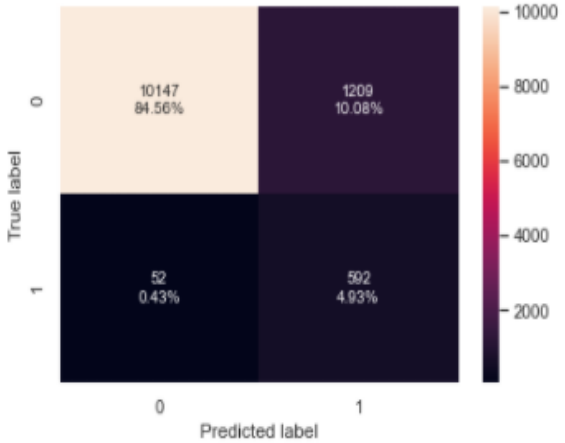
```
Best parameters are {'gamma': 5, 'learning_rate': 0.2, 'n_estimators': 150, 'scale_pos_weight': 10, 'subsample': 0.9} with CV s
core=0.8662574260510754:
```

Training performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.993843	1.0	0.987836	0.993881	0.995912

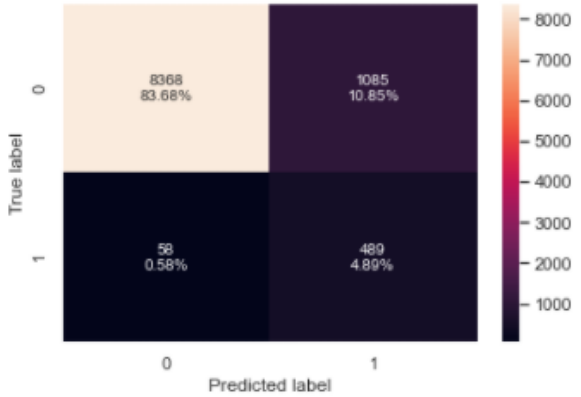
Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.894917	0.919255	0.328706	0.484254	0.568068



Test performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.8857	0.893967	0.310673	0.461103	0.544098



4- Xgboost with original data - Tuning

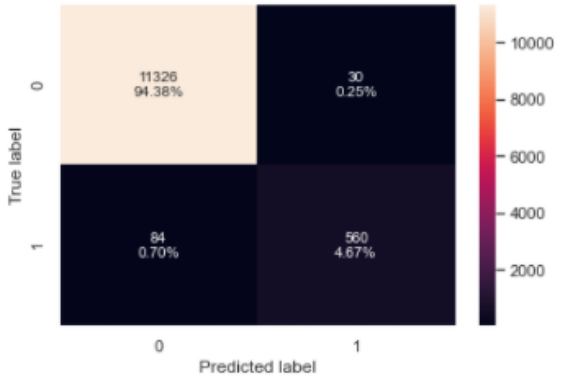
```
Best parameters are {'gamma': 5, 'learning_rate': 0.1, 'n_estimators': 250, 'scale_pos_weight': 10, 'subsample': 0.8} with CV s
core=0.8049422330772502:
```

Training performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.999321	1.0	0.987836	0.993881	0.995912

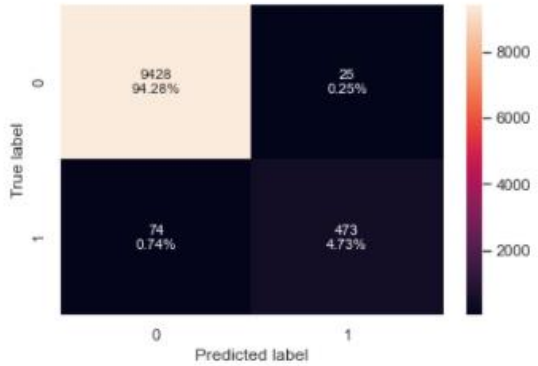
Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.9905	0.869565	0.949153	0.907618	0.811083



Test performance:

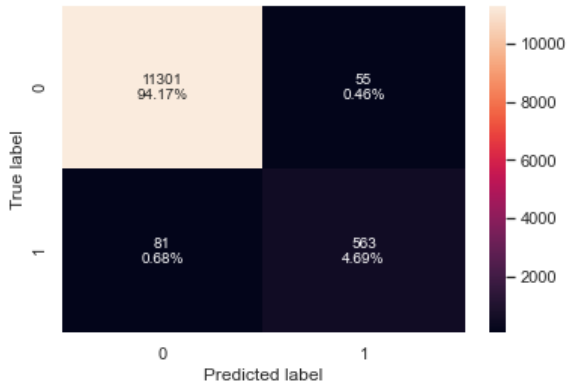
	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.9901	0.864717	0.949799	0.905263	0.805992



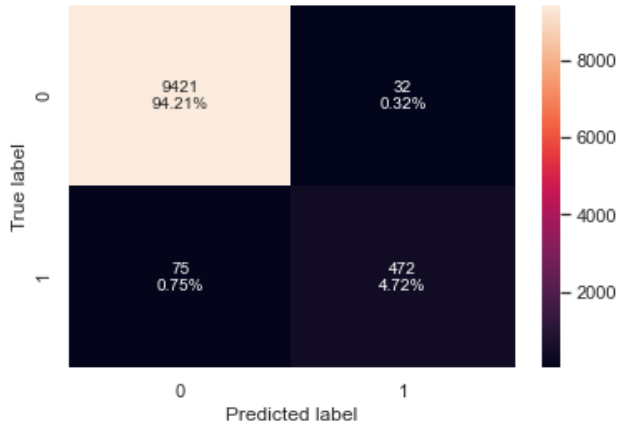
5- Random forest with over Sampled Data - Tuning

```
Best parameters are {'max_features': 'sqrt', 'max_samples': 0.5000000000000001, 'min_samples_leaf': 1, 'n_estimators': 150} wit
h CV score=0.9605285516090613:
```

```
Training performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0  0.998583  0.997505   0.999659  0.998581               0.995747
Validation performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0  0.988667  0.874224   0.911003  0.892235               0.807692
```




```
Test performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0    0.9893  0.862888   0.936508  0.898192               0.80127
```

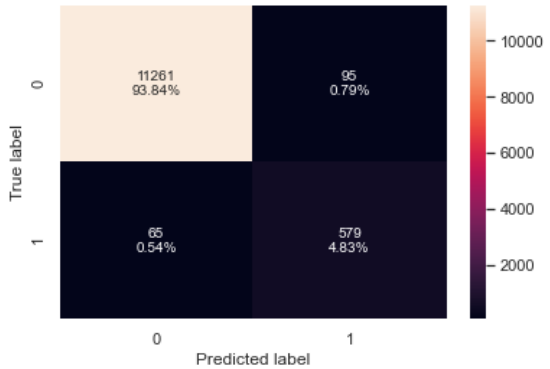


This model gives somewhat balanced performance and the metric is more than .78

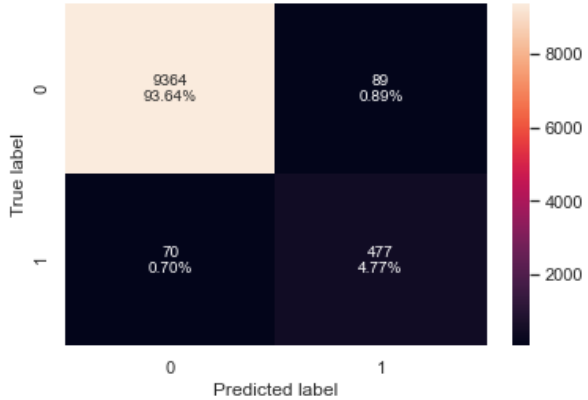
6- Xgboost with over sampled data - Tuning

```
Best parameters are {'gamma': 0, 'learning_rate': 0.2, 'n_estimators': 250, 'scale_pos_weight': 10, 'subsample': 0.9} with CV s
core=0.9887520365943508:
```

```
Training performance:
    Accuracy  Recall  Precision   F1  Minimum_Vs_Model_cost
0        1.0     1.0        1.0  1.0                   1.0
Validation performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0  0.986667  0.899068    0.85905  0.878604              0.821429
```




```
Test performance:
    Accuracy    Recall  Precision        F1  Minimum_Vs_Model_cost
0    0.9841  0.872029   0.842756  0.857143              0.788942
```

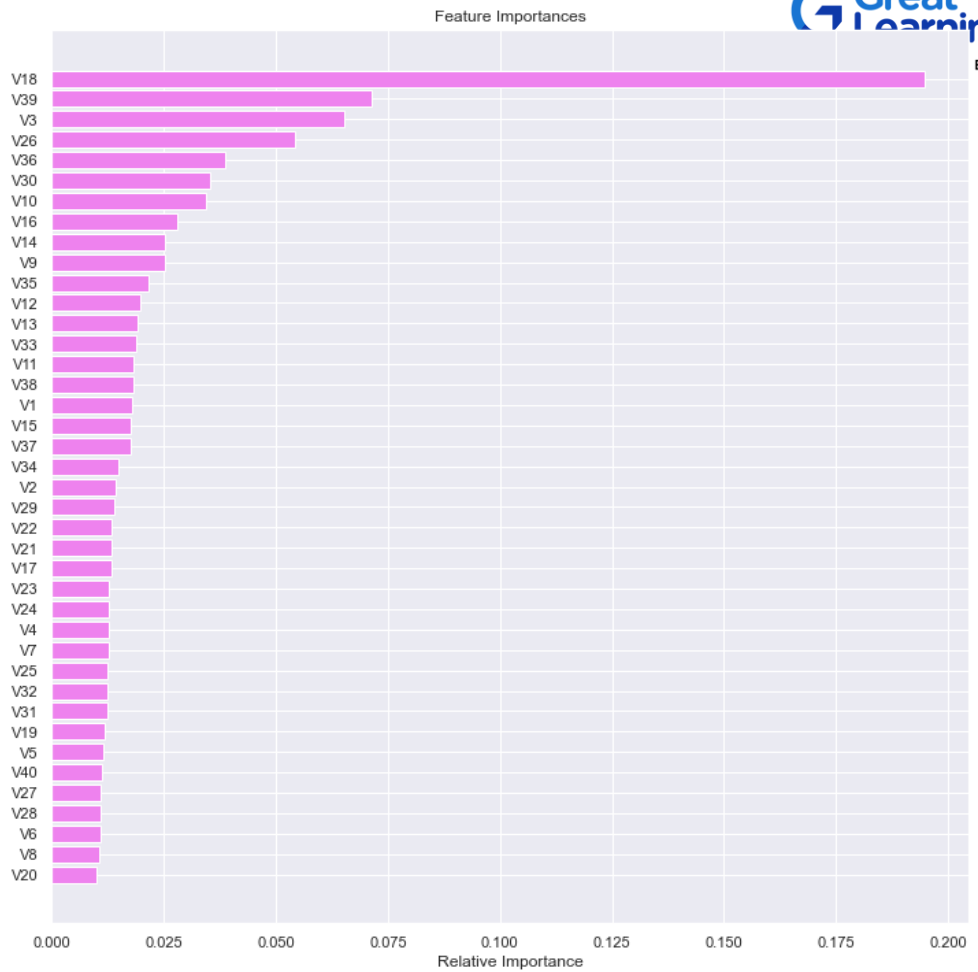


- This model gives some what balanced performance and the metric is close to .78

Feature Importance:

Xgboost with original data

Variable **V18** is the important feature



Model Performance comparison & choosing the final model

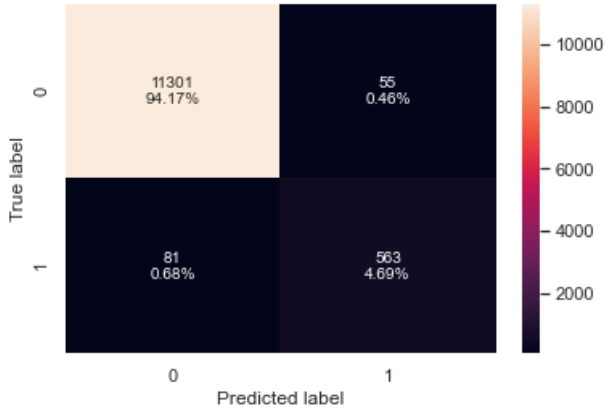
- From the metric "Minimum_Vs_Model_cost"
- The best model is **"Random forest with over sampled data"**

Training performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.998583	0.997505	0.999659	0.998581	0.995747

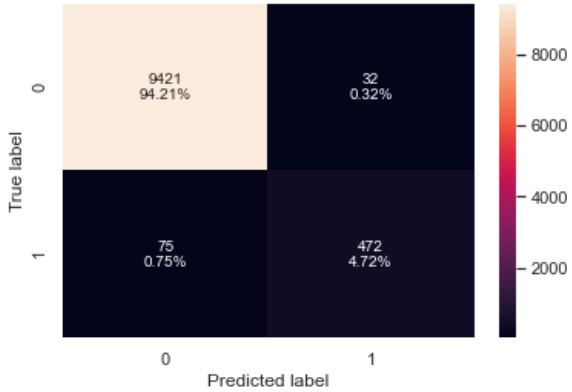
Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.988667	0.874224	0.911003	0.892235	0.807692



Test performance:

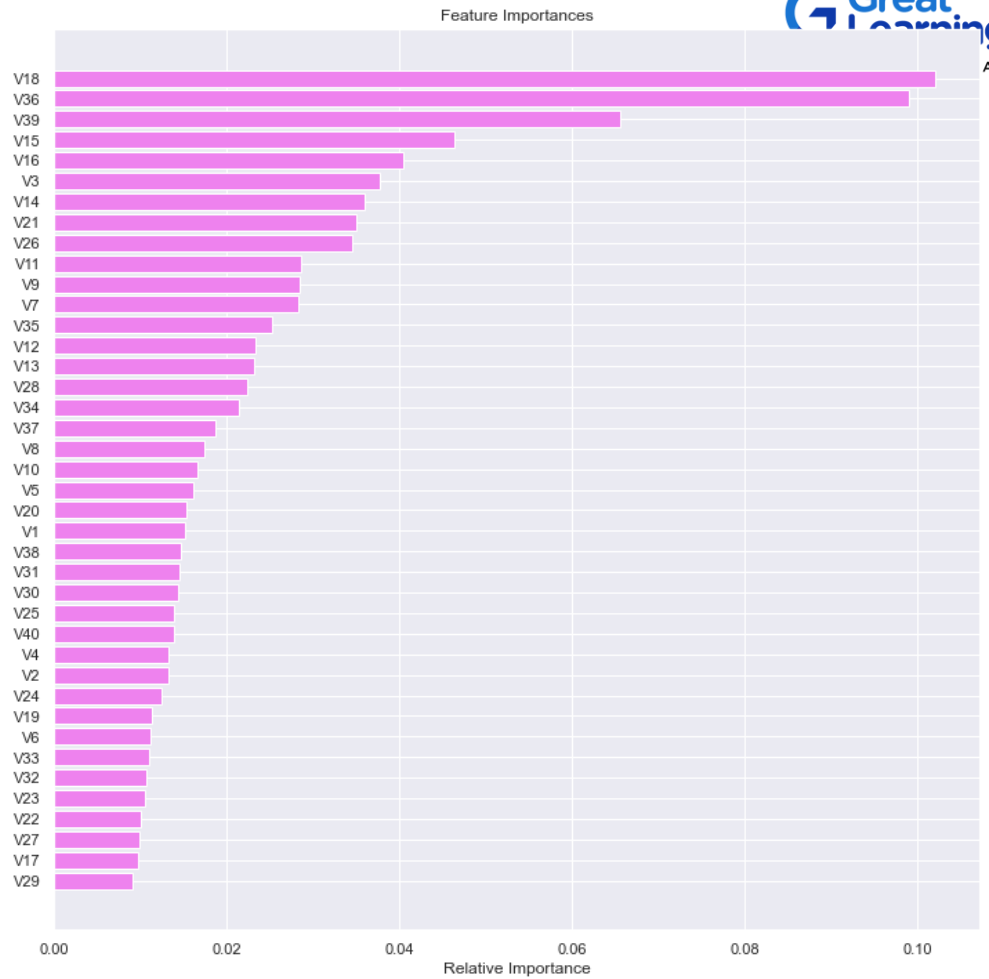
	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.9893	0.862888	0.936508	0.898192	0.80127



Feature Importance:

Random forest with over sampled data

Variable **V18** and **V36** are import features that determine the failure and helps to minimize the cost.



Pipelines to build the final model

```
numeric_transformer = Pipeline(steps=[("imputer", SimpleImputer(strategy="median"))])
```

```
# Creating new pipeline with best parameters
model = Pipeline(
    steps=[
        ("pre", numeric_transformer),
        (
            "RF_OverSampled",
             RandomForestClassifier(random_state=1,
                max_features='sqrt',
                max_samples=0.5000000000000001,
                min_samples_leaf=1,
                n_estimators=150,
            ),
        ),
    ]
)
# Fit the model on training data
model.fit(X_train_over, y_train_over)
```

```
Pipeline(steps=[('pre',
                 Pipeline(steps=[('imputer',
                                  SimpleImputer(strategy='median'))])),
                ('RF_OverSampled',
                 RandomForestClassifier(max_features='sqrt',
                                        max_samples=0.5000000000000001,
                                        n_estimators=150, random_state=1))])
```

Business Insights and Recommendations

1. Variable V18 and V36 are the important features.

2. From the model performance in test set, there is an 80% probability that failures can be identified.

3. Both the models that have more than .78 performance on test data, exhibit overfit, thus the performance in real time might be impacted.

4. Since the ratio of the failure class is less, more data can be collected on the failure class.

greatlearning
Power Ahead

Happy Learning !