This repo consists of data visualization project done for wealth management dataset from Kaggle. I have used various Machine Learning classifiers to calculate accuracy and precision to determine which model works best for this dataset. The agenda of this project is to analyze the trend of customer churn from a wealth management company.

Challenges Faced and Preliminary actions taken

- Identify primary key columns like ID, email and other irrelevant columns like name, surname etc and delete them, as these columns don't add value while analyzing data or classifying data.
- Identifying null values in certain columns, and then delete those rows or add some dummy value for that column in a row depending upon the severity of the row data.
- Identifying the datatype for each columns. If categorical data, then we have to handle it carefully and convert into numeric using various techniques like ordinal and one-hot depending on how we will compute the data.
- Identifying the various classifiers that will fit for the use case.
- Identifying the libraries required to implement those classifiers.
- Installing all the required packages to support various libraries and operations.

Dataset used:

The dataset used for this exercise can be found here *https://www.kaggle.com/filippoo/deep-learning-az-ann*

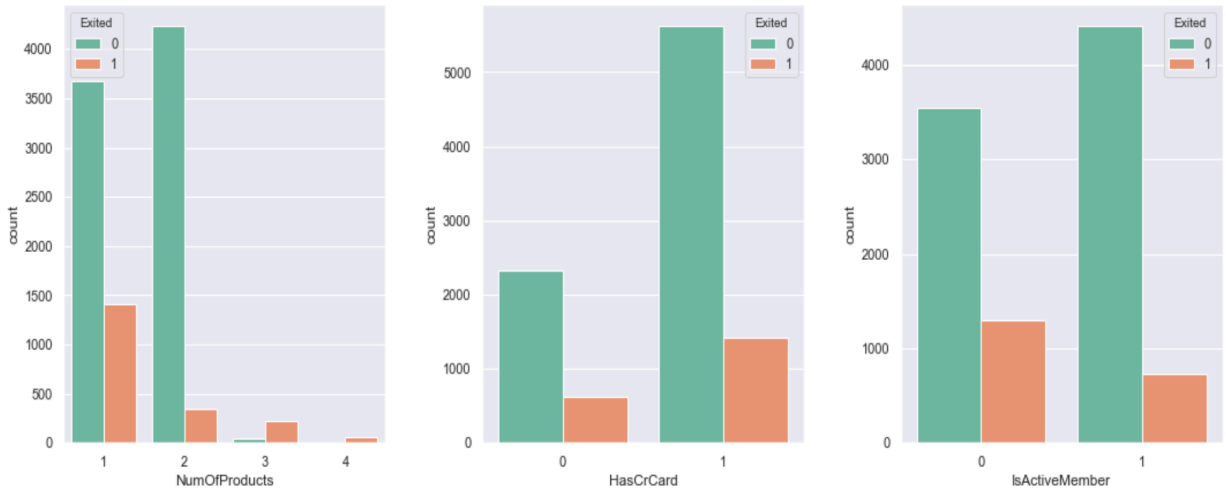
Data Visualization:

Before using different Classifiers we will first visualize the data and how the features in dataset directly affects the customer churn. Here I am using 'matplotlib' and 'seaborn' libraries for plotting and visualizing.

1. Number of Products customers own **vs** Total number of customers
2. Has Credit Card **vs** Total number of customers
3. Is an active member **vs** Total number of customers

```
In [9]: _, ax = plt.subplots(1, 3, figsize=(18, 6))
        plt.subplots_adjust(wspace=0.3)
        sns.countplot(x = "NumOfProducts", hue="Exited", data = dataset, ax= ax[0])
        sns.countplot(x = "HasCrCard", hue="Exited", data = dataset, ax = ax[1])
        sns.countplot(x = "IsActiveMember", hue="Exited", data = dataset, ax = ax[2])
```

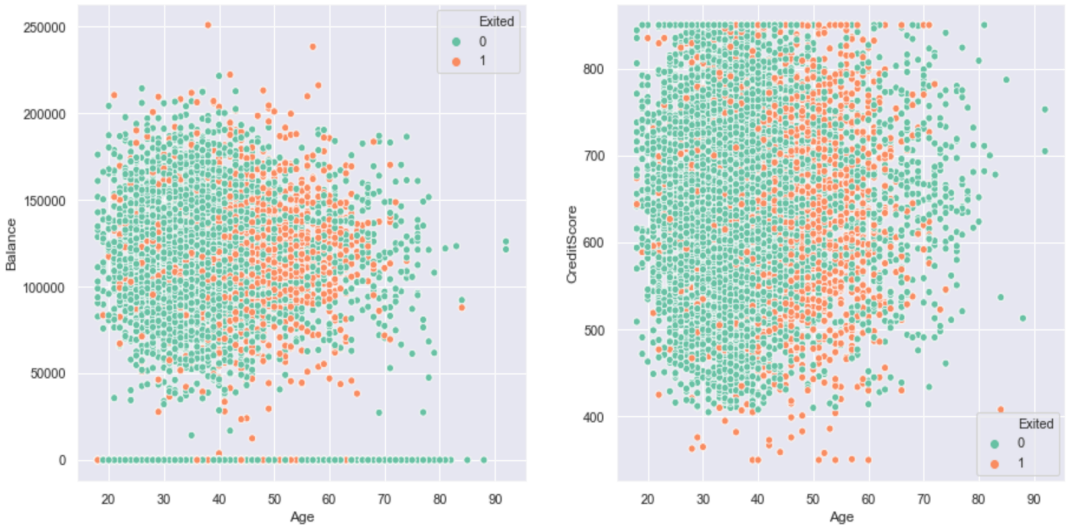
```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x137ccff28>
```



4. Age **vs** Balance
5. Age **vs** Credit Score

```
In [14]: _, ax =  plt.subplots(1, 2, figsize=(15, 7))
         cmap = sns.cubehelix_palette(light=1, as_cmap=True)
         sns.scatterplot(x = "Age", y = "Balance", hue = "Exited", cmap = cmap, sizes = (10, 200), data = dataset, ax=ax[0])
         sns.scatterplot(x = "Age", y = "CreditScore", hue = "Exited", cmap = cmap, sizes = (10, 200), data = dataset, ax=ax[1])
```

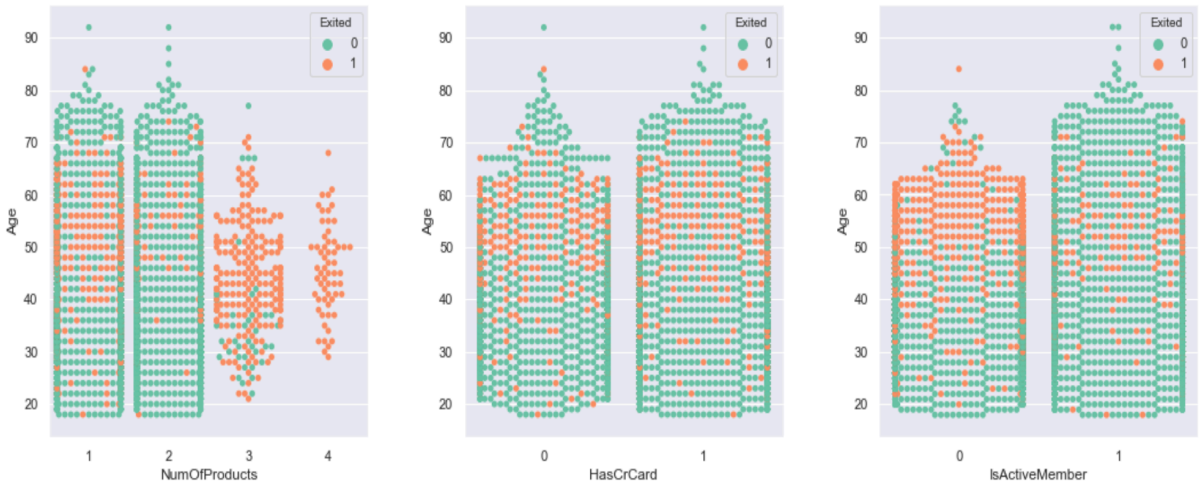
```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x139bbd4e0>
```



6. Number of Products customers own **vs** Age
7. Has Credit Card **vs** Age
8. Is an active member **vs** Age

```
In [12]: _, ax = plt.subplots(1, 3, figsize=(18, 6))
         plt.subplots_adjust(wspace=0.3)
         sns.swarmplot(x = "NumOfProducts", y = "Age", hue="Exited", data = dataset, ax= ax[0])
         sns.swarmplot(x = "HasCrCard", y = "Age", data = dataset, hue="Exited", ax = ax[1])
         sns.swarmplot(x = "IsActiveMember", y = "Age", hue="Exited", data = dataset, ax = ax[2])
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1394c7f60>
```



About Different Classifiers:

	Pros	Cons
Naïve Bayes	• Naïve Bayes follows conditional independence assumption which rarely holds true. • Most easy to implement and can scale with your dataset.	• Due to their sheer simplicity, NB models are often beaten by models which are properly trained and tuned.
Logistic Regression	• Outputs have a nice probabilistic interpretation • Algorithm can be regularized to avoid overfitting. • Logistic models can be updated easily with new data using stochastic gradient descent.	• Logistic regression tends to underperform when there are multiple or non-linear decision boundaries. • They are not flexible enough to naturally capture more complex relationships.
Decision Tree	• Optimum performance in practice. • They are robust to outliers, scalable, and able to naturally model non-linear decision boundaries • Follows a hierarchical structure. • Different ensemble methods of decision tree are: ▪ Random Forest ▪ Gradient Boosted Tree	• Unconstrained, individual trees are prone to overfitting, but this can be alleviated by ensemble methods.
SVM (Support Vector Machine)	• SVM's can model non-linear decision boundaries, and there are many kernels to choose from (Linear, Gaussian, Polynomial) • They are also fairly robust against overfitting, especially in high-dimensional space.	• SVM's are memory intensive • Trickier to tune due to the importance of picking the right kernel, and don't scale well to larger datasets. • Currently in the industry, random forests are usually preferred over SVM's.
Random Forest	• Random Forest is an ensemble of decision trees. • It solves both regression and classification problems with large data sets. • It also helps identify most significant variables from thousands of input variables. • Random Forest is highly scalable to any number of dimensions and has generally quite acceptable performances.	• Learning may be slow (depending on the parameterization) and it is not possible to iteratively improve the generated models.
XGB	• Extremely fast and parallel computation • Highly efficient • Versatile (Can be used for classification, regression or ranking).	• Only works with numeric features. • Leads to overfitting if hyperparameters are not tuned properly.

	• Can be used to extract variable importance. • Do not require feature engineering (missing values imputation, scaling and normalization)	

Accuracy Scores for various classifiers on the dataset:

On implementing various classifiers, the accuracy score for models are as follows:

Classifier	Accuracy Score
Naïve Bayes	0.784
Logistic Regression	0.787
Decision Tree	0.7915
Random Forest	0.864
Extreme Gradient Boost	0.857
SVM	0.7975

Model Visualization:

For better understanding, we will now visualize different models based on their prediction errors. We have already compared the Accuracy score in the above table. Now we will compare the precision score as well to help us determine the best model for the Wealth Management Dataset. I have used Yellowbrick visualizer to visualize different classification models. This visualizer uses 'sci-kit learn' and 'matplotlib' libraries for visualizing.

Steps for visualizing using Yellowbrick packages:

Step 1: The first step for this will be to install yellow brick in your system:

```
Vinishas-MacBook-Pro:~ vinisha$ pip3 install yellowbrick
```

Step 2: Now launch the jupyter notebook from your terminal

```
Vinishas-MacBook-Pro:~ vinisha$ jupyter notebook
```

Step 3: Now import the following files from yellowbrick package

```
In [26]: from yellowbrick.classifier import ClassPredictionError
```

```
In [ ]: from yellowbrick.classifier import PrecisionRecallCurve
```

Now we can use the code from BankChurn_ModelViz.py for creating training and testing datasets and perform visualization.

Note: 0 – Customers who did not Exit

1 – Customers Exited

1. **Logistic Regression:**

The figure below illustrates the actual and the error in predicting number of customers exited

```
In [21]: clf = LogisticRegression()
         visualizer = ClassPredictionError(clf)
         visualizer.fit(X_train, y_train)
         visualizer.score(X_test,y_test)
         visualizer.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

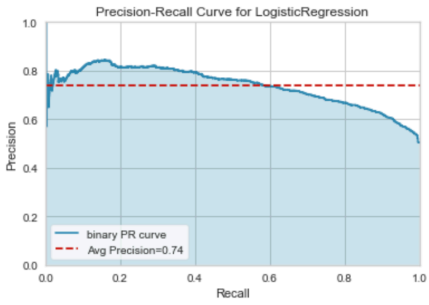

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1415300f0>
```

The figure below illustrates the Precision Recall Curve

Avg Precision Score: **0.74**

```
In [22]: clf = LogisticRegression()
         visualizer = PrecisionRecallCurve(clf)
         visualizer.fit(X_train, y_train)
         visualizer.score(X_test,y_test)
         visualizer.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

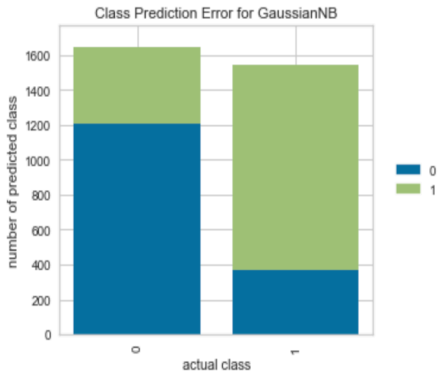



```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x140f2b748>
```

2. Gaussian Naïve Bayes

The figure below illustrates the actual and the error in predicting number of customers exited

In [17]:
```
clf = GaussianNB()
visualizer = ClassPredictionError(clf)
visualizer.fit(X_train, y_train)
visualizer.score(X_test,y_test)
visualizer.show()
```

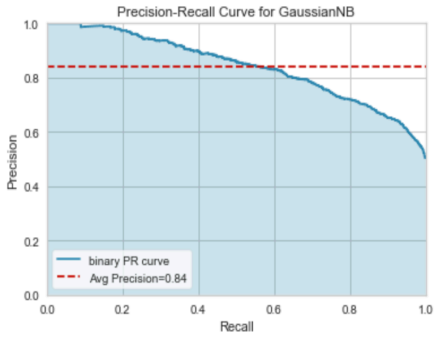


Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x14120a198>

The figure below illustrates the Precision Recall Curve

Avg Precision Score: **0.84**

In [20]:
```
from yellowbrick.classifier import PrecisionRecallCurve
visualizer = PrecisionRecallCurve(GaussianNB())
visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

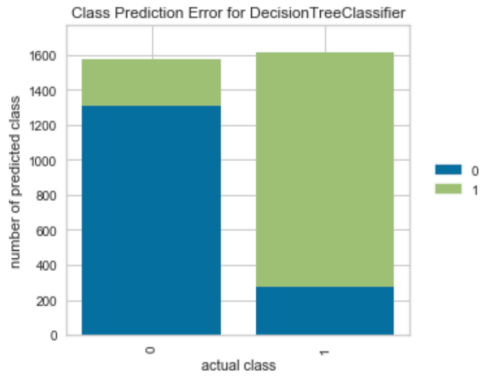


Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1414ad828>

3. **Decision Tree**

The figure below illustrates the actual and the error in predicting number of customers exited

```
In [10]: clf = tree.DecisionTreeClassifier()
         visualizer = ClassPredictionError(clf)
         visualizer.fit(X_train, y_train)
         visualizer.score(X_test,y_test)
         visualizer.show()
```

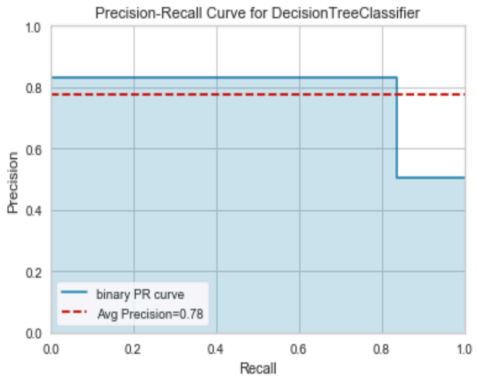



```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x140b04eb8>
```

The figure below illustrates the Precision Recall Curve

Avg Precision Score: **0.78**

```
In [43]: clf = tree.DecisionTreeClassifier()
         visualizer = PrecisionRecallCurve(clf)
         visualizer.fit(X_train, y_train)
         visualizer.score(X_test,y_test)
         visualizer.show()
```




```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x110a554e0>
```

4. SVM

The figure below illustrates the actual and the error in predicting number of customers exited

```
In [7]: svclassifier = SVC(kernel='rbf')
        visualizer = ClassPredictionError(svclassifier)
        visualizer.fit(X_train, y_train)
        visualizer.score(X_test,y_test)
        visualizer.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/svm/base.py:193: FutureWarning:
The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled feature
s. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

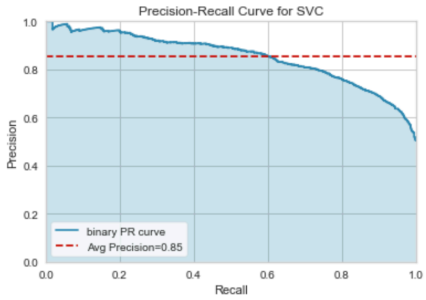

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1401ada90>
```

The figure below illustrates the Precision Recall Curve

Avg Precision Score: **0.85**

```
In [25]: svclassifier = SVC(kernel='rbf')
         visualizer = PrecisionRecallCurve(svclassifier)
         visualizer.fit(X_train, y_train)
         visualizer.score(X_test,y_test)
         visualizer.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/svm/base.py:193: FutureWarning:
The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled feature
s. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

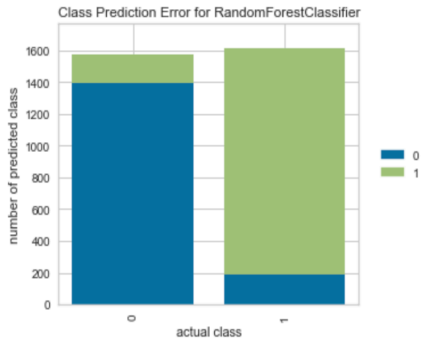



```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1419d46a0>
```

5. Random Forest

The figure below illustrates the actual and the error in predicting number of customers exited

In [6]:
```
classes = ['Exited', 'Not Exited']
clf = RandomForestClassifier(n_estimators = 200, random_state=200)
visualizer = ClassPredictionError(clf)
visualizer.fit(X_train, y_train)
visualizer.score(X_test,y_test)
visualizer.show()
```



Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1403e1a58>

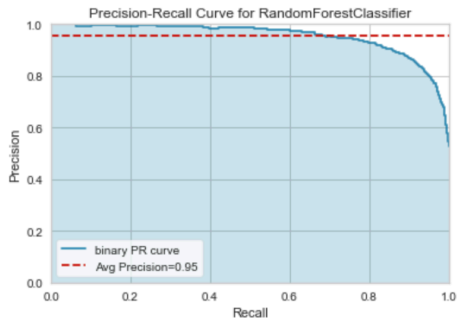
In [7]:
```
svclassifier = SVC(kernel='rbf')
visualizer = ClassPredictionError(svclassifier)
visualizer.fit(X_train, y_train)
visualizer.score(X_test,y_test)
visualizer.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/svm/base.py:193: FutureWarning:
The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled feature
s. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
```

The figure below illustrates the Precision Recall Curve

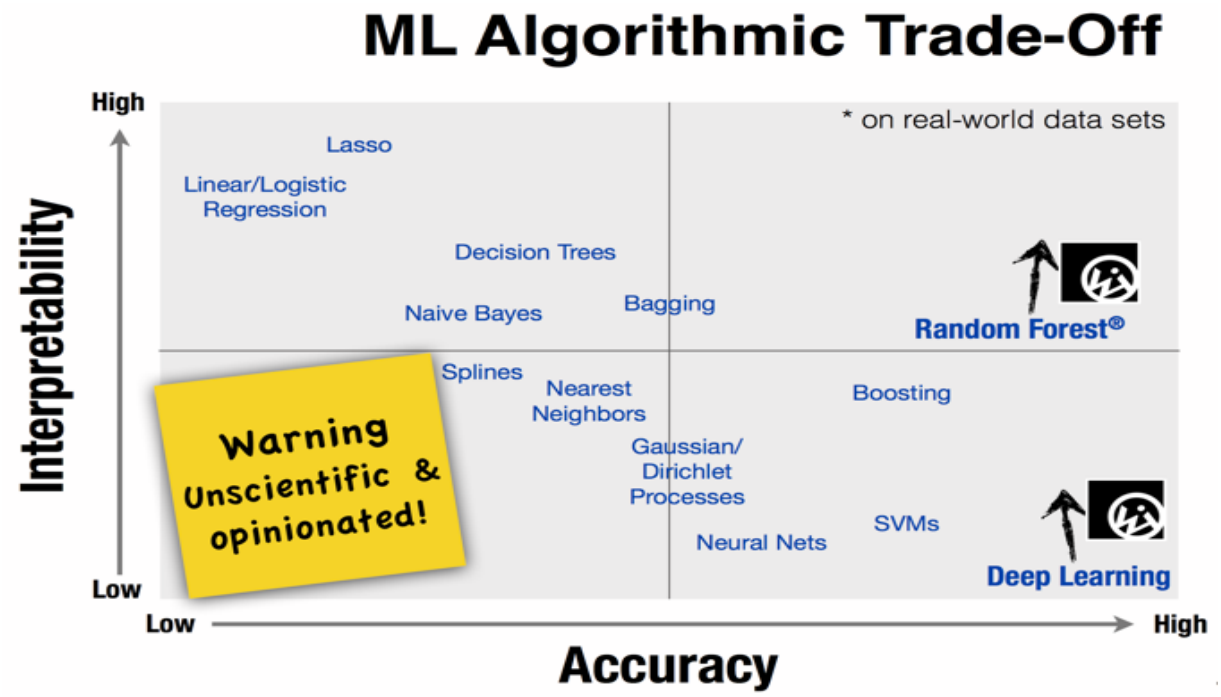
Avg Precision Score: **0.95**

In [24]:
```
clf = RandomForestClassifier(n_estimators = 200, random_state=200)
visualizer = PrecisionRecallCurve(clf)
visualizer.fit(X_train, y_train)
visualizer.score(X_test,y_test)
visualizer.show()
```



Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x14165ac50>

Factors to be considered before choosing any Machine learning algorithm:



Conclusion

We choose Random Forest Classifier for the Wealth Management Dataset because of the following reasons:

- It yields the Highest Accuracy Sccore – **0.864**
- It yields the Highest Precision Value – **0.95**
- The error prediction is relatively low as compared with other classifying models.

Additional interesting facts about random forest why and how it fits best for our dataset:

- Random forest works great in financial sector to predict loyal customers and fraud customers
- Helps in predicting the disease by analyzing patients medical records.
- Random forest is also used in e-commerce websites to identify the likelihood of a customer and predicting items based on the likelihood.
- Considered as a very strong algorithm for accurate prediction based on the inputs given.

References

https://www.dataquest.io/blog/sci-kit-learn-tutorial/
https://www.youtube.com/watch?v=J4Wdy0Wc_xQ
https://www.youtube.com/watch?v=D_2LkhMJcfY
https://www.kaggle.com/nasirislamsujan/bank-customer-churn-prediction
https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/
https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
https://www.scikit-yb.org/en/latest/api/classifier/class_prediction_error.html