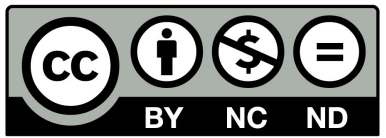
Low Code Visualization and ML

An initiative by Kimaru Thagana, AICE first cohort

An AI Commons initiative



The Low Code paradigm

In software development, low code involves tools that require minimal to no effort to produce results such as software solutions. In this lecture, we will discuss low code tools in the Exploratory Data Analysis (EDA) and ML spaces.

Low code analysis provides easy access to information and data visualization.

Exploratory Data Analysis (EDA)

- EDA is the preliminary process by which data scientists or analysts get to explore the datasets they are to work with.
- The main objective is to get an idea of what the data landscape is, what columns are there, what data types are present, the distribution of the data, presence of missing values,...
- Getting acquainted with the data, paired with domain knowledge, gives a powerful foundation for the next task which is feature engineering.

EDA tools: visualization & summary

The main EDA tools are visualization and summary: they give a bird's eye view of the data, and allow you to quickly learn about the characteristics of the data.

The EDA process is quite standard, and predictable. Tasks are:

- Visualizing distribution
- Calculating percentage of missing values
- Calculating mean, max, min and standard deviation
- Identifying categorical columns
- Identifying distribution of classes among columns

EDA tools vs. standard libraries

EDA streamlines the standard steps in data analysis, resulting in an efficient tool that can examine datasets for quick insights.

Functions exist, such as Pandas describe() function for datasets that summarizes data but does not offer visualizations. Seaborn pairplot() function offers visualization.

The value of EDA tools such as SweetViz is to provide a low code, easy to use, library to generates EDA with one line of code.

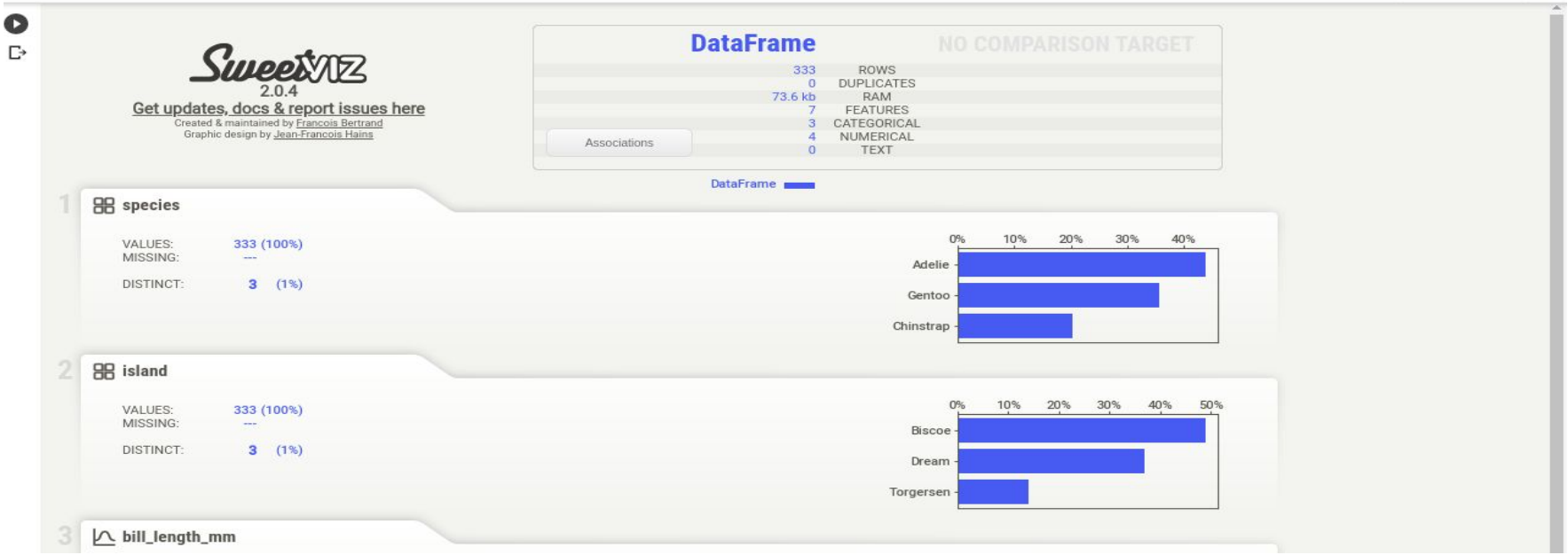
SweetViz

SweetViz applies its functions on a pandas dataframe and then displays the result in a notebook output or specified HTML file. The single line of code required to generate the visualization is demonstrated below:

```
import sweetviz as sv
import IPython
analysis_report = sv.analyze(df)# perform EDA analysis
analysis_report.show_html(EDA_results.html',open_browser=True)
IPython.display.HTML(EDA_results.html')
```

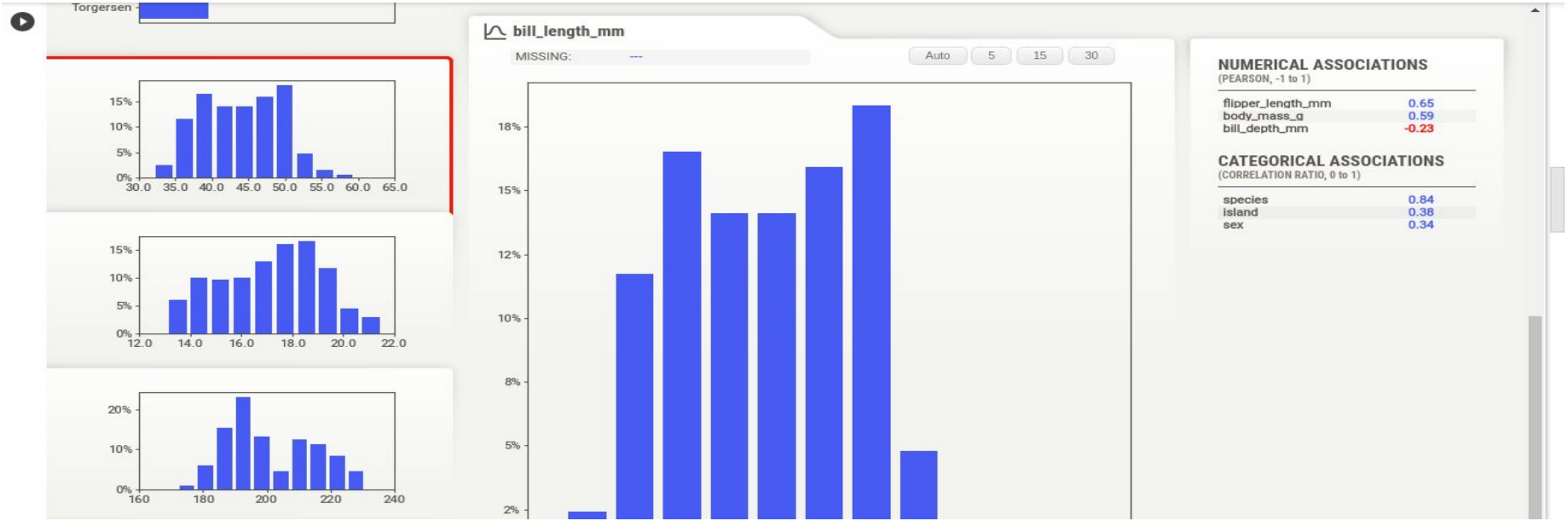
That’s all. Your visualizations are ready for viewing.

SweetViz - general dataframe summary

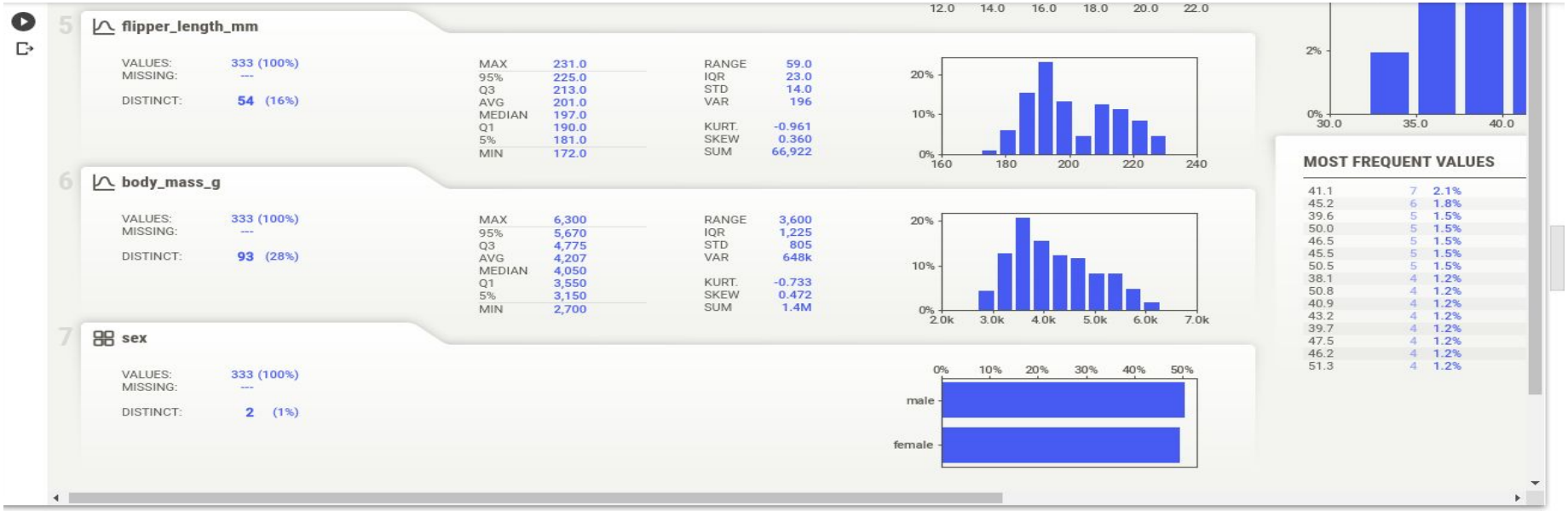


SweetViz - summary on categorical variable

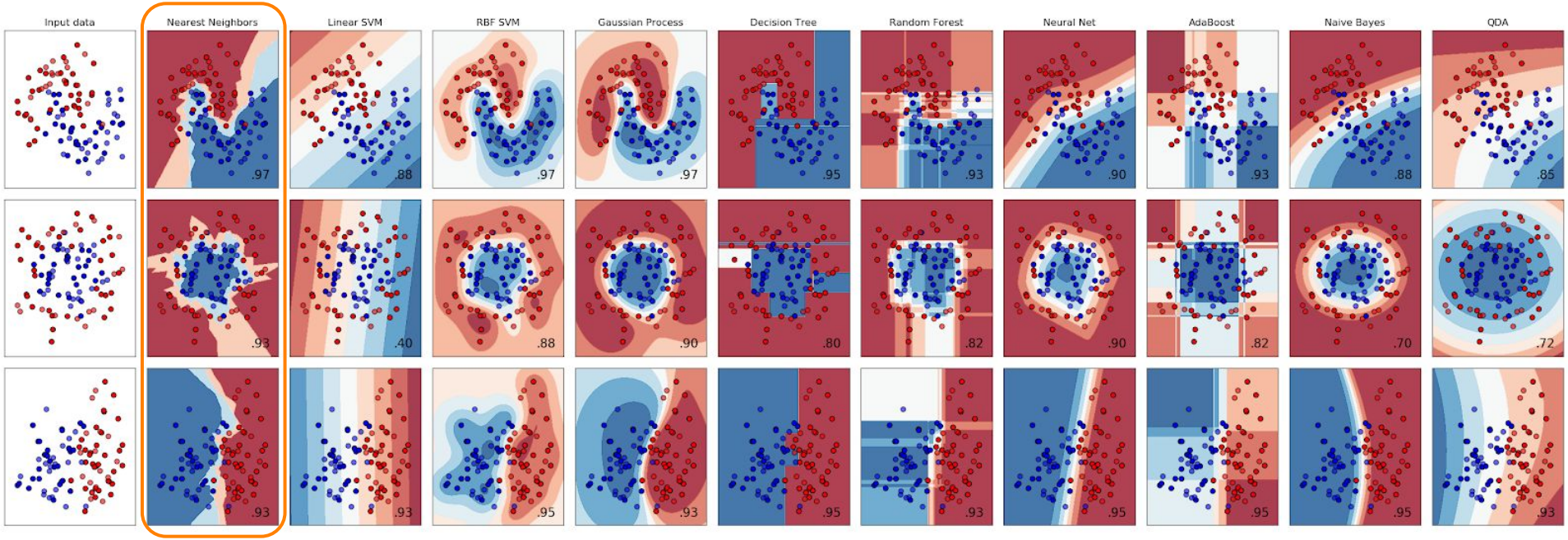
SweetViz - summary of numerical variable



SweetViz - summary of numerical variable - (cont'd)



How to pick the best algorithm?



Source: scikitLearn

It is difficult to know in advance what algorithm would work best.
This is where Low code/AutoML-type tools come handy.

How to evaluate the best classification models?

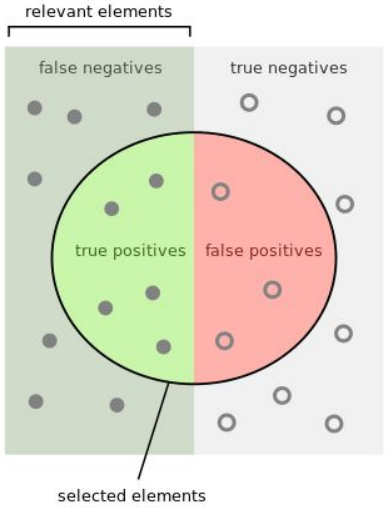
- **Precision** (positive predictive value)
  - Identifies the ratio of items correctly identified as positive to that of all items identified as positive
  - Used when the accuracy of the prediction is important.

  Precision = true positive / (true positive + false positive)
- **Recall** (sensitivity)
  - Identifies the ratio of items correctly identified as positive to that of all truly positive items
  - Used when it is important not to miss positive items

  Recall = true positive (true positive + false negative)
  - Is not a performance criteria for the algorithm
- **Accuracy**
  - Identifies the ratio of items correctly identified to that of all items identified

  Accuracy = (true positive+true negative)/(positives + negatives)
  - Can be misleading when you have unbalanced datasets
- **F1 Score**
  - The F1 score is a better measure of a test's accuracy with unbalanced datasets

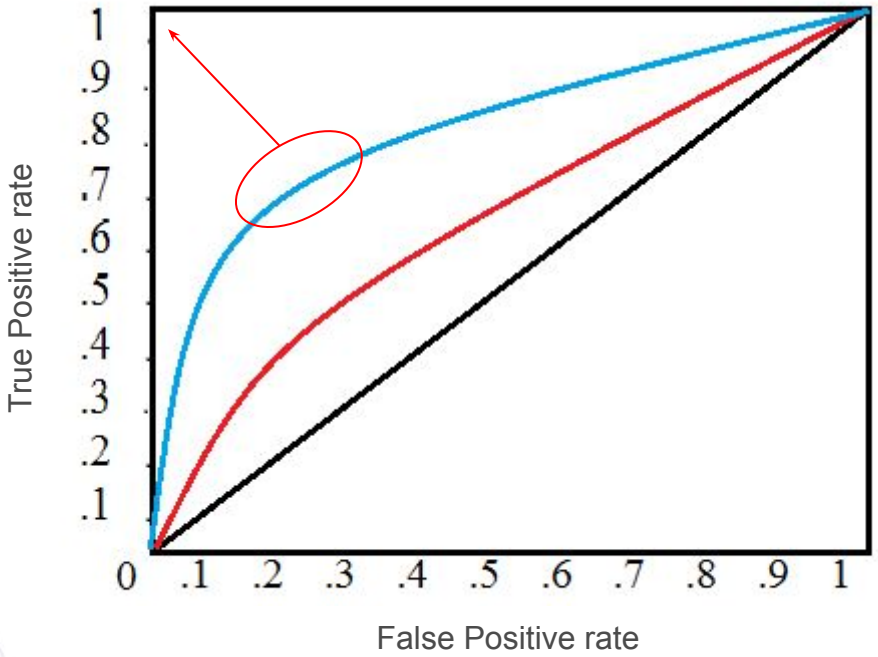
  F1 = 2 * (Precision*Recall) / (Precision + Recall)
  - Useful when precision and recall have similar importance. Otherwise use F_β, where β is chosen such that recall is considered β times as important as precision.



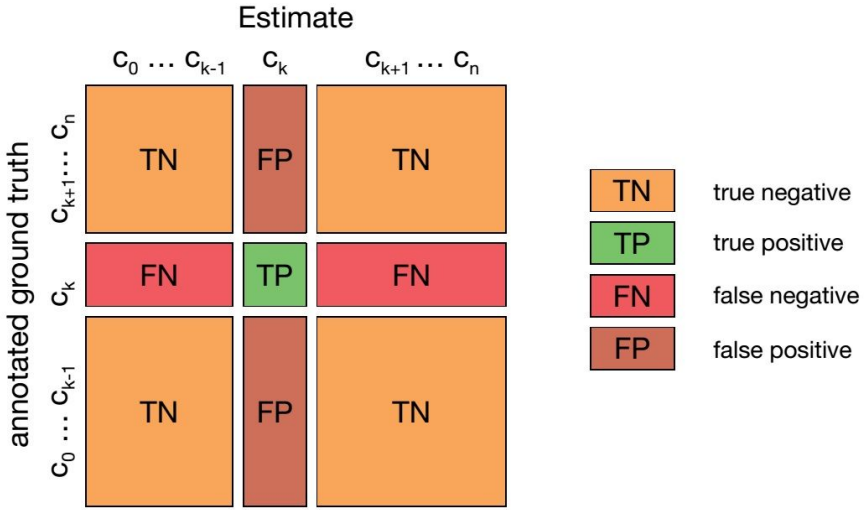
Source: wikipedia

Other evaluation considerations

- **AUC** (**A**rea **U**nder the **C**urve)
  - $AUC_{min} = 0.5$, $AUC_{max} = 1$



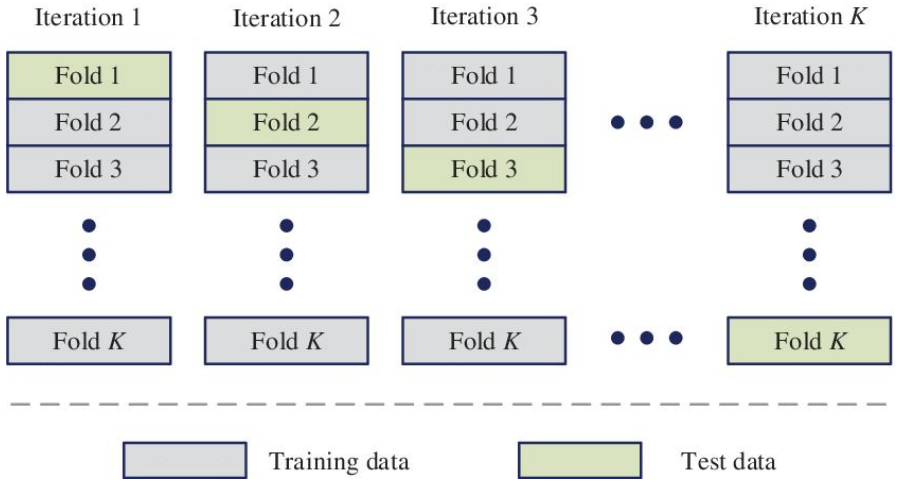
- **Confusion Matrix**



Source: Devopedia

K-Fold cross validation

- Data samples should be split:
  - Training (learning)
  - Validation (tuning)
  - Test (evaluation)
- Cross validation:
  - sampling validation sets iteratively from the training set
- K-fold:
  - Split training set into k segments



Source: Researchgate

Low Code machine learning - Pycaret example

Low code ML is a paradigm where ML is done with as little code as possible. It builds on the fact that some initial processes in ML tasks are fairly standard so tools and libraries can be streamlined to reduce the development time of a model.

Pycaret is a popular tool for that. Its advantage is in its ability to evaluate the dataset against various algorithms using a defined criteria to give the best algorithm. This saves a lot of time that would have been used coding and evaluating algorithms to find the best ones.

Classification example: setup & compare

For this example, we will use a Python notebook to demonstrate pyCaret on a classification task.

Load pycaret, sample data and setup the task:

```
from pycaret.datasets import get_data

dataset = get_data('juice')

from pycaret.classification import *

setup_1 = setup(dataset, target='Purchase',ignore_features=['Id'])
```

Then perform model comparison (compare multiple models in one operation):

```
#perform model comparison

compare_models()
```

Datasets are split by default 70/30 (training/test)

Model comparison

Default ranking for classification is by accuracy, and by R^2 for regression. One can choose other parameters such as precision and recall depending on the use case.

```
In [4]: #perform model comparison
        compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.8397	0.0000	0.7683	0.8048	0.7810	0.6552	0.6607	0.021
lda	Linear Discriminant Analysis	0.8383	0.9021	0.7648	0.8054	0.7791	0.6523	0.6582	0.021
lr	Logistic Regression	0.8290	0.9022	0.7329	0.8047	0.7618	0.6294	0.6358	0.454
gbc	Gradient Boosting Classifier	0.8263	0.8995	0.7720	0.7795	0.7699	0.6312	0.6371	0.140
ada	Ada Boost Classifier	0.8196	0.8899	0.7686	0.7605	0.7608	0.6165	0.6203	0.137
catboost	CatBoost Classifier	0.8183	0.8983	0.7507	0.7781	0.7564	0.6125	0.6201	1.218
xgboost	Extreme Gradient Boosting	0.8090	0.8815	0.7612	0.7505	0.7506	0.5967	0.6018	1.254
lightgbm	Light Gradient Boosting Machine	0.8023	0.8857	0.7506	0.7440	0.7415	0.5824	0.5881	0.071
rf	Random Forest Classifier	0.7876	0.8713	0.7010	0.7393	0.7137	0.5458	0.5517	0.517
et	Extra Trees Classifier	0.7755	0.8306	0.6797	0.7227	0.6959	0.5187	0.5234	0.518
dt	Decision Tree Classifier	0.7728	0.7609	0.7015	0.7016	0.6946	0.5151	0.5211	0.021
nb	Naive Bayes	0.7702	0.8364	0.7762	0.6678	0.7166	0.5254	0.5313	0.022
knn	K Neighbors Classifier	0.7299	0.7896	0.5695	0.6626	0.6100	0.4062	0.4103	0.122
svm	SVM - Linear Kernel	0.5510	0.0000	0.3000	0.1133	0.1645	0.0000	0.0000	0.020
qda	Quadratic Discriminant Analysis	0.4316	0.4730	0.6204	0.3545	0.4403	-0.0513	-0.0717	0.025

```
Out[4]: RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                        max_iter=None, normalize=False, random_state=4087,
                        solver='auto', tol=0.001)
```

Tune Models

```
#creating your model
model = create_model('gbc',fold=10)

#tuning your model
tuned_model = tune_model('gbc',fold=10)
```

create_model()

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8095	0.8392	0.6667	0.8000	0.7273	0.5828
1	0.7302	0.8563	0.7500	0.6207	0.6792	0.4499
2	0.8254	0.8771	0.6667	0.8421	0.7442	0.6144
3	0.8387	0.8438	0.7500	0.8182	0.7826	0.6548
4	0.8871	0.9320	0.8333	0.8696	0.8511	0.7602
5	0.8226	0.8596	0.7083	0.8095	0.7556	0.6173
6	0.8548	0.9452	0.7500	0.8571	0.8000	0.6869
7	0.8387	0.9298	0.7083	0.8500	0.7727	0.6493
8	0.8548	0.8964	0.7500	0.8571	0.8000	0.6869
9	0.8387	0.9192	0.6522	0.8824	0.7500	0.6349
Mean	0.8301	0.8899	0.7236	0.8207	0.7663	0.6337
SD	0.0390	0.0377	0.0518	0.0712	0.0444	0.0770

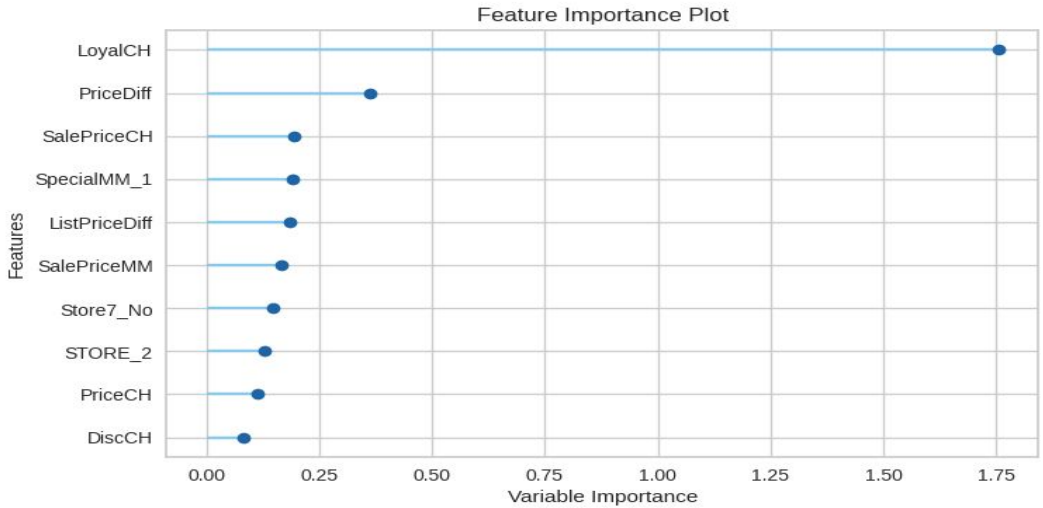
tune_model()

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.8413	0.8339	0.7083	0.8500	0.7727	0.6523
1	0.6984	0.8547	0.7500	0.5806	0.6545	0.3945
2	0.8254	0.8611	0.6667	0.8421	0.7442	0.6144
3	0.7258	0.7796	0.5833	0.6667	0.6222	0.4085
4	0.8548	0.9364	0.7917	0.8261	0.8085	0.6917
5	0.8065	0.8914	0.7083	0.7727	0.7391	0.5857
6	0.7903	0.9101	0.7083	0.7391	0.7234	0.5547
7	0.8387	0.9178	0.7500	0.8182	0.7826	0.6548
8	0.7903	0.8580	0.7083	0.7391	0.7234	0.5547
9	0.8387	0.9186	0.7391	0.8095	0.7727	0.6481
Mean	0.8010	0.8762	0.7114	0.7644	0.7343	0.5760
SD	0.0494	0.0455	0.0535	0.0817	0.0548	0.0970

Source: Pycaret.org

Model visualization - feature importance plot

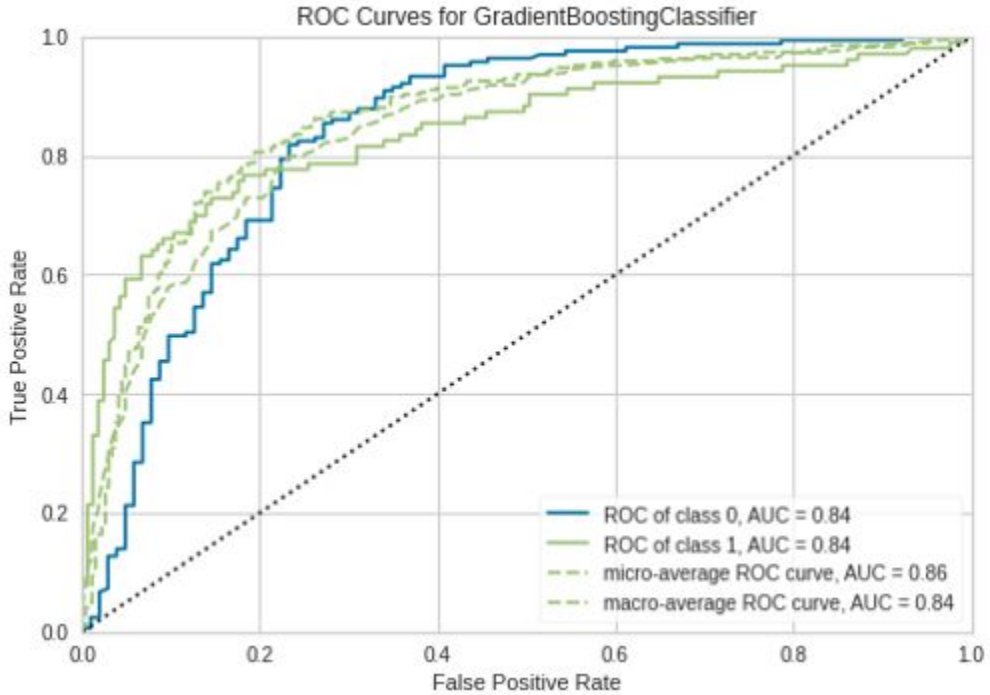
```
In [27]: plot_model(tuned_ridge_clf,plot='feature') #pr, boundary feature
```



Try out ensembles to improve model performance

Model visualization - ROC curve

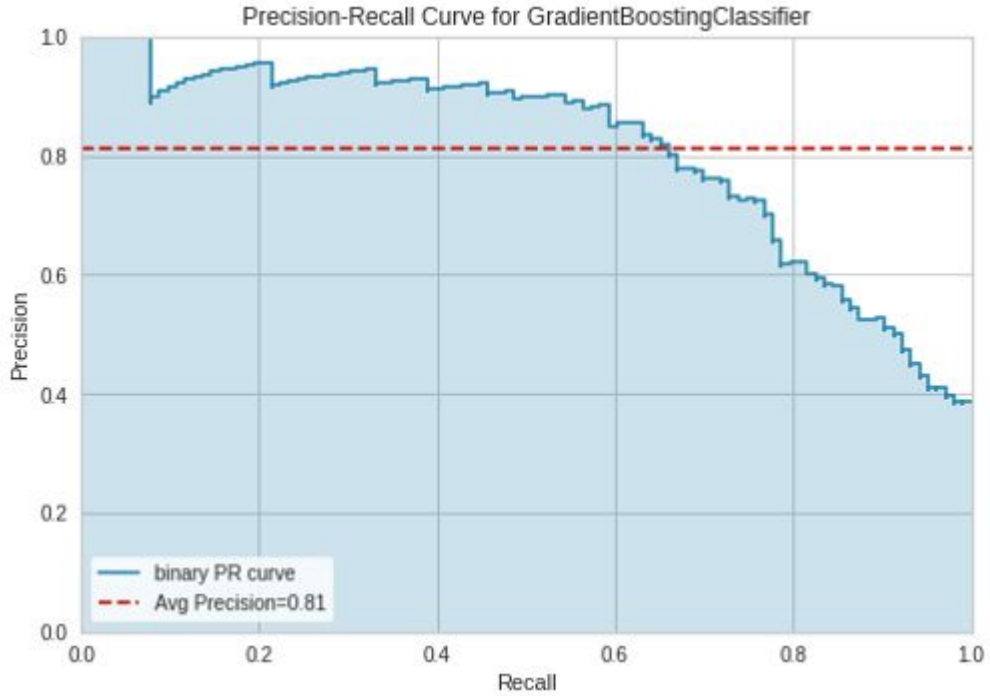
```
1  # plotting a AUC using the tuned model
2  plot_model(tuned_model,plot = 'auc')
```



Source: Pycaret.org

Model visualization - Precision recall plot

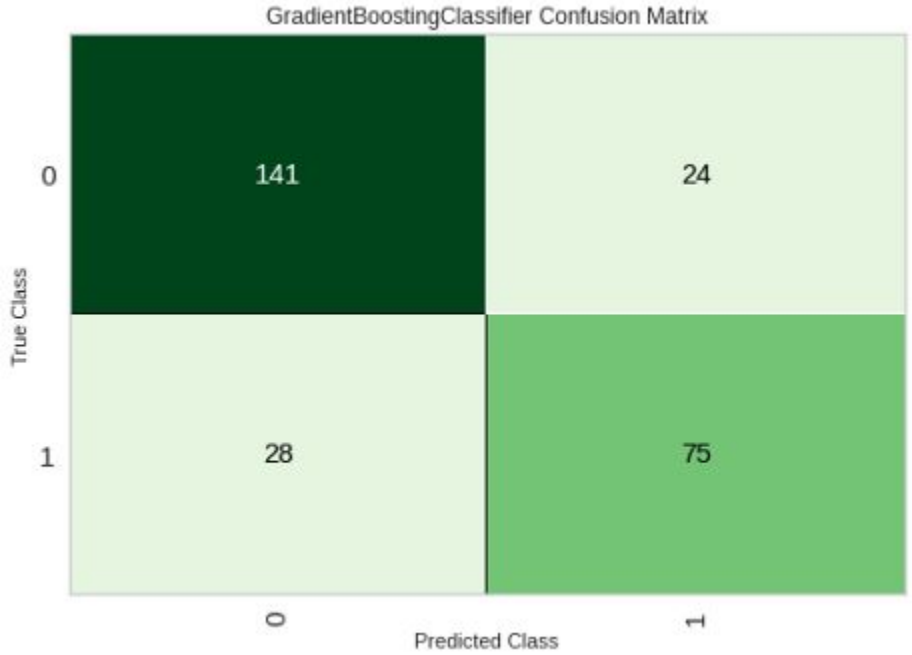
```
7  # plotting precision-recall
8  plot_model(tuned_model,plot = 'pr')
```



Source: Pycaret.org

Model visualization - Confusion matrix

```
10  # plotting confusion matrix
11  plot_model(tuned_model,plot = 'confusion_matrix')
```



Source: Pycaret.org

Model Plot - R^2 regression

regression in pycaret.ipynb

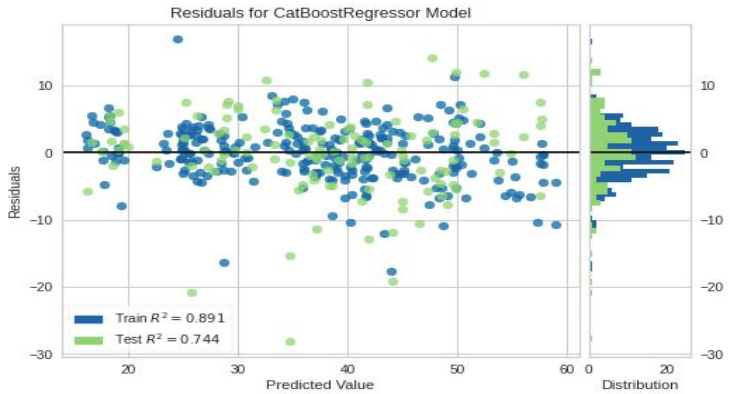
File Edit View Insert Runtime Tools Help

Share

+ Code + Text Copy to Drive

Connect Editing

```
plot_model(tuned_model)
```




```
[ ]  evaluate_model(tuned_model)
```

Model interpretation using SHAP

Model interpretation visualization using SHAP

regression in pycaret.ipynb

File Edit View Insert Runtime Tools Help

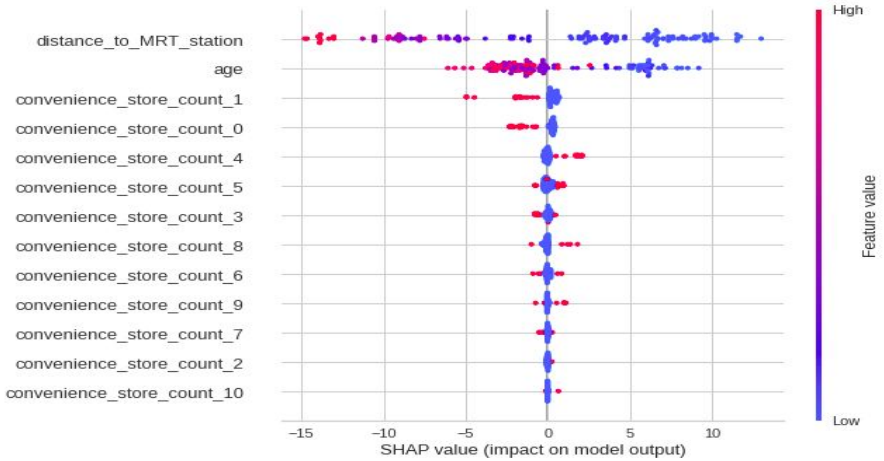
Share

+ Code + Text Copy to Drive

Connect Editing

Model interpretation using SHAP

```
interpret_model(tuned_model)
```



Further Reading on Low Code in ML

We have explored the SweetViz and Pycaret libraries but there are more.

To broaden your horizon and knowledge on the subject, read upon Py-AutoML

An open source python based project that employs the low code paradigm to reduce the time taken from hypothesis to insight in machine learning.

References

More on Pycaret and SweetViz

https://github.com/KimaruThagna/data-science-in-pycaret

Classification in Pycaret

https://colab.research.google.com/drive/1meiGGYG0OniUwNmZBSJqnK8UJbMHbnm8?usp=sharing

Regression using Pycaret

https://colab.research.google.com/drive/1zliNvt37LkZ7T8n8uqIdfHk0MaxOjqvP?usp=sharing

Pycaret reference webpage

https://pycaret.org/

Pycaret tutorial

https://www.pycaret.org/tutorials/html/CLF101.html

Contributors

Kimaru Thagana

UNIVERSIDAD DE CHILE

Jose Ignacio Diaz

AI Commons

Gilles Fayad