Contents

Utilizing InterpretML for Model Explainability

Below is an excerpt that I had sent earlier to summarize information on InterpretML. Later in this document I would try to elaborate each section with screenshots to make it easier to understand

"Here are few findings around the same :-

1. **InterpretML** is a simpler library as compared to **Alibi** and easy to understand
2. **InterpretML** is managed by **Microsoft Research team** and has active collaboration with the **developer community of SHAP and LIME**
3. **InterpretML** comes inbuilt with **Azure SDK and** also has a **text version for NLP.**
4. Here are few additional details around InterpretML :-
   - **A.** **InterpretML** supports : **Glassbox Models** and **Blackbox Explainers**
   - **B.** **Glassbox models** - models that are meant to be interpretable.
     - Thus there are 4 models which are created inhouse as part of IntepretML : Explainable Boosting Machines, Decision Tree, Decision Rule List, Linear/Logistic Regression
     - **EBM : explainable as Linear models, but accurate as RF or any ensemble models**
     - Basic idea behind building these models is to make general additive models more explainable even with Boosting (explainable boosting)

- These also provide both Local as well as Global explainability.
- This doesn't work for a multi class classification problem

C. **Blackbox explanations** : These are explanations that can be given for any models coming from any other library such as sklearn
   a. They consider **only input and output values** and assume that based on these values they need to **identify the explainability.**
   b. In this process, changes are made to inputs and passed through model to analyse the change in model output and thus provide **explainability.**
   c. **Local explainers** in blackbox explanations include: **SHAP, LIME**
   d. **Global explainers** in Blackbox explanations include : **Partial Dependency Plot** and **Morris Sensitivity.**
   e. This can work very well on deep neural nets or on complex ml pipelines

I have used the loan dataset shared earlier by Ankita and implemented both Glassbox model and blackbox explainers. "

Step 1: InterpretML : Glassbox models and explainers

A : Installing InterpretML

```
[ ]: !pip install interpret
```

Dataset being used :- https://www.kaggle.com/arashnic/banking-loan-prediction

- **Although this was a multi class classification dataset, but InterpretML currently only supports binary classification.**
- **Hence the encoding for target variable will be made to consider only 2 classes**

Step 2: Initial Pre-processing of the data using pandas

1. Understanding the data using descriptive statistics
2. Looking at the null values present in the data
3. Treating the null values based on frequency presence in the data
4. Label encoding the target variable based on Step 1 mentioned.

Step 3: Using InterpretML's functions to obtain descriptive stats (visually)

- Code:

```
from interpret import show
from interpret.data import ClassHistogram

hist = ClassHistogram().explain_data(X_train, y_train, name = 'Train Data')
show(hist)
```

- **Visualization:** by selecting features from drop down:
  We can select different features from the dropdown generated and look at the distribution of target binary classes across these continuous as well as categorical features

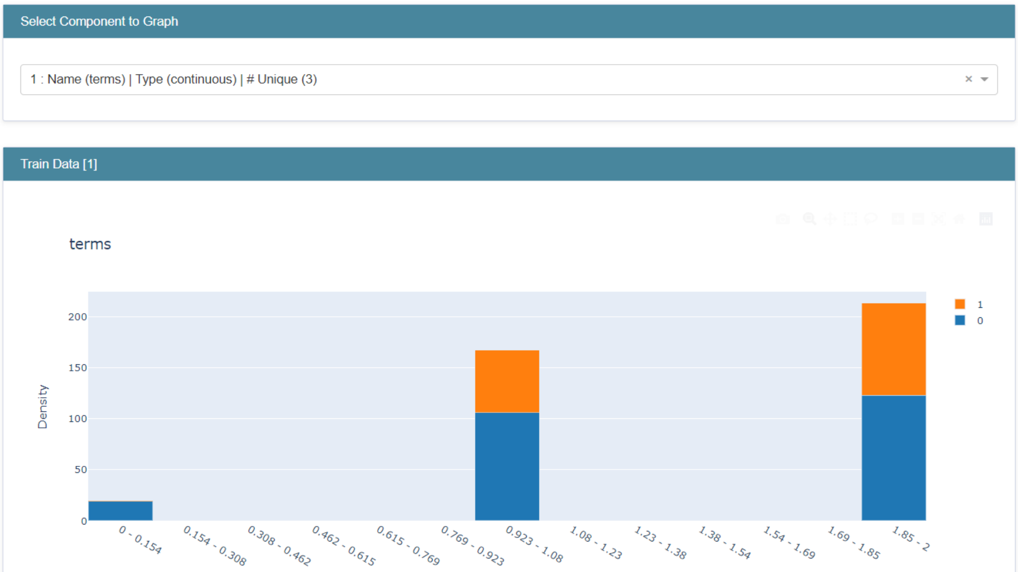


Figure 1 : Distribution of Term (continuous column vs target variable)

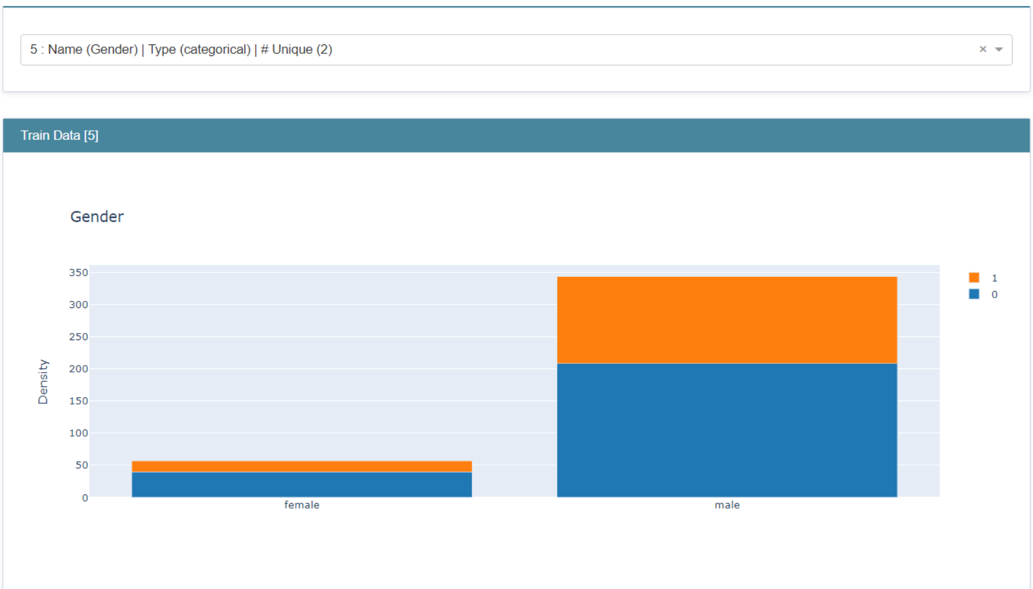


Figure 2 Distribution of Gender (categorical column vs target variable)

Step 4: Using Glassbox model from InterpretML

Glassbox models

a. Models that are meant to be interpretable and are internally present within interpretML.
b. There are 4 models which are created inhouse as part of IntepretML : Explainable Boosting Machines, Decision Tree, Decision Rule List, Linear/Logistic Regression

c. Basic idea behind building these models is to make general additive models more explainable even with Boosting (explainable boosting)
d. These also provide both Local as well as Global explainability.
e. This doesn't work for a multi class classification problem

```
[30]: from interpret.glassbox import ExplainableBoostingClassifier, LogisticRegression, ClassificationTree, DecisionListClassifier

[31]: ebm_bin = ExplainableBoostingClassifier(random_state=seed, n_jobs=-1)
      ebm_bin.fit(X_train, y_train)   #Works on dataframes and numpy arrays

[31]: ExplainableBoostingClassifier(feature_names=['Principal', 'terms',
                                                 'past_due_days', 'age',
                                                 'education', 'Gender',
                                                 'past_due_days x age',
                                                 'terms x past_due_days',
                                                 'past_due_days x Gender',
                                                 'Principal x past_due_days',
                                                 'past_due_days x education',
                                                 'terms x age', 'age x education',
                                                 'terms x education',
                                                 'terms x Gender',
                                                 'Principal x terms'],
                              feature_types=['continuous', 'continuous',
                                             'continuous', 'continuous',
                                             'continuous', 'categorical',
                                             'interaction', 'interaction',
                                             'interaction', 'interaction',
                                             'interaction', 'interaction',
                                             'interaction', 'interaction',
                                             'interaction', 'interaction'],
                              n_jobs=-1)
```

Figure 3 : ML model fitting from InterpretML library

Step 5: Using global explainers from Glassbox models

- **An overall summary plot provides details on how each feature impacts overall model outcome**
- Individual plots can help us understand how model prediction varies for different range of values of each individual feature
- Higher you are on y axis in this individual plot, the higher chances that you will either have a collection or collection_paidoff
- Looking at each segment in an individual feature helps in saving: - **Sampling Bias , Overfitting

```
[32]: ebm_bin_global = ebm_bin.explain_global(name='EBM')
show(ebm_bin_global)
```

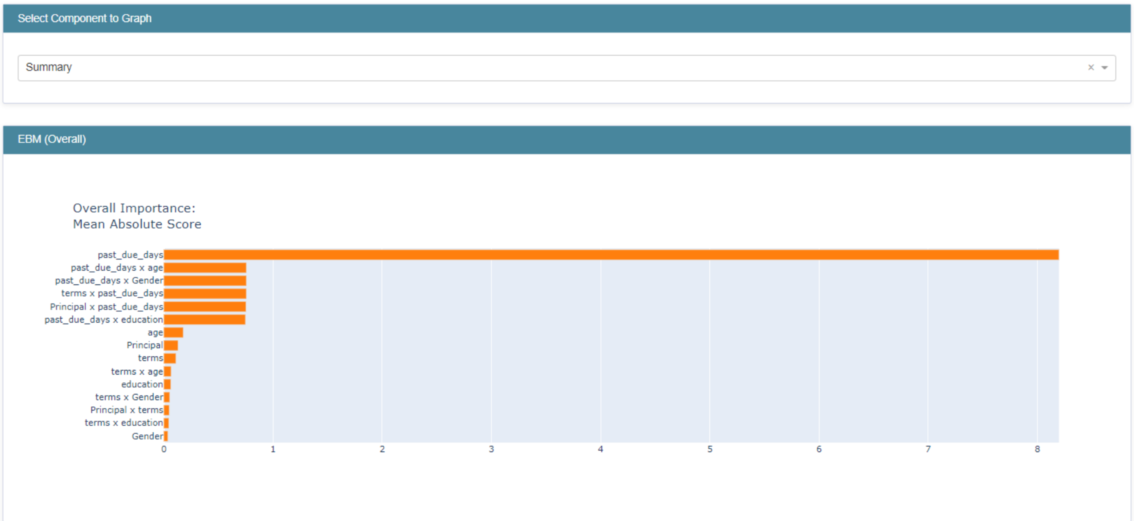


Figure 4 - Global explainers from InterpretML : Provides feature importance at an overall model level.

Step 6: Using local explainers from Glass box models

- Local explainers help in looking at each row item and understand feature contribution towards prediction.
- Select each row from the drop down to look at feature contribution

6 : Local Explanations: How an individual prediction was made

```
[33]: ebm_bin_local = ebm_bin.explain_local(X_test[:5], y_test[:5], name='EBM')
show(ebm_bin_local)
```

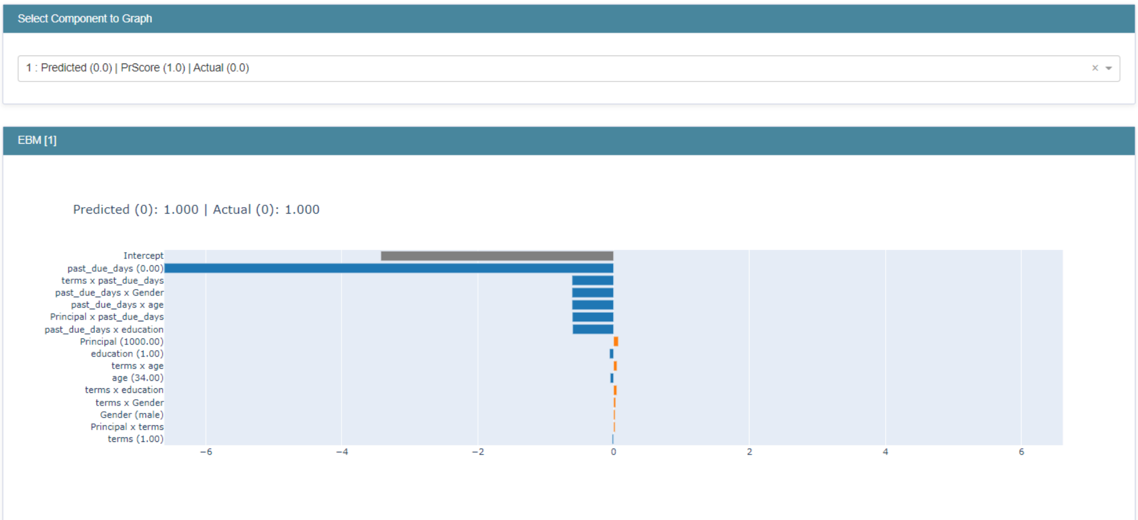


Figure 5 - InterpretML local explanation

Step 7: Glassbox Model: Performance metrics

7 : Evaluate EBM performance

7.1 : Looking at ROC curve for EBM

```
[34]: from interpret.perf import ROC

ebm_perf = ROC(ebm_bin.predict_proba).explain_perf(X_test, y_test, name='EBM')
show(ebm_perf)
```

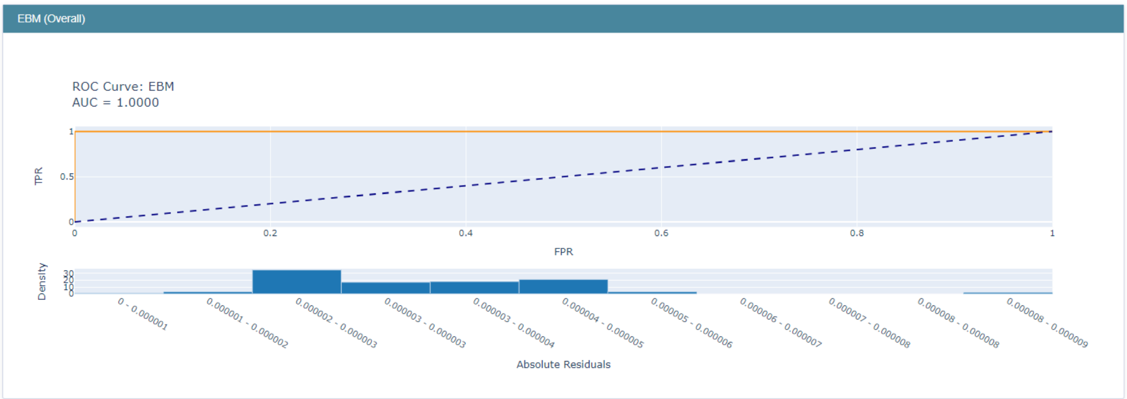


Figure 6 :- ROC curve from the interpret Explainable Boosting Machine

Step 8: Glassbox Model: Training other models

```
[38]: from interpret.glassbox import LogisticRegression, ClassificationTree

lr = LogisticRegression(random_state=seed,penalty='l1', solver='liblinear') #tunability of glassbox models
lr.fit(X_train_enc, y_train)
```

- Comparing the performance metrics for each model

7.3 : Compare performance using Dashboard

```
40]: lr_perf = ROC(lr.predict_proba).explain_perf(X_test_enc, y_test, name='Logistic Regression')
tree_perf = ROC(tree.predict_proba).explain_perf(X_test_enc, y_test, name='Classification Tree')

show(lr_perf)
show(tree_perf)
show(ebm_perf)
```

Figure 7 - Code to generate performance metrics for each model

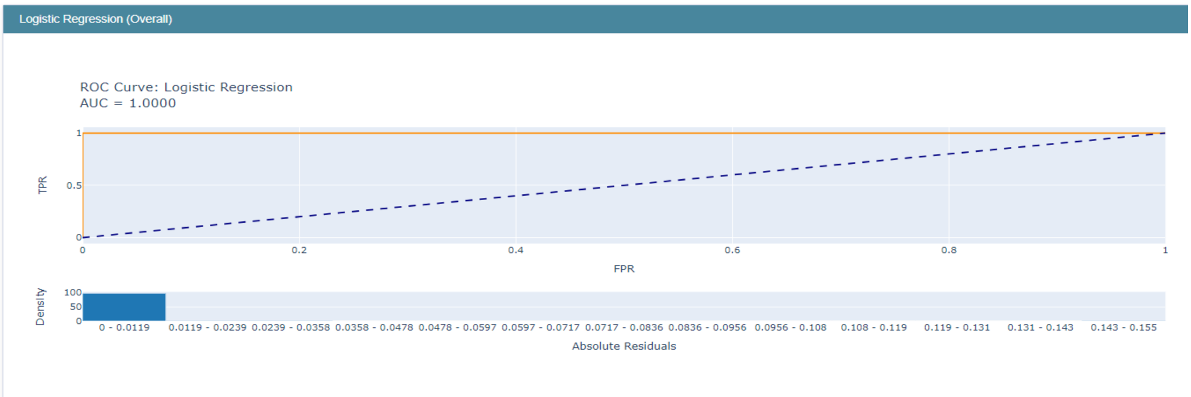


Figure 8 InterpretML: Logistic regression: ROC

Step 9: Glassbox Model: Generating global and local explanations for all models

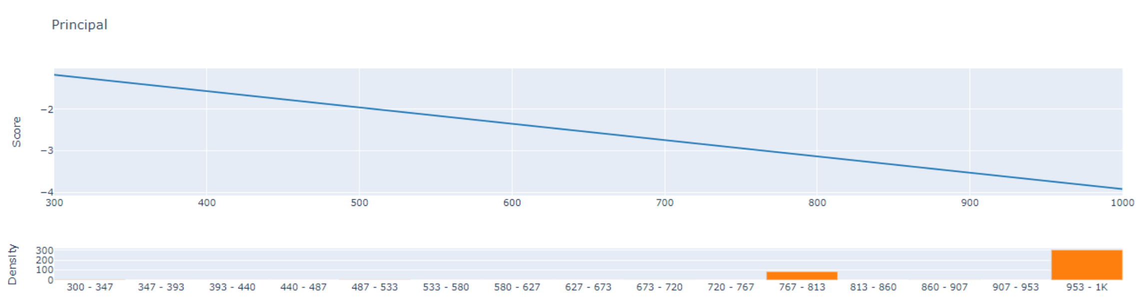
```
lr_global = lr.explain_global(name='Logistic Regression')
tree_global = tree.explain_global(name='Classification Tree')

show(lr_global)
show(tree_global)
show(ebm_bin_global)
```

Select Component to Graph

0 : Name (Principal) | Type (continuous) | # Unique (6)

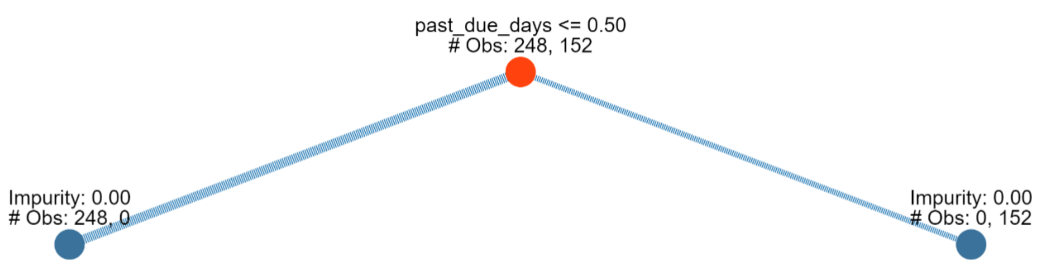
Logistic Regression [0]



Select Component to Graph

2 : Name (past_due_days) | Type (continuous) | # Unique (31)

Classification Tree [2]



Step 10: Dashboard view for all models

7.5 : Dashboard: look at everything once ¶

```
# Do everything in one shot with the InterpretML Dashboard by passing a list into show

show([hist, lr_global, lr_perf, tree_global, tree_perf, ebm_bin_global, ebm_perf], share_tables=True)

#Check what if analysis in interpretML
```

Open in new window

1. The above code would generate a dashboard with 4 tabs:
    - Overview: explains about all sections in the dashboard
    - Data: Select the data/features
    - Performance: Model performance by selecting model type
    - Global: provides global explanations by selecting model type
    - Local: provides row wise explanations

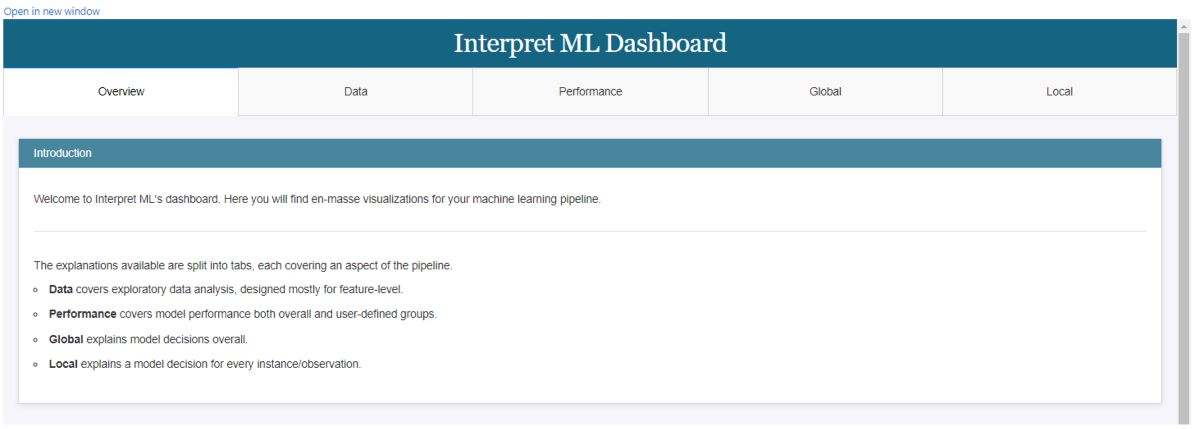


Figure 9 - Dashboard view

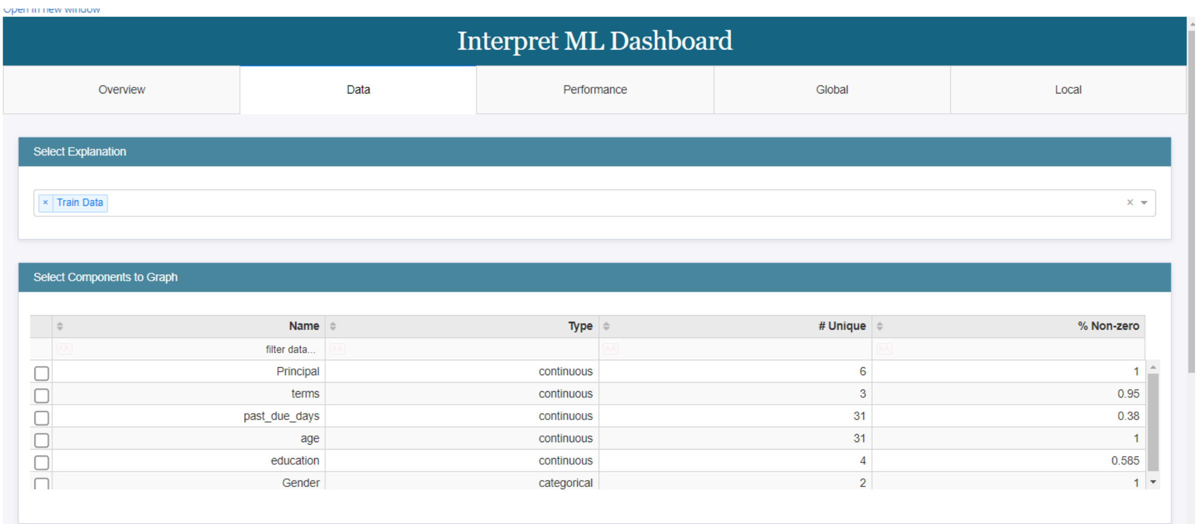


Figure 10 - Data tab selected

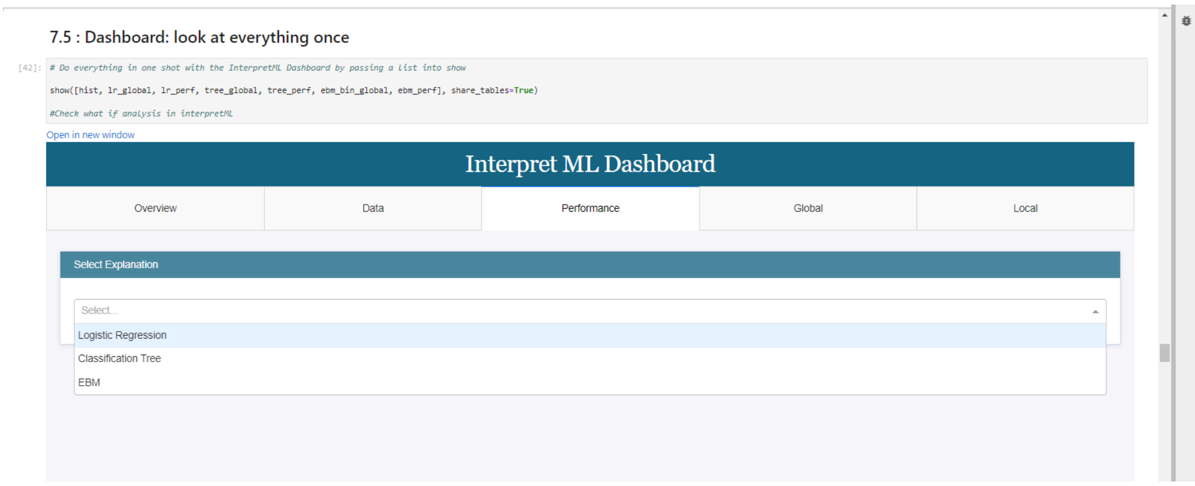


Figure 11 - Select model type from Performance tab

Step 11: Blackbox explainers using sklearn model

Blackbox explanations: These are explanations that can be given for any models coming from any other library such as sklearn

a. They consider **only input and output values** and assume that based on these values they need to **identify the explainability.**
b. In this process, changes are made to inputs and passed through model to analyse the change in model output and thus provide **explainability.**
c. **Local explainers** in blackbox explanations include: **SHAP, LIME**
d. **Global explainers** in Blackbox explanations include: **Partial Dependency Plot** and **Morris Sensitivity.**
e. This can work very well on deep neural nets or on complex ml pipelines

Step 12: Training blackbox model:

8.1 : Training Blackbox - sklearn model

```
]: from sklearn.ensemble import RandomForestClassifier
   from sklearn.pipeline import Pipeline

   #Blackbox system can include preprocessing, not just a classifier!
   rf = RandomForestClassifier(n_estimators=100, n_jobs=-1)

   blackbox_model = Pipeline([('rf', rf)])
   blackbox_model.fit(X_train_enc, y_train)

]: Pipeline(steps=[('rf', RandomForestClassifier(n_jobs=-1))])
```

Figure 12 - Training Sklearn based model

Step 13: Generating model performance from Blackbox model using interpret library

8.2 :Show blackbox model performance

```
from interpret import show
from interpret.perf import ROC

blackbox_perf = ROC(blackbox_model.predict_proba).explain_perf(X_test_enc, y_test, name='Blackbox')
show(blackbox_perf)
```

Figure 13 - Using ROC method from interpret to understand model performance for sklearn based model

Using the code above helps in generating the model performance and showing the ROC curve (refer below) :

Step 14: LIME for local explanations on blackbox model

- Lime tabular comes inbuilt with Interpret library and can provide local explanations for any tabular data:

```
from interpret.blackbox import LimeTabular
from interpret import show

#Blackbox explainers need a predict function, and optionally a dataset
lime = LimeTabular(predict_fn=blackbox_model.predict_proba, data=X_train_enc, random_state=1)

#Pick the instances to explain, optionally pass in labels if you have them
lime_local = lime.explain_local(X_test_enc[:5], y_test[:5], name='LIME')

show(lime_local)
```

Figure 14 - Generating Lime value for tabular data (sklearn models)

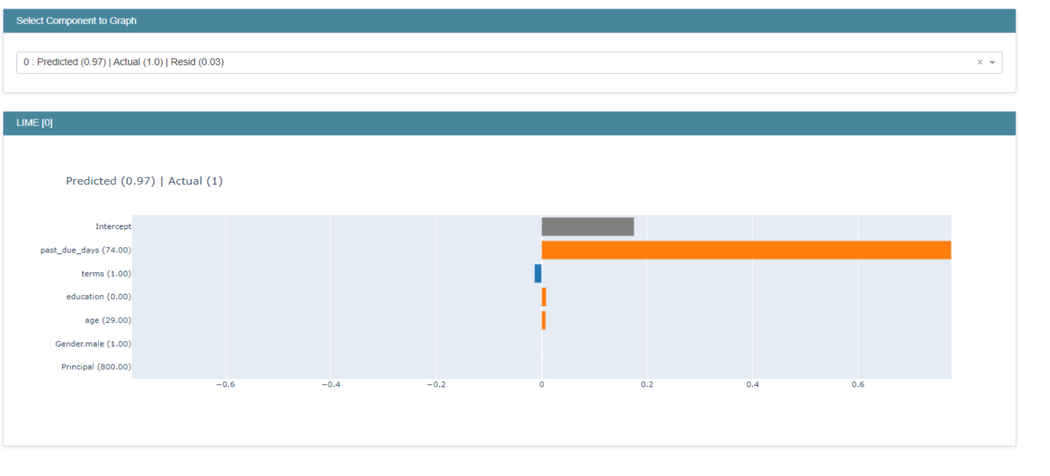


Figure 15 : For 1st rows prediction: individual feature importance

- 'We can select from a dropdown row of the data (here first row = row 0 has been selected)
- **“Past_Due_days”:** is the major contributor in prediction for this row.

Step 15: SHAP for local explanations on blackbox model

- **Code: -**

```
from interpret.blackbox import ShapKernel
import numpy as np

background_val = np.median(X_train_enc, axis=0).reshape(1, -1)
shap = ShapKernel(predict_fn=blackbox_model.predict_proba, data=background_val, feature_names=feature_names)
shap_local = shap.explain_local(X_test_enc[:5], y_test[:5], name='SHAP')
show(shap_local)
```

Figure 16 - Code to generate Shap Value

- **Output: -**
  - Output provides a UI to select the dropdown (row number from the data)
  - Later we can look at how each feature contributes to row wise prediction

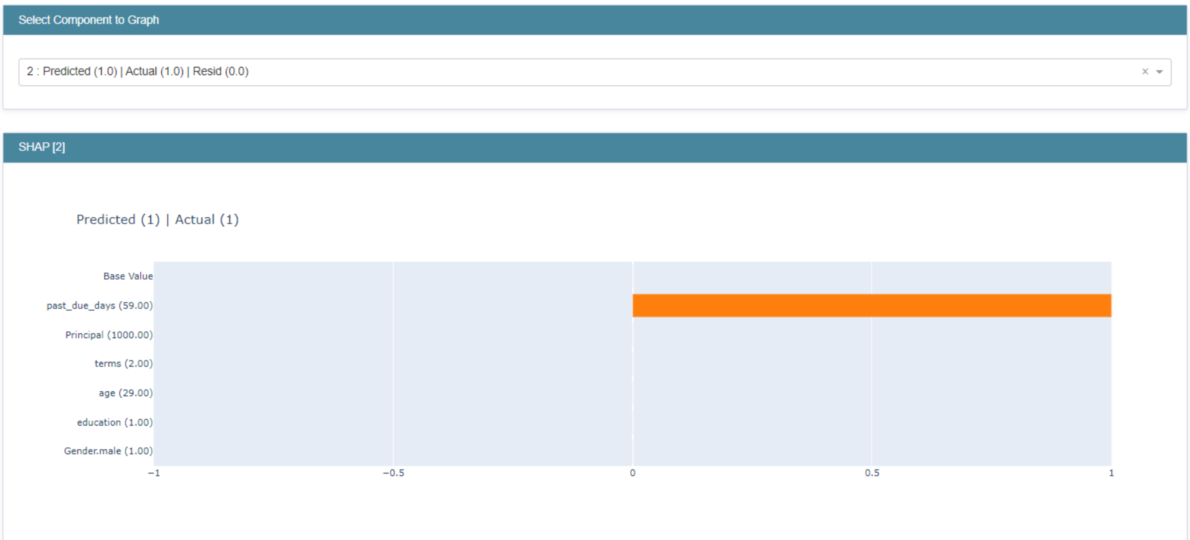


Figure 17- SHAP value: blackbox model

Step 16: Global explanations: Morris sensitivity on blackbox model

```
from interpret.blackbox import MorrisSensitivity

sensitivity = MorrisSensitivity(predict_fn=blackbox_model.predict_proba, data=X_train_enc)
sensitivity_global = sensitivity.explain_global(name="Global Sensitivity")

show(sensitivity_global)
```

Figure 18- Code to generate Morris sensitivy using interpret Library

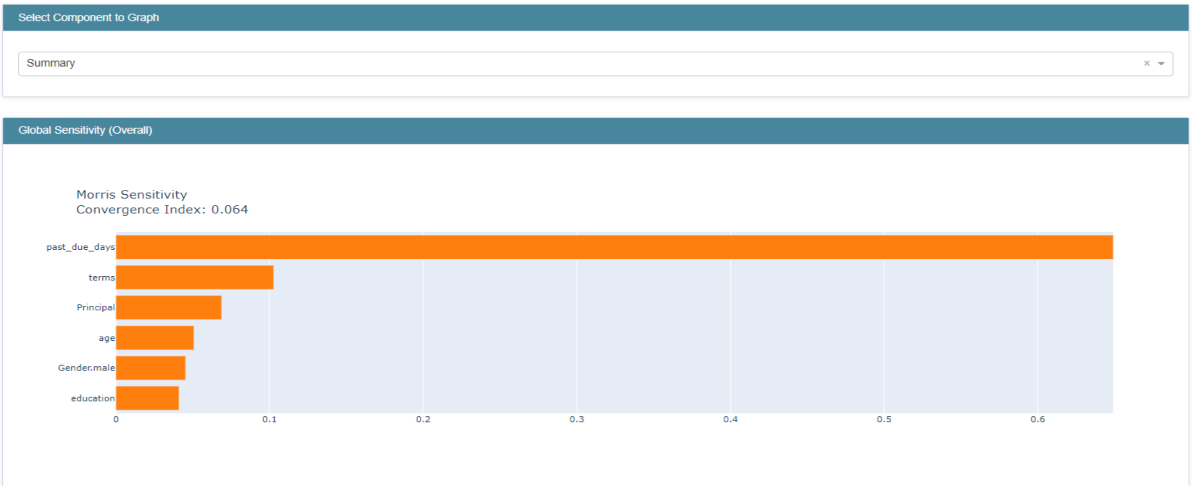


Figure 19 - Output for Morris sensitivity show at a Model level : feature importance

Step 17: Global explanations: Partial Dependence plots on blackbox model

8.4.2 :Partial Dependence Plots ¶

Reference read: Christopher's book InterpretML Docs

```
from interpret.blackbox import PartialDependence

pdp = PartialDependence(predict_fn=blackbox_model.predict_proba, data=X_train_enc)
pdp_global = pdp.explain_global(name='Partial Dependence')

show(pdp_global)
```

Figure 20 - Code to use interpret library to generate PDP

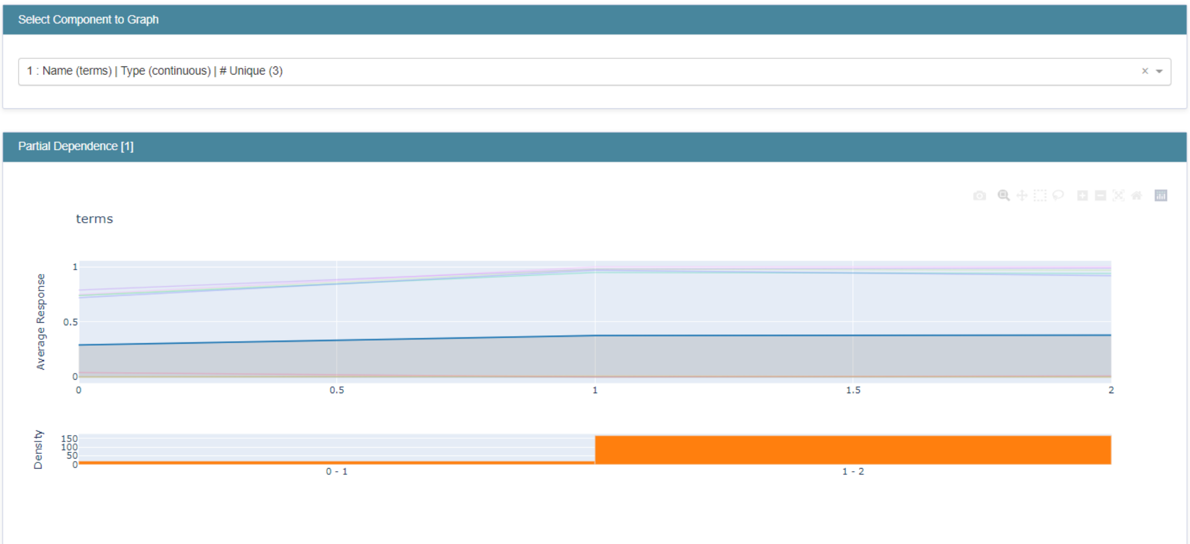


Figure 21 : Select features from dropdown

- For different values of a feature, we can see how much it is contributing to the final model
- This would help in doing a **what-if analysis** later on
- *In above output: The model prediction gets impacted with values between 1-2 for the feature: Term.*

Step 18: Comparing all explainer Dashboard: Blackbox model

8.5 : Comparing all explainers in single view

```
show([blackbox_perf, lime_local,shap_local,sensitivity_global, pdp_global])
```

Open in new window

Figure 22 : Code to generate dashboard using interpret library

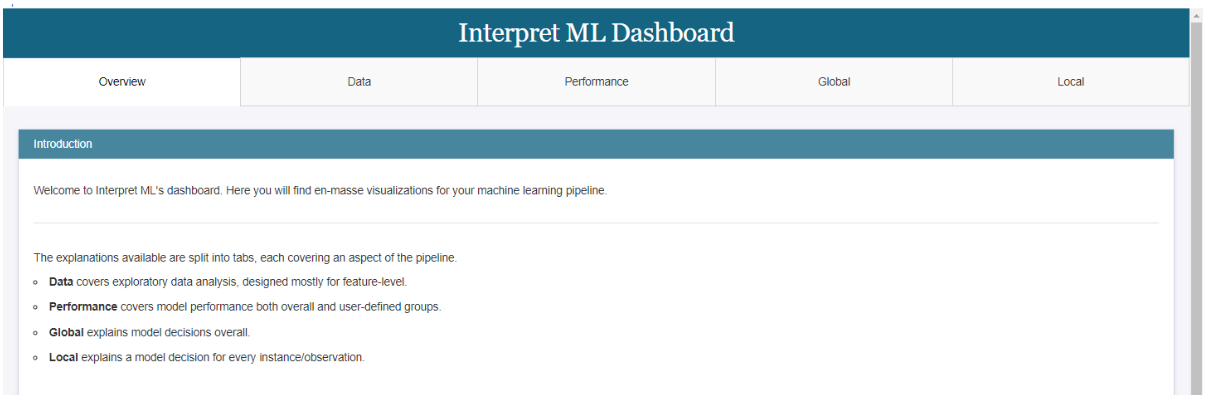


Figure 23 - 5 tabs are generated

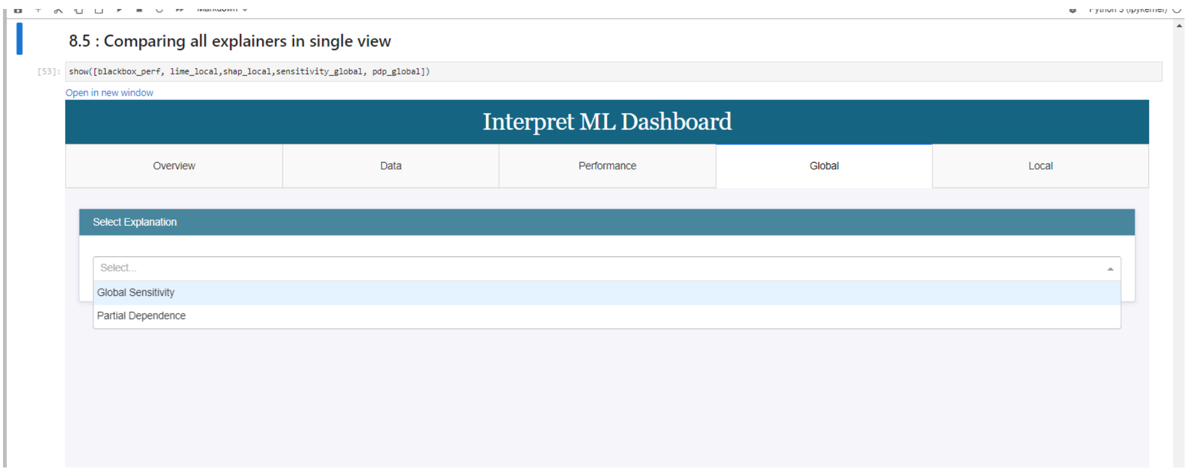


Figure 24: Option to select different global explainers from dropdown

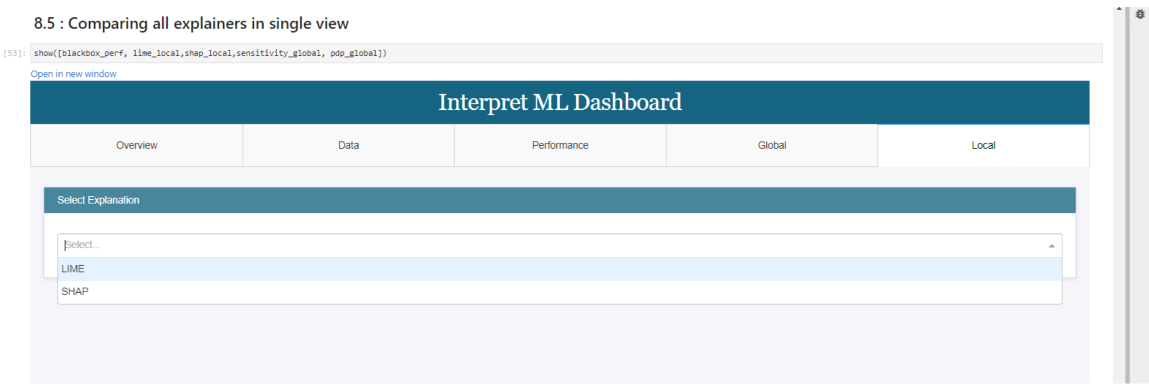


Figure 25 - Option to select different local explainers from dropdown (SHAP, LIME)