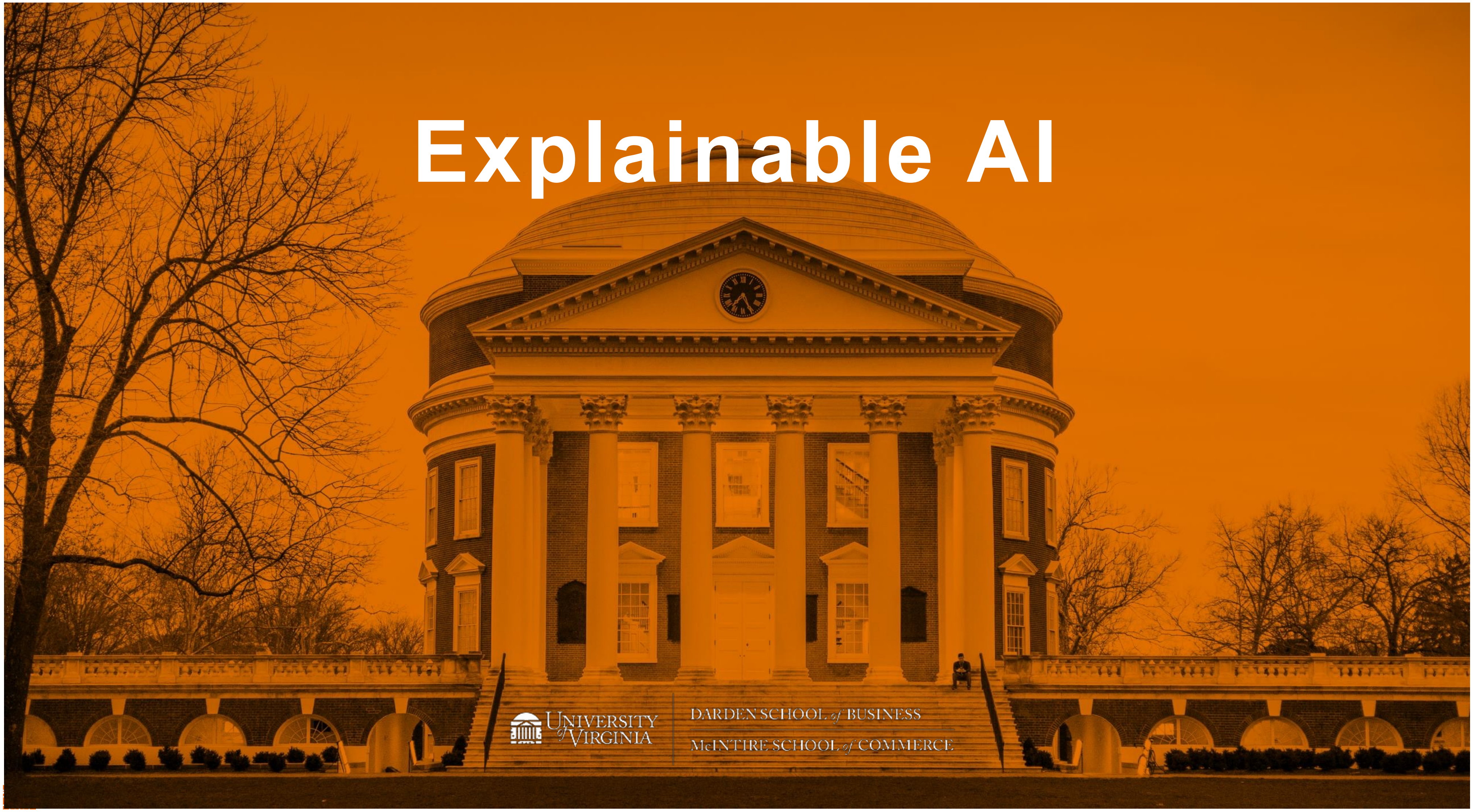
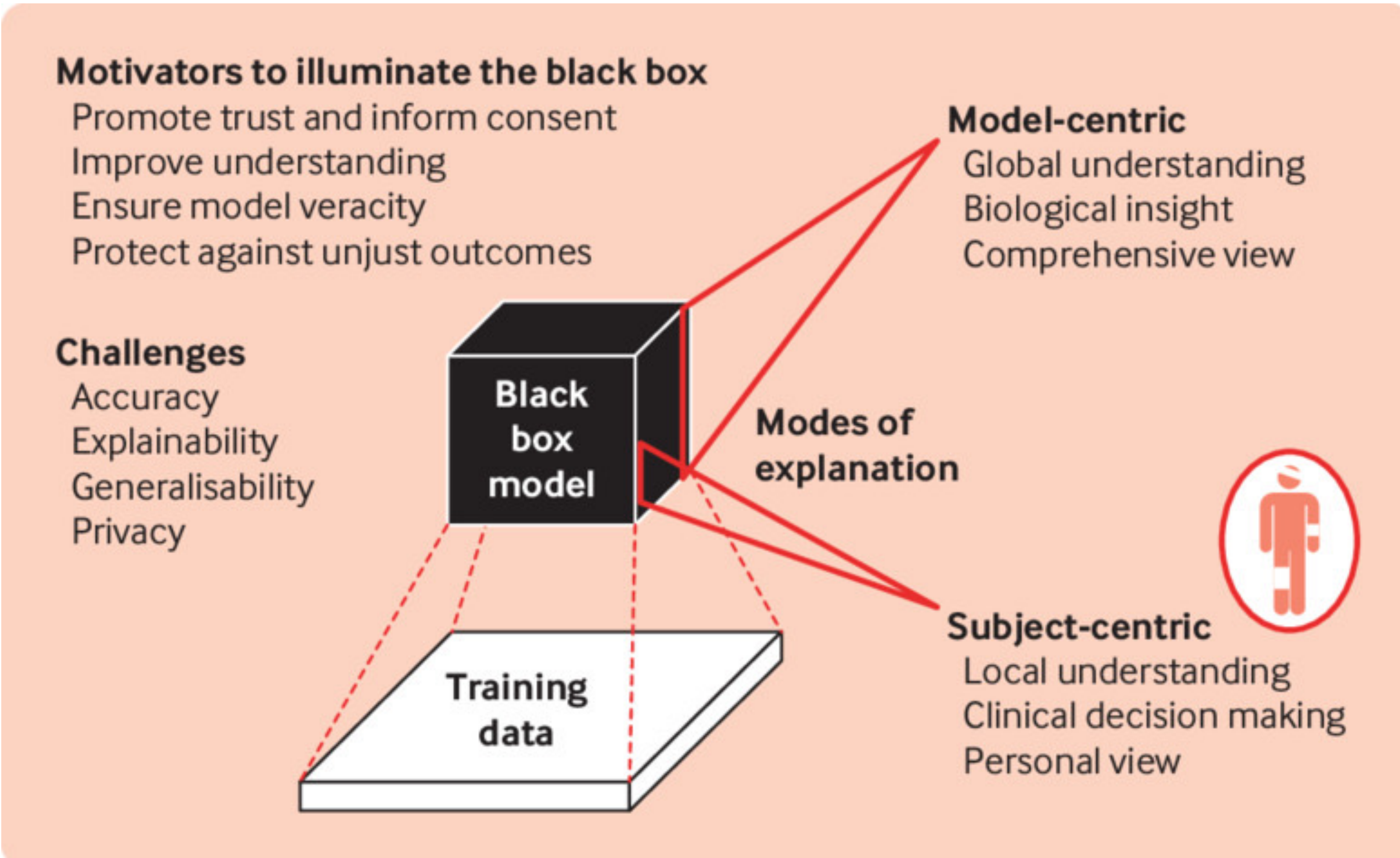
Explainable AI




Motivators to illuminate the black box
Promote trust and inform consent
Improve understanding
Ensure model veracity
Protect against unjust outcomes
Challenges
Accuracy
Explainability
Generalisability
Privacy
Black box model
Training data
Modes of explanation
Model-centric
Global understanding
Biological insight
Comprehensive view
Subject-centric
Local understanding
Clinical decision making
Personal view

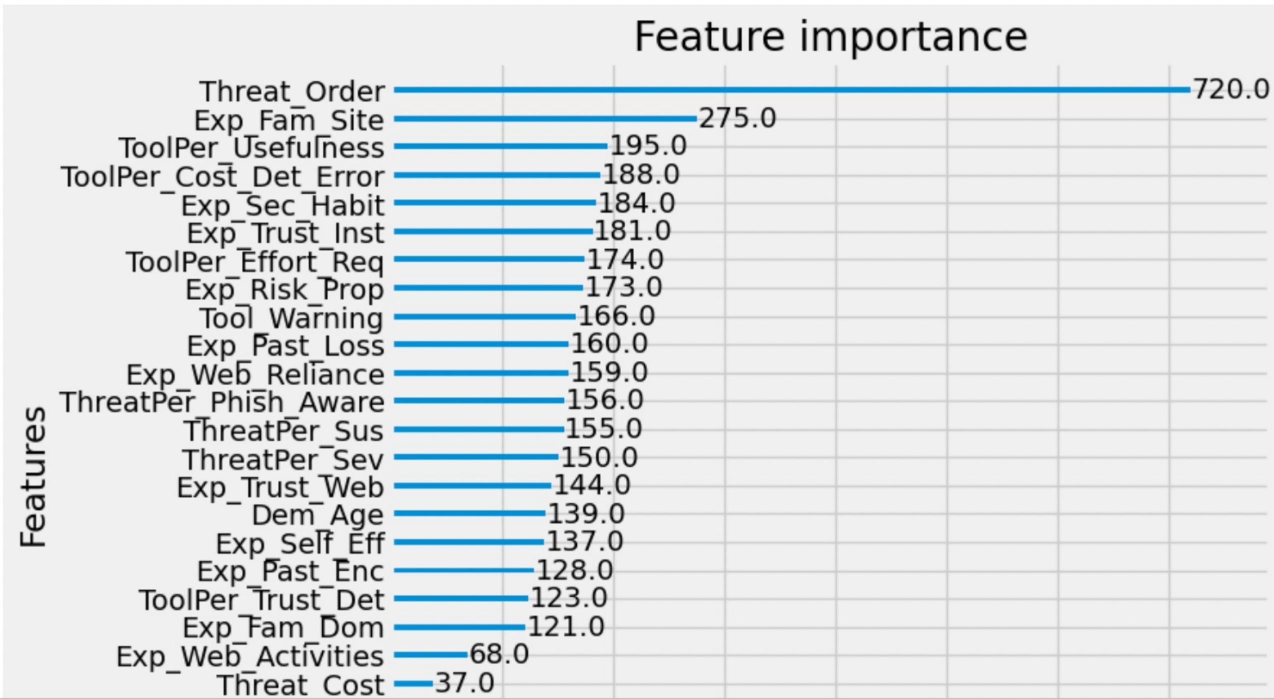
Explainable AI Cheat Sheet

Approach Name	Model Centric	Subject Centric
Feature Ranking	plot_importance(clf, …)	
Recursive Feature Engineering (RFE)	RFE(clf, …)	
Boruta	BorutaPy(estimator=clf, …)	
Logistic Regression	sm.Logit(y, X).fit()	
Shap	shap.summary_plot(shapValues, …) shap.plots.scatter(shapValuess[:,"col1"], …)	shap.force_plot(…) shap.plots.waterfall(…)
Lime		explainer.explain_instance(…)
iModels	HSTreeClassifierCV(max_leaf_nodes=6)	
Explainer Dashboard	ClassifierExplainer(clf, testData, testLabels)	ClassifierExplainer(clf, testData, testLabels)

Model Centric Approach 1: Feature Ranking

```python
from xgboost import plot_importance

# plot feature importance
plot_importance(clf, importance_type = "weight")
```

```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Fe
atures'>
```



Model Centric Approach 2: RFE

```
from sklearn.feature_selection import RFE

selector = RFE(clf, n_features_to_select=5, # Max number of features desired
               step=1)
selector = selector.fit(trainData, trainLabels)
rfe_results = pd.DataFrame({"Feature":trainData.columns,
                            "Selected":selector.support_})
rfe_results.sort_values('Selected', ascending = False)
```

	Feature	Selected
4	Threat_Cost	True
23	Exp_Fam_Dom	True
22	ToolPer_Trust_Det	True
21	ToolPer_Usefulness	True
20	ToolPer_Cost_Det_Error	True
0	Tool_Det_Accuracy	False
15	ThreatPer_Phish_Aware	False
26	Threat_Order	False
25	Exp_Web_Activities	False

Model Centric Approach 3: Boruta

```python
from boruta import BorutaPy

# let's initialize Boruta
feat_selector = BorutaPy(
    verbose=2,
    estimator=clf,
    n_estimators='auto',
    max_iter=10  # number of iterations to perform
)

# train Boruta
# N.B.: X and y must be numpy arrays
feat_selector.fit(np.array(trainData),np.array(trainLabels))

boruta_results = pd.DataFrame({"Feature":trainData.columns,
                               "Selected":feat_selector.support_})
boruta_results.sort_values('Selected', ascending = False)
```

```
BorutaPy finished running.

Iteration:      9 / 10
Confirmed:      28
Tentative:      0
Rejected:       0
```

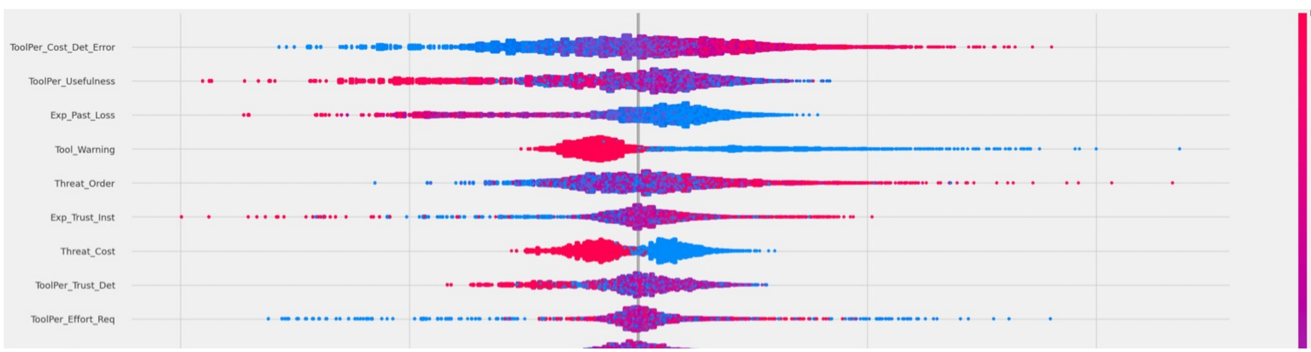
Model Centric Approach 4: Logistic Regression

```python
import statsmodels.api as sm
X = sm.add_constant(trainData)
log_reg = sm.Logit(trainLabels, X).fit()
print(log_reg.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                54384
Model:                          Logit   Df Residuals:                    54355
Method:                           MLE   Df Model:                           28
Date:                Tue, 13 Dec 2022   Pseudo R-squ.:                  0.1199
Time:                        02:08:30   Log-Likelihood:                -30935.
converged:                       True   LL-Null:                       -35149.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
==========
                          coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----------
const                  -0.7194      0.092     -7.781      0.000      -0.901
-0.538
Tool_Det_Accuracy      -0.1067      0.022     -4.891      0.000      -0.149
-0.064
Tool_Det_Time          -0.0793      0.020     -3.988      0.000      -0.118
-0.040
```

Model Centric Approach 5: SHAP

```
import shap

shap.initjs() # Please do not change this. We use this to create the Shapley plot
explainer = shap.TreeExplainer(clf) # PLEASE DO NOT CHANGE THIS.
shapValues = np.array(explainer.shap_values(trainData))
plt_shap = shap.summary_plot(shapValues, #Use Shap values array
                             features=trainData, # Use training set features
                             feature_names=trainData.columns, #Use column names
                             # show=False, #Set to false to output to folder
                             plot_size=(30,15)) # Change plot size
```



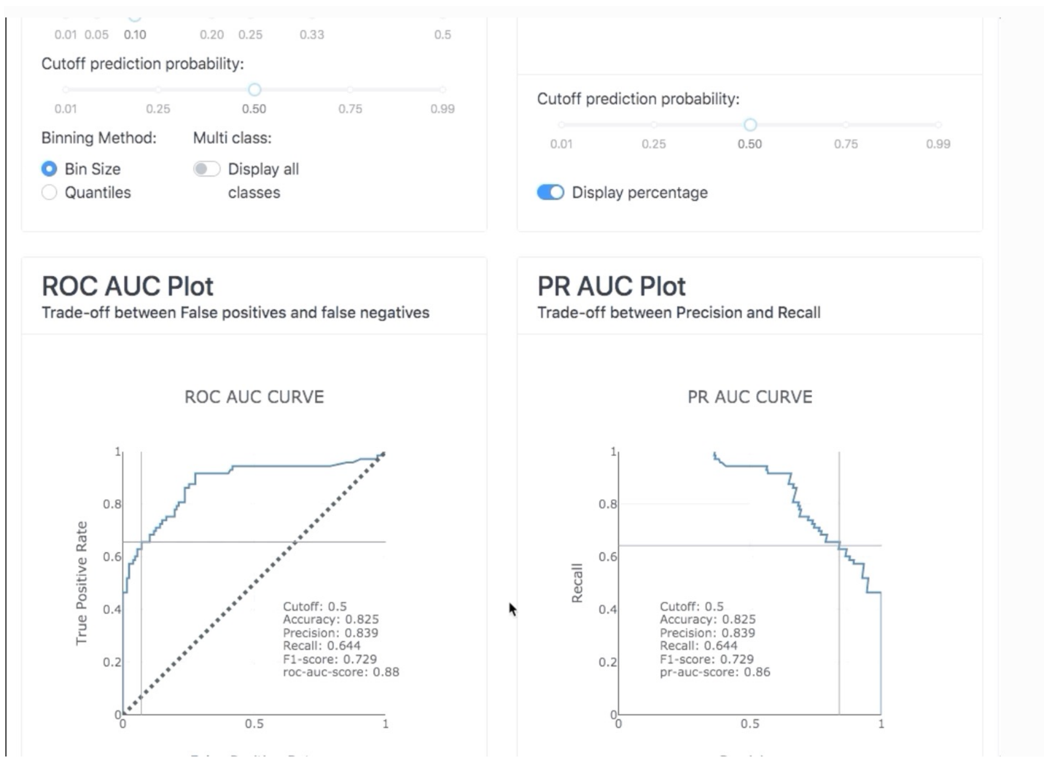
Model Centric Approach 6: iModels

```
from imodels import HSTreeClassifierCV # import any imodels model here

# fit the model
model = HSTreeClassifierCV(max_leaf_nodes=6)  # initialize a tree model and speci
model.fit(trainData, trainLabels,
          feature_names=list(trainData.columns))   # fit model
preds = model.predict(testData) # discrete predictions: shape is (n_
preds_proba = model.predict_proba(testData) # predicted probabilitie
print(model) # print the model
```

```
> ------------------------------
|--- feature_21 <= 7.06
|   |--- feature_24 <= 1.25
|   |   |--- feature_8 <= 5.10
|   |   |   |--- weights: [0.77, 0.23] class: 0.0
|   |   |--- feature_8 >  5.10
|   |   |   |--- feature_27 <= 0.50
|   |   |   |   |--- weights: [0.33, 0.67] class: 1.0
|   |   |   |--- feature_27 >  0.50
|   |   |   |   |--- weights: [0.54, 0.46] class: 0.0
|   |--- feature_24 >  1.25
|   |   |--- weights: [0.66, 0.34] class: 0.0
|--- feature_21 >  7.06
|   |--- feature_27 <= 0.50
|   |   |--- weights: [0.59, 0.41] class: 0.0
|   |--- feature_27 >  0.50
|   |   |--- weights: [0.82, 0.18] class: 0.0
```

Model Centric Approach 7: Explainer Dashboard

```
from explainerdashboard import ClassifierExplainer, ExplainerDashboard
explainer = ClassifierExplainer(clf, testData, testLabels)
ExplainerDashboard(explainer, mode='inline').run(port=8051)
```



Subject Centric Approach 1: SHAP (1 of 2)

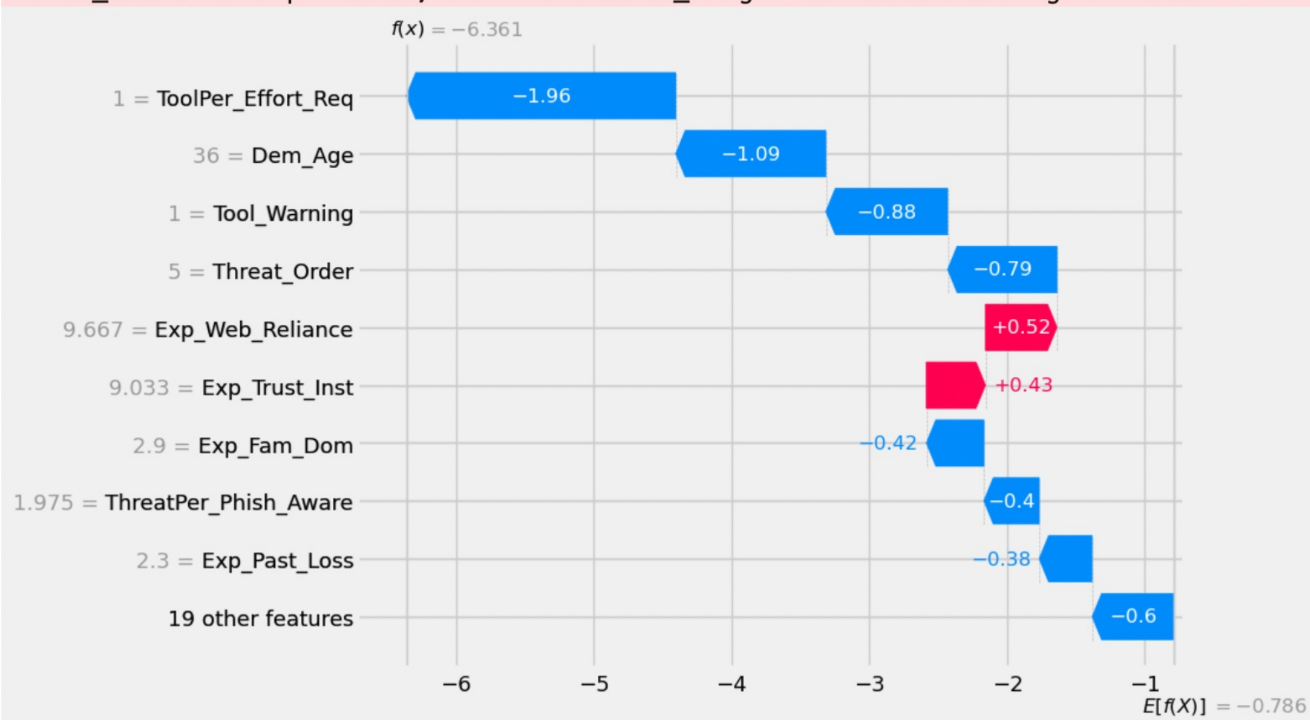
```
explainer = shap.Explainer(clf)
shap_values = explainer.shap_values(trainData)

index = 0 # Row index, 0
shap.force_plot(explainer.expected_value, shap_values[index,:],
                trainData.iloc[index,:])
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

Subject Centric Approach 1: SHAP (2 of 2)

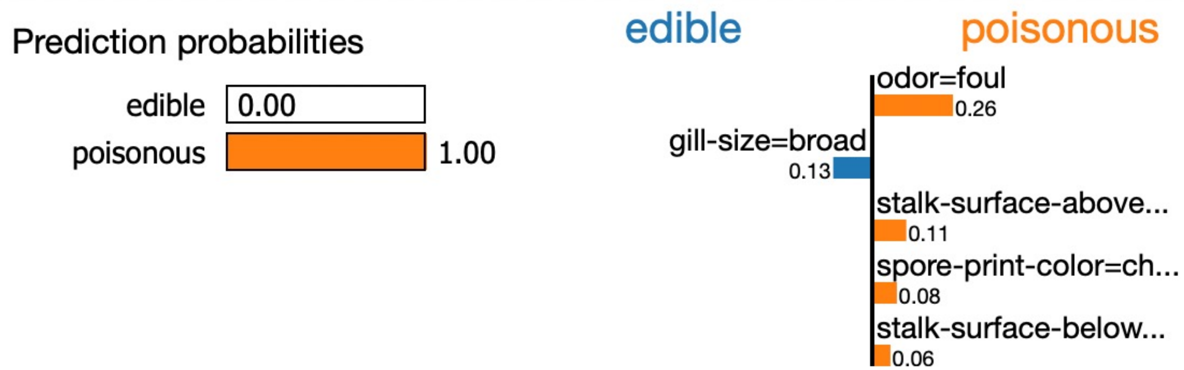
```
explainer = shap.Explainer(clf)
shap_values = explainer(trainData)

# visualize the first prediction's explanation
shap.plots.waterfall(shap_values[0])
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```



Subject Centric Approach 2: Lime

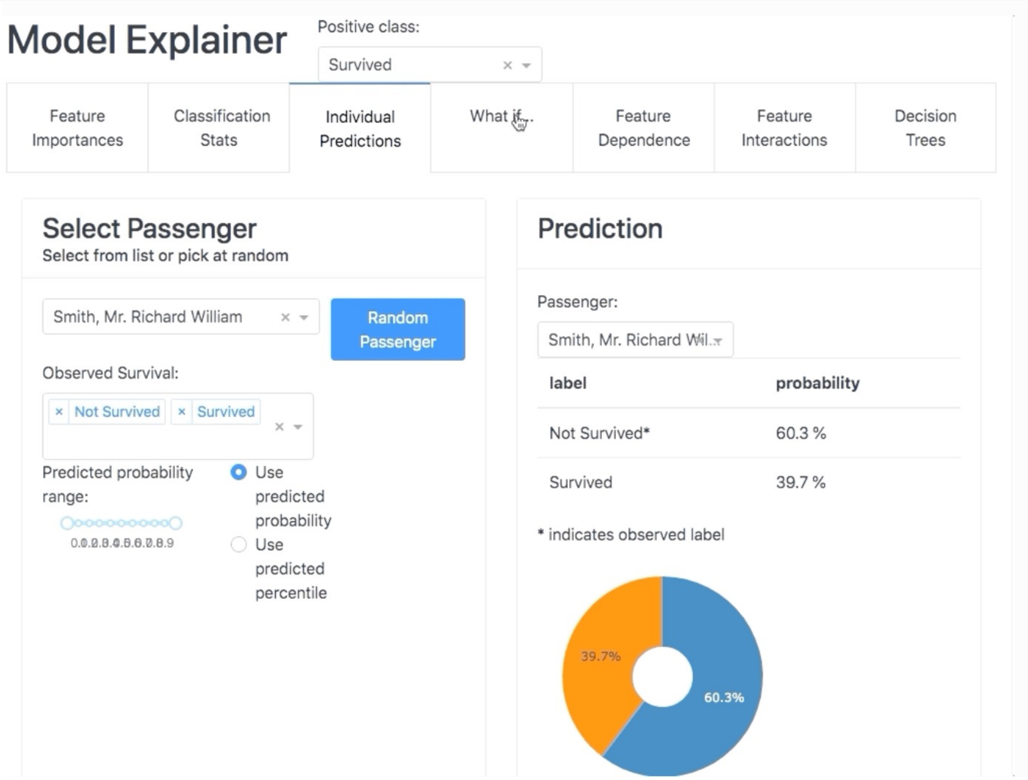
```
exp = explainer.explain_instance(test[i], predict_fn, num_features=5)
exp.show_in_notebook()
```



Feature	Value
odor=foul	True
gill-size=broad	True
stalk-surface-above-ring=silky	True
spore-print-color=chocolate	True
stalk-surface-below-ring=silky	True

Subject Centric Approach 3: Explainer Dashboard

```
from explainerdashboard import ClassifierExplainer, ExplainerDashboard
explainer = ClassifierExplainer(clf, testData, testLabels)
ExplainerDashboard(explainer, mode='inline').run(port=8051)
```



?

VIRGINIA