OmniXAI in a ML workflow Using Shark Research Data (August 2017-2022)

This tutorial shows how to apply XAI in different stages in a standard ML workflow. The OmniXai library is used in the example. The goal is to use prepared data from transmitters and receivers off of the Coast of Cape Cod to predict shark presence (0 or 1) in the month of August. The data is limited and prepared for use during WiDS Charlotte 2025, and can only be used for educational, not research purposes.

```
%%capture
import warnings
warnings.simplefilter("ignore")
!pip install omnixai
!pip install kaleido
# This default renderer is used for sphinx docs only. Please delete this cell in IPythor
import plotly.io as pio
pio.renderers.default = "png"
import os
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
```

The partial dataset used in this example is July 10 to August 10, allowing for a more balanced target class "#sharks".

The dataset in this notebook can be read from: GitHub

```
# fetch dataset from github
# Convert to DataFrame
# URL of the raw dataset without 15% of the data (holdout for prediction and validation
url = "https://raw.githubusercontent.com/DrPamelaThompson/WiDS-Charlotte-2025/refs/heads
# Read the CSV
df = pd.read_csv(url)
#drop na not necessary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8887 entries, 0 to 8886
Data columns (total 22 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   #sharks                        8887 non-null   int64
 1   dewF                           8887 non-null   float64
 2   tempF_air                      8887 non-null   float64
 3   datetime                       8887 non-null   object
 4   tau_gust_mag                   8887 non-null   float64
 5   taux_gust                      8887 non-null   float64
```

```
 6   tempF_water_cat                 8887 non-null   int64
 7   stability_value                 8887 non-null   float64
 8   tempF_water                     8887 non-null   float64
 9   precip                          8887 non-null   float64
 10  alpha                           8887 non-null   float64
 11  Tide_water_level_cat_encoded    8887 non-null   int64
 12  unique_shark_count_cat_encoded  8887 non-null   int64
 13  Tide_ebb_flood_cat_encoded      8887 non-null   int64
 14  press_change_cat_encoded        8887 non-null   int64
 15  Light                           8887 non-null   object
 16  tempF_Diff_encoded              8887 non-null   int64
 17  press                           8887 non-null   float64
 18  Moon_Phase_cat                  8887 non-null   object
 19  Year                            8887 non-null   int64
 20  Month                           8887 non-null   int64
 21  Day                             8887 non-null   int64
dtypes: float64(9), int64(10), object(3)
memory usage: 1.5+ MB
```

```
1 df.value_counts('#sharks')
```

	count
#sharks	
0	4484
1	4403

dtype: int64

```
1 df.tail()
```

	#sharks	dewF	tempF_air	datetime	tau_gust_mag	taux_gust	tempF_water_
8882	0	65.833333	72.233333	2022-08-10 21:30:00+00:00	0.016946	0.013035	
8883	1	65.366667	72.000000	2022-08-10 22:00:00+00:00	0.017251	0.011803	
8884	1	65.800000	71.400000	2022-08-10 22:30:00+00:00	0.020704	0.010926	
8885	0	66.000000	71.066667	2022-08-10 23:00:00+00:00	0.013833	0.008830	
8886	1	66.766667	71.000000	2022-08-10 23:30:00+00:00	0.011247	0.006504	

5 rows × 22 columns

```
#check balance of second target class
df.value_counts('unique_shark_count_cat_encoded')
```

unique_shark_count_cat_encoded	count
0	4484
1	4347
2	56

dtype: int64

July 12 and 13, 2022: Severe weather affected the region, with reports of heavy rainfall and strong winds impacting Cape Cod. Record numbers will be extracted for later investigation.

```
# Convert datetime column to datetime format (if not already)
df["datetime"] = pd.to_datetime(df["datetime"])

# Find the index (record number) for July 12, 2022
record_index = df[df["datetime"] == "2022-07-12"].index

# Display the record number(s)
print("Record number(s) for July 12, 2022:", record_index.tolist())
```

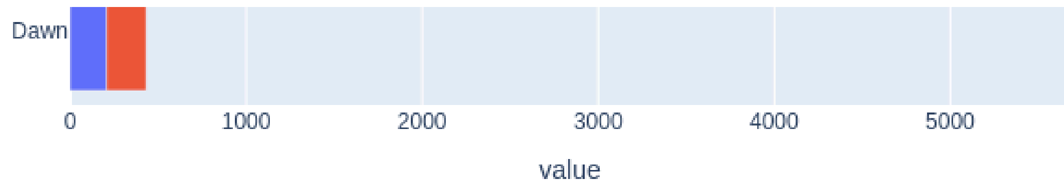
```
Record number(s) for July 12, 2022: [7529]
```

The target is #sharks and a second target, not used, is unique_shark_count_cat_encoded

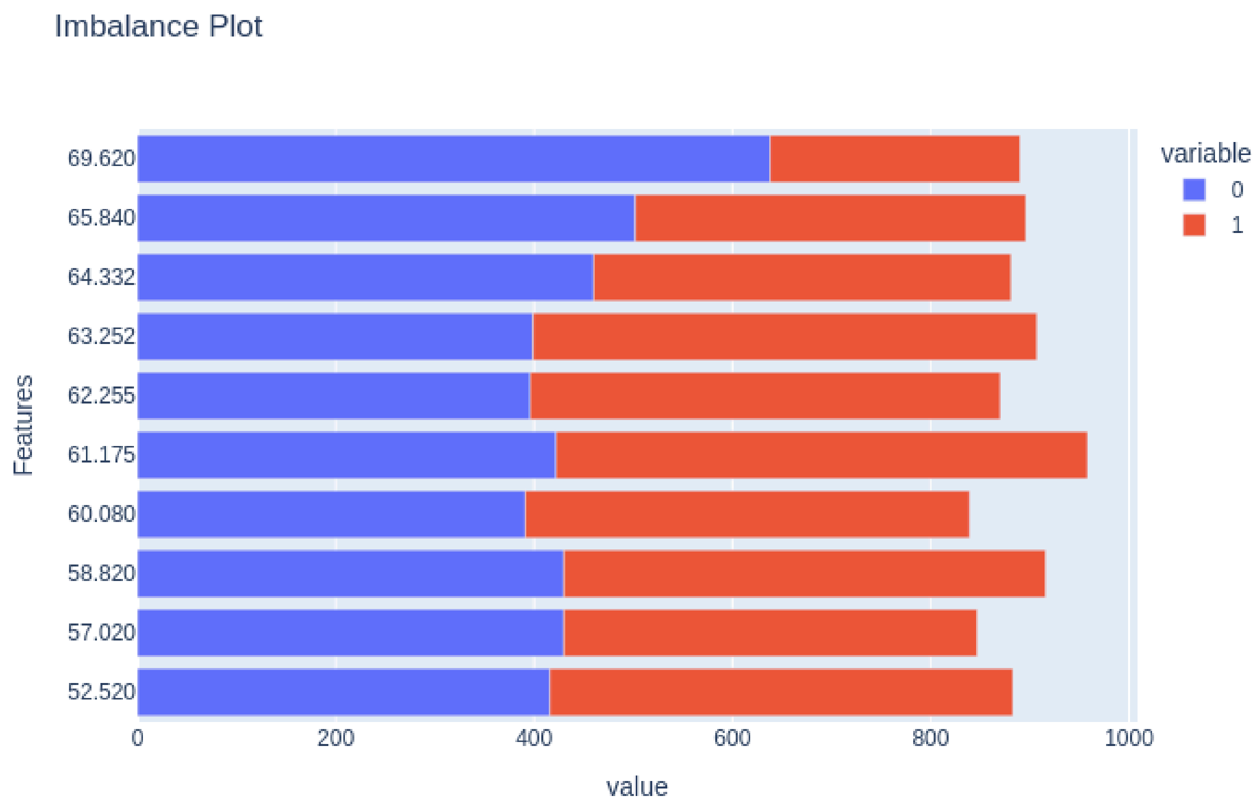
Let's first check if some features are correlated and if there exists data imbalance issues that leads to any potential bias. We can create an `DataAnalyzer` explainer from OmniXai to do this task.

```
df1 = df.drop(columns=['datetime','unique_shark_count_cat_encoded','Year','Month','Day']
```

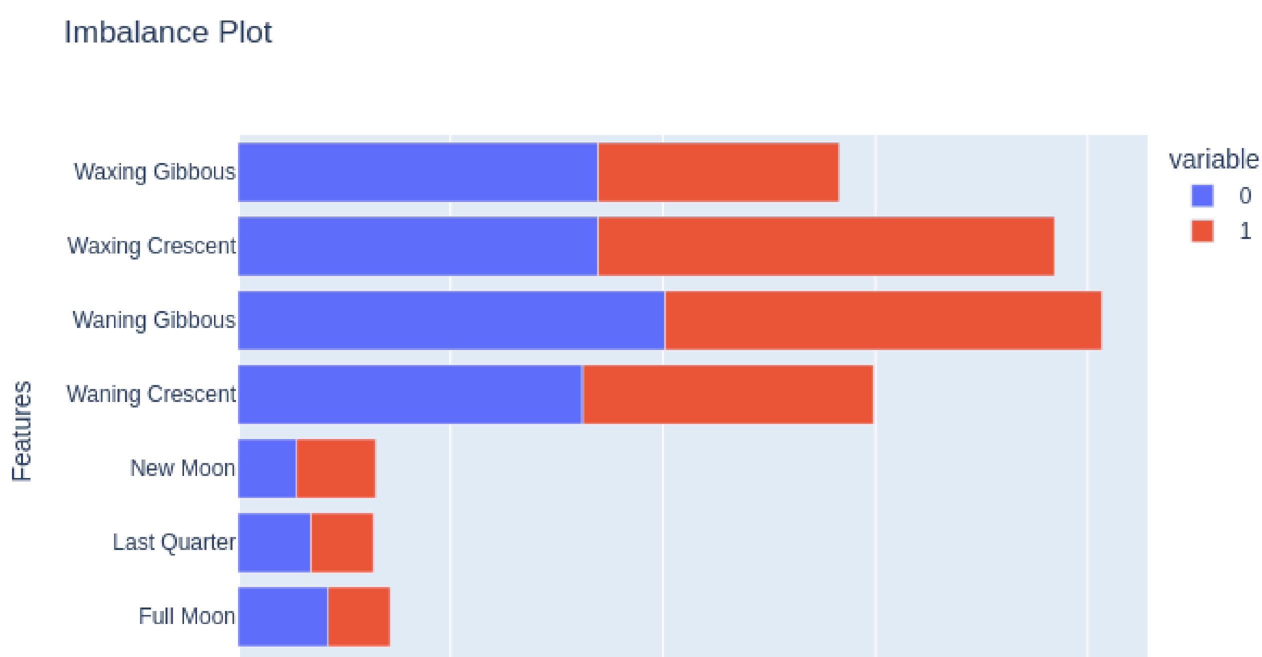
```
from omnixai.data.tabular import Tabular
from omnixai.explainers.data import DataAnalyzer

tabular_data = Tabular(
    df1,
    categorical_columns=['Light','Moon_Phase_cat'],
    target_column='#sharks'
)

# Initialize a `DataAnalyzer` explainer.
# We can choose multiple explainers/analyzers by specifying analyzer names.
# In this example, the first explainer is for feature correlation analysis and
# the others are for feature imbalance analysis (the same explainer with different param
explainer = DataAnalyzer(
    explainers=["correlation", "imbalance#0", "imbalance#1", "imbalance#2"],
    mode="classification",
    data=tabular_data
)
# Generate explanations by calling `explain_global`.
explanations = explainer.explain_global(
    params={"imbalance#0": {"features": ["Light"]},
            "imbalance#1": {"features": ["tempF_water"]},
            "imbalance#2": {"features": ["Moon_Phase_cat"]},
            "imbalance#3": {"features": ["Light", "tempF_water"]}}
)

print("Correlation:")
explanations["correlation"].ipython_plot()
print("Imbalance#0: features tempF_water_cat")
explanations["imbalance#0"].ipython_plot()
print("Imbalance#1: features tempF_water")
explanations["imbalance#1"].ipython_plot()
print("Imbalance#2: features Light, tempF_water")
explanations["imbalance#2"].ipython_plot()
```




```
Imbalance#1: features tempF_water
```




```
Imbalance#2: features Light, tempF_water
```




First Quarter

From the correlation plot we can observe that "tempF_water" has strong correlations with "tempF_water_cat", so we may remove one of these features. We will also remove taus_gust based on domain information. From the data imbalance plots we can see that the class labels are relatively balanced in the features.

```
df2 = df1.drop(columns=["tempF_water_cat", "taux_gust"])
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8887 entries, 0 to 8886
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   #sharks                      8887 non-null   int64  
 1   dewF                         8887 non-null   float64
 2   tempF_air                    8887 non-null   float64
 3   tau_gust_mag                 8887 non-null   float64
 4   stability_value              8887 non-null   float64
 5   tempF_water                  8887 non-null   float64
 6   precip                       8887 non-null   float64
 7   alpha                        8887 non-null   float64
 8   Tide_water_level_cat_encoded 8887 non-null   int64  
 9   Tide_ebb_flood_cat_encoded   8887 non-null   int64  
 10  press_change_cat_encoded     8887 non-null   int64  
 11  Light                        8887 non-null   object 
 12  tempF_Diff_encoded           8887 non-null   int64  
 13  press                        8887 non-null   float64
 14  Moon_Phase_cat               8887 non-null   object 
dtypes: float64(8), int64(5), object(2)
memory usage: 1.0+ MB
```

In the next step, we do a rough feature selection by analyzing the information gain and chi-squared stats between features and targets.

Interpretation:

- Mutual Information tells you how much knowing a feature reduces uncertainty about the target.
- Chi-Square tells you whether there is a statistically significant relationship between a categorical feature and the target.

Practical Explanation Using Your Code

- If "Mutual Information" for Light is high, it means Light helps predict #sharks well.
- If "Chi-Square" for Light is significant, it means LIght and #sharks are not independent, but it doesn’t tell us how much it improves prediction. Chi-Square goes from 0 to infinity - the larger the value the more the dependence between the features.

Mutual information is better for feature selection in machine learning, while chi-square is useful for statistical hypothesis testing.

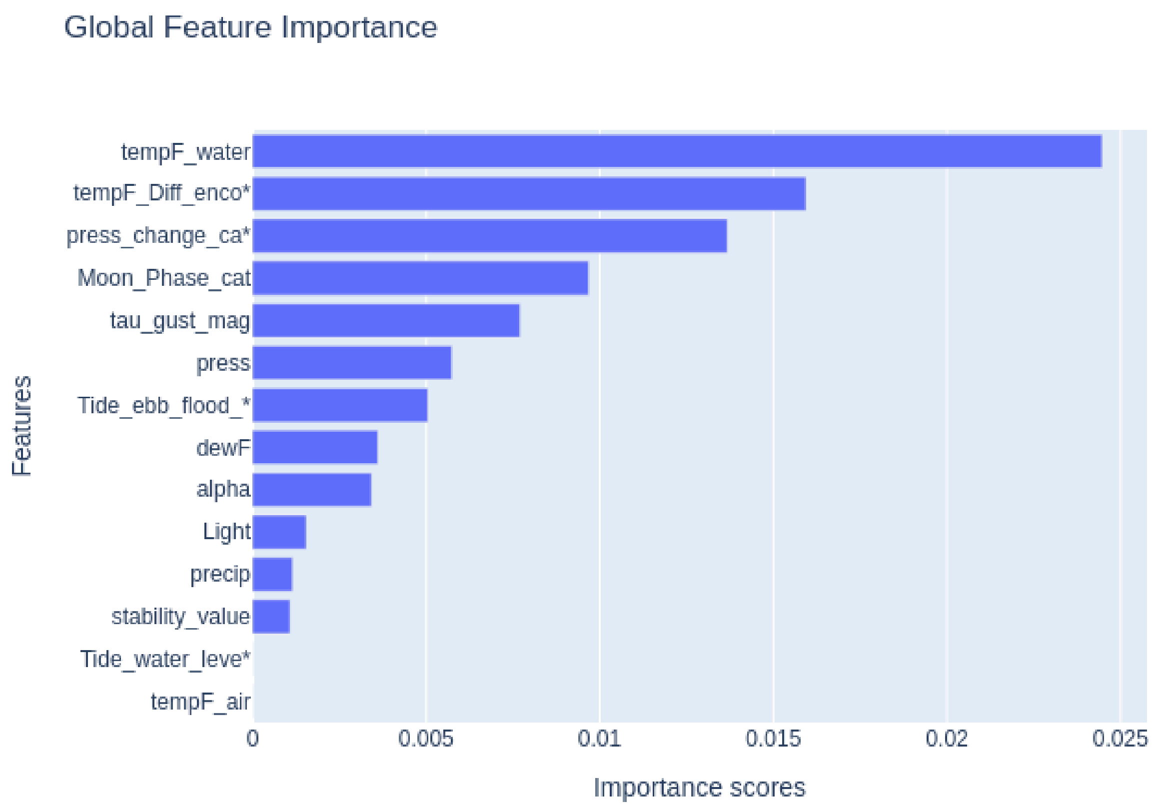
Example: Let's say we're analyzing Education Level vs. Income Group:
Chi-Square might only tell us if there's a relationship.
Mutual Information tells us how much knowing Education Level helps predict Income Group.

Feature Importance for model can be explained globally or locally:

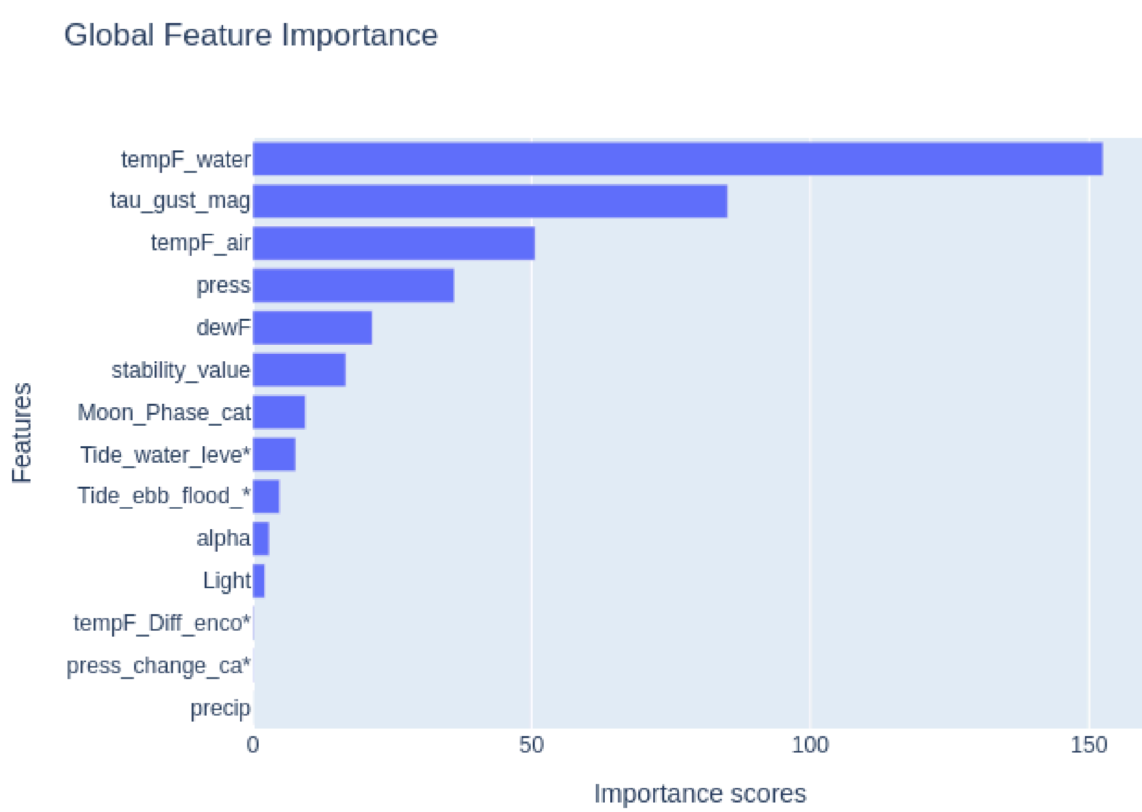
Global Feature Importance: How important each feature is for the entire dataset.
Local Feature Importance: How important each feature is for a single prediction.

```
tabular_data = Tabular(
    df2,
    categorical_columns=['Light', 'Moon_Phase_cat'],
    target_column='#sharks'
)
explainer = DataAnalyzer(
    explainers=["mutual", "chi2"],
    mode="classification",
    data=tabular_data
)
data_explanations = explainer.explain_global()

print("Mutual information:")
data_explanations["mutual"].ipython_plot()
print("Chi square:")
data_explanations["chi2"].ipython_plot()
```

```
Mutual information:
```




```
Chi square:
```



The most important features showed above are "tempF_water", "tau_gust_mag", "tempF_air", "press", "dewF", "stability_value", "Moon_Phase_cat", "Tide_water_level", "Tide_ebb_flood_cat", and "alpha". Domain information will guide us to keep all but tempF_Diff_encoded" and "press_change_cat" in the dataframe for modeling.

```
# We drop this features because the feature has relatively low importance scores.
df3 = df2.drop(columns=["tempF_Diff_encoded"])
tabular_data = Tabular(
    df3,
    categorical_columns=['Light', 'Moon_Phase_cat'],
    target_column='#sharks'
)
#print(tabular_data)
```

Train two models:
1. XGBoost classifier
2. Random Forest Classifier

Training XGBoost and Random Forest together can be useful because they have different strengths and weaknesses. Using both allows you to compare performance, interpret feature importance, and potentially combine their outputs for better predictions.

Factor	When to Use RF	When to Use XGBoost
Dataset Size	Small to medium	Large datasets
Missing Data	Handles missing values well	Requires imputation
Overfitting Risk	Lower	Higher, but tunable
Training Time	Faster on small datasets	Faster on large datasets
Interpretability	Easier to explain	Harder to interpret
Performance on Complex Data	Good, but may miss subtle patterns	Captures complex relationships better

```
import sklearn
import xgboost
from omnixai.preprocessing.tabular import TabularTransform

np.random.seed(12345)
# Train an XGBoost model
transformer = TabularTransform().fit(tabular_data)
x = transformer.transform(tabular_data)
train, test, train_labels, test_labels = sklearn.model_selection.train_test_split(
    x[:, :-1], x[:, -1], train_size=0.80, stratify=x[:, -1]
)
print('Training data shape: {}'.format(train.shape))
print('Test data shape:     {}'.format(test.shape))

class_names = transformer.class_names
#
gbtree = xgboost.XGBClassifier(
    objective="binary:logistic",
    n_estimators=700,     # More trees for better learning
    learning_rate=0.1,   # Lower learning rate to generalize better
    max_depth=7,          # Slightly deeper trees
    subsample=0.9,        # Use 85% of data per tree
    colsample_bytree=1, # Use 80% of features per tree
    gamma=2,              # Avoid unnecessary splits
    reg_lambda=3,         # Add regularization
    reg_alpha=1           # Helps with feature selection
)
gbtree.fit(train, train_labels)
print('Test accuracy: {}'.format(
    sklearn.metrics.accuracy_score(test_labels, gbtree.predict(test))))
# Convert the transformed data back to Tabular instances
train_data = transformer.invert(train)
test_data = transformer.invert(test)
```

```
Training data shape: (7109, 23)
Test data shape:     (1778, 23)
Test accuracy: 0.6827896512935883
```

```
np.random.seed(12345)
# Train an Random Forest model
transformer = TabularTransform().fit(tabular_data)
x = transformer.transform(tabular_data)
train, test, train_labels, test_labels = sklearn.model_selection.train_test_split(
    x[:, :-1], x[:, -1], train_size=0.80, stratify=x[:, -1]
)
print('Training data shape: {}'.format(train.shape))
print('Test data shape:     {}'.format(test.shape))

class_names = transformer.class_names
# Train Random Forest on your training data
rf_model = RandomForestClassifier(
    n_estimators=1000,     # Increase trees for better learning
    max_depth=12,         # Limit tree depth to avoid overfitting
    min_samples_split=4,  # Prevent too many small splits
    min_samples_leaf=3,   # Avoid overfitting small leaf nodes
    max_features=.8,  # Select a subset of features per tree
    class_weight="balanced",  # Handle class imbalance
    random_state=42,
    n_jobs=-1  # Use all CPU cores for faster training
)
# The line below has been modified to use the transformed data 'train' instead of 'trair
rf_model.fit(train, train_labels)  # Use the transformed data for training
#
print('Test accuracy: {}'.format(
    sklearn.metrics.accuracy_score(test_labels, rf_model.predict(test))))

# Convert the transformed data back to Tabular instances
train_data = transformer.invert(train)
test_data = transformer.invert(test)
```

```
Training data shape: (7109, 23)
Test data shape:     (1778, 23)
Test accuracy: 0.6912260967379078
```

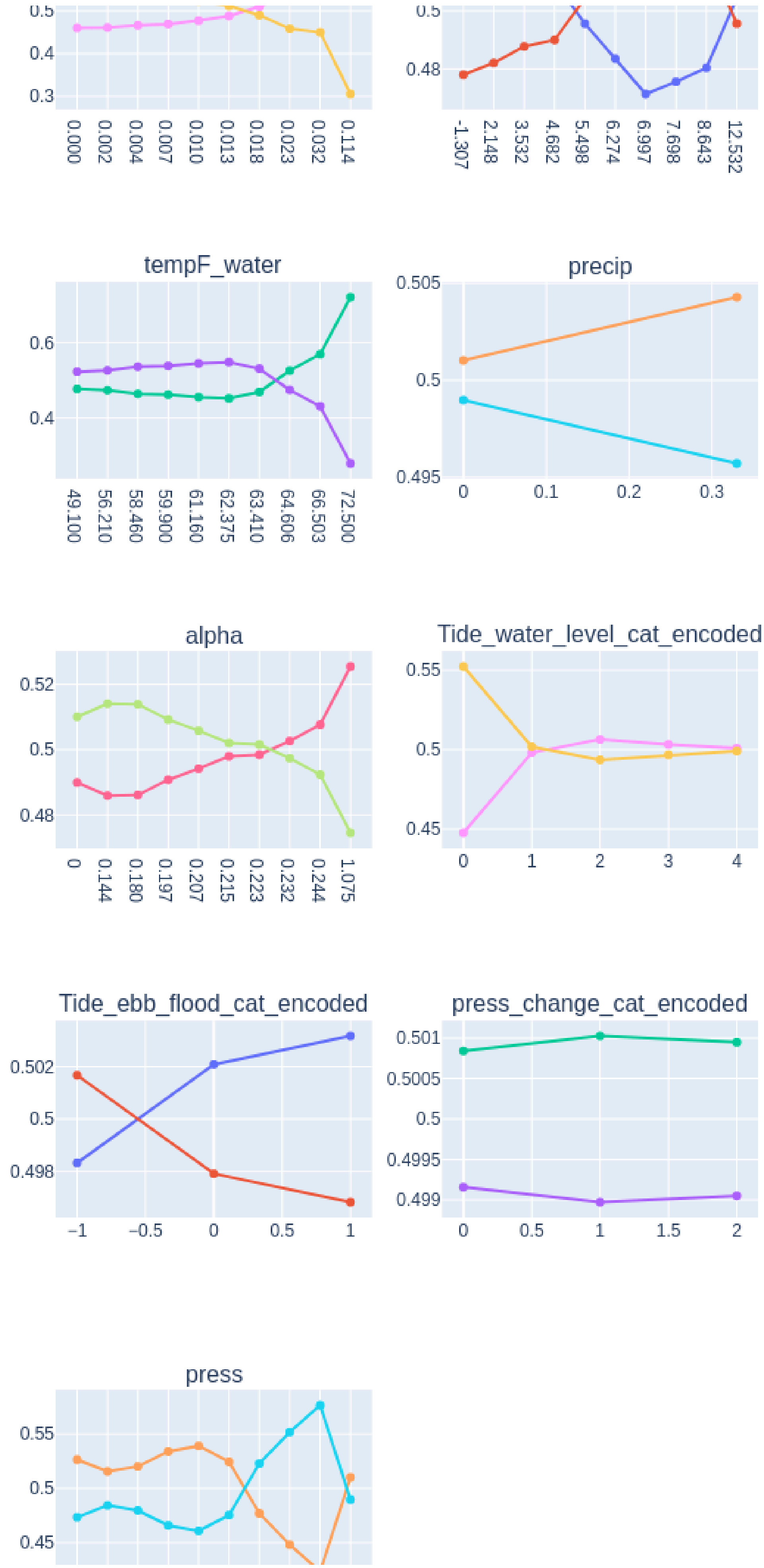
We then create an `TabularExplainer` explainer to generate local and global explanations.

```
from omnixai.explainers.tabular import TabularExplainer

# Initialize a TabularExplainer
explainers = TabularExplainer(
    explainers=["lime", "shap", "mace", "pdp", "ale"],
    mode="classification",
    data=train_data,
    model=rf_model,
    preprocess=lambda z: transformer.transform(z),
    params={
        "lime": {"kernel_width": 3},
        "shap": {"nsamples": 100},
        "mace": {"ignored_features": ["tempF_Diff_encoded"]}
    }
)
```

Interpretation of first chart: for Dusk, the model strongly favors Class = 1 (Red Line for Class 1) rather than Class 0 (Blue Line for Class 0)

You can plot just Class 1 line to more easily interpret the charts.

```
# Generate global explanations
global_explanations = explainers.explain_global()
global_explanations["pdp"].ipython_plot(class_names=class_names)
#The separation between the two lines shows that the model sees a clear difference
```

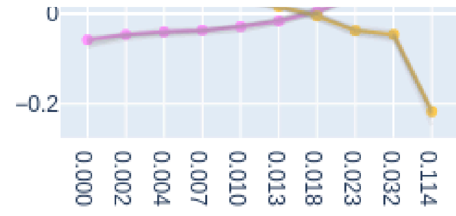

0.000 0.002 0.004 0.007 0.010 0.013 0.018 0.023 0.032 0.114
-1.307 2.148 3.532 4.682 5.498 6.274 6.997 7.698 8.643 12.532
tempF_water
49.100 56.210 58.460 59.900 61.160 62.375 63.410 64.606 66.503 72.500
precip
alpha
0 0.144 0.180 0.197 0.207 0.215 0.223 0.232 0.244 1.075
Tide_water_level_cat_encoded
Tide_ebb_flood_cat_encoded
press_change_cat_encoded
press

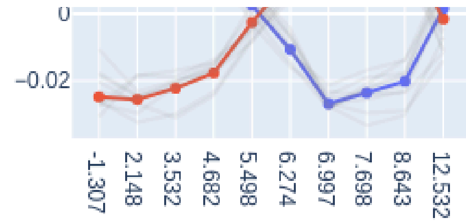
PDP compared to ALE

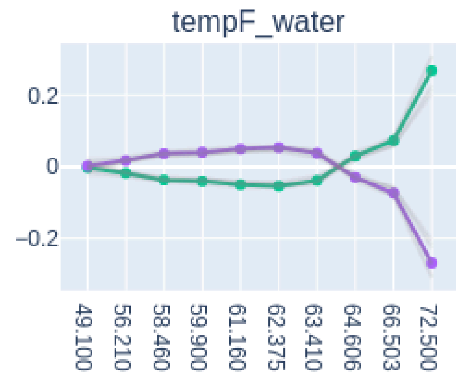
Method	
PDP Example: How Does "Light" (Dawn, Day, Dusk, Night) Affect Predictions?	PDP shows how changing "Light" affects the probabil If "Night" has a **higher probability of Class 1**, it sugge:

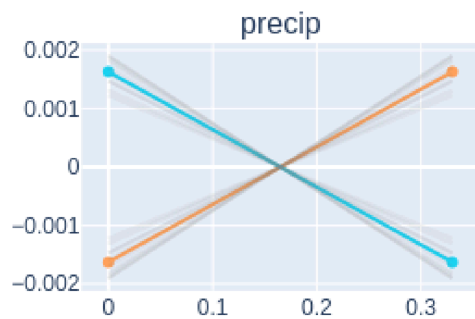
Method	
ALE Example: How Does "Light" Affect Predictions?	**Problem:** If "Light" is correlated with "Moon Phase" (e ALE looks at **small changes** in "Light" and their direct If "Light" is correlated with "Moon Phase", ALE adjusts Instead of showing a global trend, ALE focuses on **ho**

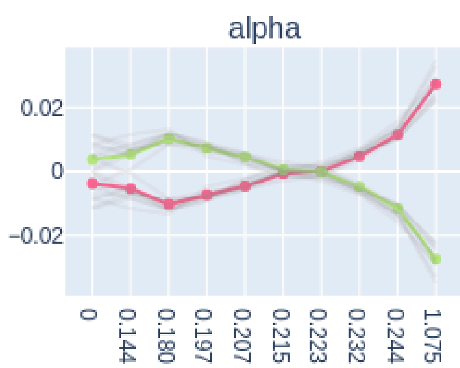
```
1 global_explanations["ale"].ipython_plot(class_names=class_names)
```

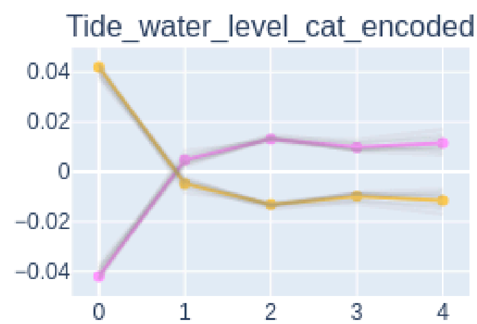

0
−0.2
0.000
0.002
0.004
0.007
0.010
0.013
0.018
0.023
0.032
0.114

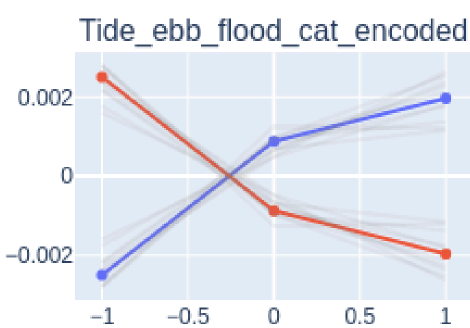

0
−0.02
-1.307
2.148
3.532
4.682
5.498
6.274
6.997
7.698
8.643
12.532

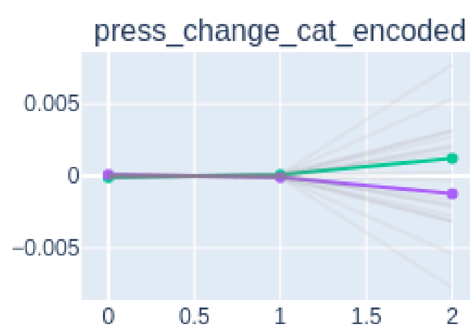

tempF_water
0.2
0
−0.2
49.100
56.210
58.460
59.900
61.160
62.375
63.410
64.606
66.503
72.500

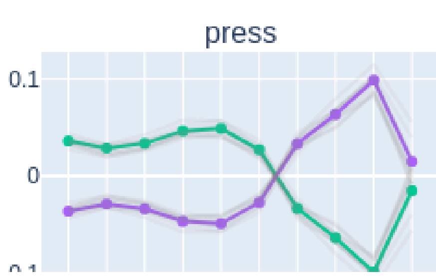

precip
0.002
0.001
0
−0.001
−0.002
0
0.1
0.2
0.3


alpha
0.02
0
−0.02
0
0.144
0.180
0.197
0.207
0.215
0.223
0.232
0.244
1.075


Tide_water_level_cat_encoded
0.04
0.02
0
−0.02
−0.04
0
1
2
3
4


Tide_ebb_flood_cat_encoded
0.002
0
−0.002
−1
−0.5
0
0.5
1


press_change_cat_encoded
0.005
0
−0.005
0
0.5
1
1.5
2


press
0.1
0

For some specific test instances, we can generate local explanations to analyze the predictions.

```
num_records = len(test_data)
print("Number of records in test_data:", num_records)
```