

```

1 # -*- coding: utf-8 -*-
2 """
3 Predictive analysis of naval incidents in the USA, 2002 - 2015:
4 Functions for model performance comparison
5
6 @author: "Oscar Anton"
7 @date: "2024"
8 @license: "CC BY-NC-ND 4.0 DEED"
9 @version: "0.9"
10 """
11
12 """
13 Example:
14 import sys
15 sys.path.append('myCustomFunctions.py')
16 import myCustomFunctions as own
17
18 own.model_metrics(nb_MA_train, X_test, y_test, styled=True)
19 """
20
21 # %% LIBRARIES
22
23 # System environment
24 import os
25
26 # Data general management
27 import numpy as np
28 import pandas as pd
29
30 # Visualization
31 import seaborn as sns
32 import matplotlib.pyplot as plt
33
34 # Model management
35 from sklearn.model_selection import learning_curve
36 from sklearn.preprocessing import LabelEncoder, label_binarize
37
38 # Model metrics
39 from sklearn.metrics import (accuracy_score, mean_squared_error, r2_score, f1_score,
40                               mean_absolute_error, roc_auc_score, roc_curve, auc,
41                               cohen_kappa_score, confusion_matrix, recall_score,
42                               precision_score)
43
44 # %% GENERAL VARIABLES
45 # Available CPU cores for multiprocessing (training models)
46 n_jobs = os.cpu_count() - 1
47
48 # Label encoder
49 label_encoder = LabelEncoder()
50
51
52 # %% SKLEARN: PERFORMANCE FOR BINOMIAL CLASSIFICATION MODELS (5.1)
53
54 # Function: Table with main metrics data
55 def model_metrics(model, X, y):
56     # Predictions (absolute)
57     y_pred = model.predict(X)
58
59     # Calculate main metrics
60     roc_auc = round(roc_auc_score(y, y_pred), 4)
61     accuracy = round(accuracy_score(y, y_pred), 4)

```

```

62     kappa = round(cohen_kappa_score(y, y_pred), 4)
63     rmse = round(mean_squared_error(y, y_pred), 4)
64     mae = round(mean_absolute_error(y, y_pred), 4)
65     r2 = round(r2_score(y, y_pred), 4)
66     f1 = round(f1_score(y, y_pred), 4)
67
68     # Sensitivity And Specificity
69     tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()
70     sensitivity = round(tp / (tp + fn), 4)
71     specificity = round(tn / (tn + fp), 4)
72
73     # Build multiindex table
74     metrics_df = pd.DataFrame([['ROC AUC:', roc_auc], ['Accuracy:', accuracy], ['Kappa:',
kappa],
75                                ['RMSE:', rmse], ['MAE:', mae], ['R2:', r2], ['F1:', f1], [
' ', ' '],
76                                ['Sensitivity:', sensitivity], ['Specificity:', specificity
]],
77                                columns=pd.MultiIndex.from_product([[model.__class__.__name__
], ['Metric', 'Value']]))
78
79     return metrics_df.style.hide()
80
81
82 # Function: Table with Confusion Matrix data
83 def confusion_matrix_table(model, X, y):
84     # Predictions (absolute)
85     y_pred = model.predict(X)
86
87     # Confusion matrix
88     tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()
89
90     # Dataframe creation
91     df = pd.DataFrame([[tp, fn], [fp, tn]],
92                       index=pd.Index(['1', '0'], name='Actual Label:'),
93                       columns=pd.MultiIndex.from_product([[model.__class__.__name__], ['1',
'0']],
94                                                         names=['Model:', 'Predicted:']))
95
96     # Dataframe style
97     styled_df = df.style.set_table_styles([
98         {'selector': 'th.col_heading', 'props': 'text-align: center;'},
99         {'selector': 'td', 'props': 'text-align: center;'},
100     ], overwrite=False)
101
102     return styled_df
103
104
105 # Function: Plot ROC Curve
106 def plot_roc_curve(model, X, y):
107     # Predicted probabilities
108     y_score = model.predict_proba(X)
109
110     # Calculate ROC for each class
111     fpr, tpr, _ = roc_curve(y, y_score[:, 1])
112
113     # Calculate AUC (Area Under Curve)
114     roc_auc = auc(fpr, tpr)
115
116     # Plot ROC Curve
117     plt.figure()
118     plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)

```

```

119 plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='dotted')
120 plt.xlim([0, 1])
121 plt.ylim([0, 1.01])
122 plt.xlabel('False Positive Rate')
123 plt.ylabel('True Positive Rate')
124 plt.title(f'ROC Curve for {model.__class__.__name__}')
125 plt.legend(loc="lower right")
126 plt.show()
127
128
129 # Function to plot learning curves
130 def plot_learning_curve(model, X, y, cv, train_sizes=np.linspace(.1, 1.0, 5)):
131     plt.figure()
132     plt.title(f"Learning Curve of {model}")
133     plt.xlabel("Training examples")
134     plt.ylabel("Score")
135
136     train_sizes, train_scores, test_scores = learning_curve(
137         model, X, y, cv=cv, train_sizes=train_sizes, n_jobs=n_jobs)
138     train_scores_mean = np.mean(train_scores, axis=1)
139     train_scores_std = np.std(train_scores, axis=1)
140     test_scores_mean = np.mean(test_scores, axis=1)
141     test_scores_std = np.std(test_scores, axis=1)
142
143     plt.grid()
144
145     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
146                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
147     plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
148                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
149     plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
150             label="Training score")
151     plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
152             label="Cross-validation score")
153
154     return plt
155
156
157 # %% -----
158
159
160 # %% SKLEARN: PERFORMANCE FOR MULTILEVEL CLASSIFICATION MODELS (5.2)
161
162 # Function: Table with main metrics data
163 def ma_model_metrics(model, X, y, styled=False):
164     # Predictions (absolute)
165     y_pred = model.predict(X)
166
167     # Data binarize for auc calculation
168     y_bin = label_binarize(y, classes=np.unique(y))
169     y_pred_bin = label_binarize(y_pred, classes=np.unique(y))
170
171     # Calculate main metrics
172     roc_auc = round(roc_auc_score(y_bin, y_pred_bin), 4)
173     accuracy = round(accuracy_score(y, y_pred), 4)
174     kappa = round(cohen_kappa_score(y, y_pred), 4)
175     rmse = round(mean_squared_error(y, y_pred), 4)
176     mae = round(mean_absolute_error(y, y_pred), 4)
177     r2 = round(r2_score(y, y_pred), 4)
178     f1 = round(f1_score(y, y_pred, average='macro'), 4)
179

```

```

180 # Build multiindex table
181 df = pd.DataFrame([['ROC AUC:', roc_auc], ['Accuracy:', accuracy], ['Kappa:', kappa],
182                    ['RMSE:', rmse], ['MAE:', mae], ['R2:', r2], ['F1:', f1]],
183                    columns=('metric', 'value'))
184
185 if styled:
186     title = f'{model.__class__.__name__} Training'
187     df.columns = pd.MultiIndex.from_tuples([(title, col) for col in df.columns])
188     return df.style.hide()
189 else:
190     return df
191
192
193 # Function: Table for recall & precision, sensitivity & specificity
194 def sens_spec(model, X, y, styled=False):
195     # Predictions (absolute)
196     y_pred = model.predict(X)
197
198     # Recall & Precision values
199     recall = round(recall_score(y, y_pred, average='macro'), 4)
200     precision = round(precision_score(y, y_pred, average='macro'), 4)
201
202     # Confusion matrix
203     conf_matrix = confusion_matrix(y, y_pred)
204
205     # List compression for sens & spec values calculation
206     sensitivity, specificity = zip(*[(round(recall_score(y, y_pred, labels=[i], average='
macro'), 4),
207                                     round(conf_matrix[i, i] / sum(conf_matrix[:, i]), 4))
208                                     for i in range(len(conf_matrix))])
209
210     # Labels and indexes
211     column_labels = label_encoder.inverse_transform(model.classes_)
212     index_1 = ['Recall:', 'Precision:']
213     index_2 = [recall, precision]
214     index_ = [' - ', ' - ']
215     index_3 = ['Sensitivity:', 'Specificity:']
216
217     # Build multiindex table
218     df = pd.DataFrame([sensitivity, specificity])
219     df.columns = column_labels
220     df.index = [index_1, index_2, index_, index_3]
221
222     # Dataframe style
223     if styled:
224         title = f'{model.__class__.__name__} Model'
225         df.columns = pd.MultiIndex.from_tuples([(title, col) for col in df.columns])
226         return df.style.set_table_styles([{'selector': 'th.col_heading',
227                                           'props': 'text-align: center;'}], overwrite=
False)
228     else:
229         return df
230
231
232 # Function: Table with Confusion Matrix
233 def ma_confusion_matrix_table(model, X, y):
234     # Predictions (absolute)
235     y_pred = model.predict(X)
236
237     # Get labels decoding target variable
238     labels = label_encoder.inverse_transform(model.classes_)
239

```

```

240     # Build table
241     df = pd.DataFrame(confusion_matrix(y, y_pred),
242                       columns=pd.MultiIndex.from_product([[f'{model.__class__.__name__}':
Confusion Matrix'], labels]),
243                       index=labels)
244
245     # Dataframe style
246     styled_df = df.style.set_table_styles([
247         {'selector': 'th.col_heading', 'props': 'text-align: center;'},
248         {'selector': 'td', 'props': 'text-align: center;'},
249     ], overwrite=False)
250
251     return styled_df
252
253
254 # Function: Plot Multiclass ROC Curve
255 def roc_curve_plot(model, X, y):
256     # Predictions (absolute)
257     y_pred = model.predict(X)
258
259     # Data binarize for auc calculation
260     y_bin = label_binarize(y, classes=model.classes_)
261     y_pred_bin = label_binarize(y_pred, classes=model.classes_)
262
263     # Compute ROC curve and ROC area for each class
264     fpr = dict()
265     tpr = dict()
266     roc_auc = dict()
267     for i in model.classes_:
268         fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_pred_bin[:, i])
269         roc_auc[i] = auc(fpr[i], tpr[i])
270
271     # Compute micro-average ROC curve and ROC area
272     fpr["micro"], tpr["micro"], _ = roc_curve(y_bin.ravel(), y_pred_bin.ravel())
273     roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
274
275     # Plot ROC curve for each class
276     plt.figure()
277     colors = sns.color_palette("hls", 5)
278     for i, color in zip(model.classes_, colors):
279         plt.plot(fpr[i], tpr[i], color=color, lw=2,
280                 label=f'ROC of class {i} (AUC = {1:0.2f})'.format(i, roc_auc[i]))
281
282     # Plot micro-average ROC curve
283     plt.plot(fpr["micro"], tpr["micro"], color='grey', lw=1, linestyle='dashed',
284             label=f'micro-average ROC (AUC = {0:0.2f})'.format(roc_auc["micro"]))
285
286     plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='dotted')
287     plt.xlim([0.0, 1.0])
288     plt.ylim([0.0, 1.01])
289     plt.xlabel('False Positive Rate')
290     plt.ylabel('True Positive Rate')
291     plt.title(f'ROC (Multiclass) for {model.__class__.__name__}')
292     plt.legend(loc="lower right", fontsize="8")
293     plt.show()
294
295
296 # Function: Feature importances
297 def sklearn_feature_importances(model, plot=False):
298     importances = pd.DataFrame({'variable_name': model.feature_names_in_,
299                                'value': model.feature_importances_}).sort_values(by='value
', ascending=False)

```

```

300 # Plot horizontal bars if enabled in the call, otherwise return values
301 if plot:
302     plt.figure(figsize=(10, 7))
303     plt.barh(importances['variable_name'], importances['value'], color='#00bfc4')
304     plt.title(f"Feature importances of {model.__class__.__name__} model")
305     plt.xlabel('Relative Feature importance')
306     plt.gca().invert_yaxis()
307     plt.show()
308 else:
309     return importances
310
311
312 # %% KERAS: PERFORMANCE FOR MULTILEVEL CLASSIFICATION MODELS (5.2)
313
314 # Function: Plot loss / accuracy train evolution
315 def keras_train_plot(data):
316     # Train process visualization
317     df_train = pd.DataFrame(data)
318     # df_train['epochs']=history.epoch
319     df_train['epochs'] = list(range(0, len(data['accuracy'])))
320
321     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6))
322
323     fig.suptitle('Train process', fontsize=12)
324
325     ax1.plot(df_train['epochs'], df_train['accuracy'], label='train_accuracy')
326     ax1.plot(df_train['epochs'], df_train['val_accuracy'], label='val_accuracy')
327
328     ax2.plot(df_train['epochs'], df_train['loss'], label='train_loss')
329     ax2.plot(df_train['epochs'], df_train['val_loss'], label='val_loss')
330
331     ax1.legend(loc='best')
332     ax2.legend(loc='best')
333     plt.show()
334
335
336 # Function: Table with main metrics data
337 def keras_model_metrics(model, X, y_ohe, styled=False):
338     y_pred_ohe = pd.DataFrame(model.predict(X))
339     y_predserie = y_pred_ohe.idxmax(axis=1)
340     yserie = pd.Series(label_encoder.fit_transform(y_ohe.idxmax(axis=1)))
341
342     roc_auc = round(roc_auc_score(y_ohe, y_pred_ohe), 4)
343     accuracy = round(accuracy_score(yserie, y_predserie), 4)
344     kappa = round(cohen_kappa_score(yserie, y_predserie), 4)
345     rmse = round(mean_squared_error(y_ohe, y_pred_ohe), 4)
346     mae = round(mean_absolute_error(y_ohe, y_pred_ohe), 4)
347     r2 = round(r2_score(y_ohe, y_pred_ohe), 4)
348     f1 = round(f1_score(yserie, y_predserie, average='macro'), 4)
349
350     # Build multiindex table
351     df = pd.DataFrame([['ROC AUC:', roc_auc], ['Accuracy:', accuracy], ['Kappa:', kappa],
352                       ['RMSE:', rmse], ['MAE:', mae], ['R2:', r2], ['F1:', f1]],
353                      columns=('metric', 'value'))
354
355     if styled:
356         title = f'{model.__class__.__name__} Training'
357         df.columns = pd.MultiIndex.from_tuples([(title, col) for col in df.columns])
358         return df.style.hide()
359     else:
360         return df
361

```

```

362
363 # Function: Table for recall & precision, sensitivity & specificity
364 def keras_sens_spec(model, X, y, styled=False):
365     # Predictions (absolute)
366     y_pred_ohe = pd.DataFrame(model.predict(X))
367     y_predserie = y_pred_ohe.idxmax(axis=1)
368     yserie = pd.Series(label_encoder.fit_transform(y.idxmax(axis=1)))
369
370     # Recall & Precision values
371     recall = round(recall_score(yserie, y_predserie, average='macro'), 4)
372     precision = round(precision_score(yserie, y_predserie, average='macro'), 4)
373
374     # Confusion matrix
375     conf_matrix = confusion_matrix(yserie, y_predserie)
376
377     # List compression for sens & spec values calculation
378     sensitivity, specificity = zip(*[(round(recall_score(yserie, y_predserie, labels=[i
379 ], average='macro'), 4),
380                                     round(conf_matrix[i, i] / sum(conf_matrix[:, i]), 4))
381                                     for i in range(len(conf_matrix))])
382
383     # Labels and indexes
384     column_labels = y.columns
385     index_1 = ['Recall:', 'Precision:']
386     index_2 = [recall, precision]
387     index_ = [' - ', ' - ']
388     index_3 = ['Sensitivity:', 'Specificity:']
389
390     # Build multiindex table
391     df = pd.DataFrame([sensitivity, specificity])
392     df.columns = column_labels
393     df.index = [index_1, index_2, index_, index_3]
394
395     # Dataframe style
396     if styled:
397         title = f'{model.__class__.__name__} Model'
398         df.columns = pd.MultiIndex.from_tuples([(title, col) for col in df.columns])
399         return df.style.set_table_styles([{'selector': 'th.col_heading',
400                                           'props': 'text-align: center;'}], overwrite=
False)
401     else:
402         return df
403
404 # Function: Table with Confusion Matrix
405 def keras_confusion_matrix_table(model, X, y):
406     # Predictions (max)
407     y_pred_max = np.argmax(model.predict(X), axis=1)
408     y_max = np.argmax(y, axis=1)
409
410     # Build table
411     df = pd.DataFrame(confusion_matrix(y_max, y_pred_max),
412                       columns=pd.MultiIndex.from_product([f'{model.name}: Confusion
413 Matrix'], y.columns)),
414                       index=y.columns)
415
416     # Dataframe style
417     styled_df = df.style.set_table_styles([
418         {'selector': 'th.col_heading', 'props': 'text-align: center;'},
419         {'selector': 'td', 'props': 'text-align: center;'},
420     ], overwrite=False)

```

```

421     return styled_df
422
423
424 # Function: Plot Multiclass ROC Curve
425 def keras_roc_curve_plot(model, X, y):
426     # Predictions (max)
427     y_pred = pd.DataFrame(model.predict(X))
428     y_pred.columns = label_encoder.inverse_transform(y_pred.columns)
429
430     # Compute ROC curve and ROC area for each class
431     fpr, tpr, roc_auc = {}, {}, {}
432     for i in y.columns:
433         fpr[i], tpr[i], _ = roc_curve(y[i], y_pred[i])
434         roc_auc[i] = auc(fpr[i], tpr[i])
435
436     # Compute micro-average ROC curve and ROC area
437     fpr["micro"], tpr["micro"], _ = roc_curve(y.values.ravel(), y_pred.values.ravel())
438     roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
439
440     # Plot ROC curve for each class
441     plt.figure()
442     colors = sns.color_palette("hls", 5)
443     for i, color in zip(y.columns, colors):
444         plt.plot(fpr[i], tpr[i], color=color, lw=2,
445                 label='ROC of class {0} (AUC = {1:0.2f})'.format(i, roc_auc[i]))
446
447     # Plot micro-average ROC curve
448     plt.plot(fpr["micro"], tpr["micro"], color='grey', lw=1, linestyle='dashed',
449             label='micro-average ROC (AUC = {0:0.2f})'.format(roc_auc["micro"]))
450
451     plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='dotted')
452     plt.xlim([0.0, 1.0])
453     plt.ylim([0.0, 1.01])
454     plt.xlabel('False Positive Rate')
455     plt.ylabel('True Positive Rate')
456     plt.title(f'ROC (Multiclass) for {model.name}')
457     plt.legend(loc="lower right", fontsize="8")
458     plt.show()
459
460
461 # Function: Plot feature importances
462 def keras_sec_importances(model, X, plot=False):
463     # Calculate input layer weights
464     weights = model.layers[0].get_weights()[0]
465
466     # Create dataframe for weights and variable names
467     importances = pd.DataFrame({'variable_name': X.columns,
468                                'value': np.mean(np.abs(weights), axis=1)}).sort_values(by=
469 'value', ascending=False)
470
471     # Plot horizontal bars if enabled in the call, otherwise return values
472     if plot:
473         plt.figure(figsize=(10, 7))
474         plt.barh(importances['variable_name'], importances['value'], color='#00bfc4')
475         plt.title(f"Feature importances in Keras {model.__class__.__name__} model")
476         plt.xlabel('Relative Feature importance (based on first layer weights)')
477         plt.gca().invert_yaxis()
478         plt.show()
479     else:
480         return importances
481

```



```

482 # Function: Plot feature importances
483 def keras_func_importances(model, plot = False):
484     weight_data = []
485     # Considering all entrance dense layers have a name like corresponding variable name
486     # Iterate through dense layers with a particular name, obtaining their weights
487     for layer in model.layers:
488         if 'Dense' in layer.__class__.__name__ and 'dense_' not in layer.name:
489             layer_name = layer.name
490             weights = np.mean(np.abs(layer.get_weights()[0]), axis=1)
491             for i, weight in enumerate(weights):
492                 weight_data.append([f"{layer_name}_{i}", weight])
493
494     # Build dataframe
495     importances = pd.DataFrame(weight_data, columns=['variable_name', 'value']).sort_values
    (by='value', ascending=False)
496
497     # Plot horizontal bars if enabled in the call, otherwise return values
498     if plot:
499         plt.figure(figsize=(10, 7))
500         plt.barh(importances['variable_name'], importances['value'], color='#00bfc4')
501         plt.title(f"Feature importances in Keras {model.__class__.__name__} model")
502         plt.xlabel('Relative Feature importance (based on first layer weights)')
503         plt.gca().invert_yaxis()
504         plt.show()
505     else:
506         return importances
507
508
509
510 # %% H2O: PERFORMANCE FOR MULTILEVEL CLASSIFICATION MODELS (5.2)
511
512 # Function: Table with main metrics data
513 def h2o_model_metrics(h2o_model, h2o_test, styled=False):
514     h2o_predict = pd.Series(label_encoder.fit_transform(h2o_model.predict(h2o_test))['
predict'].as_data_frame()))
515     h2o_y = pd.Series(label_encoder.fit_transform(h2o_test['y'].as_data_frame()))
516
517     h2o_y_bin = label_binarize(h2o_y, classes=np.unique(h2o_y))
518     h2o_predict_bin = label_binarize(h2o_predict, classes=np.unique(h2o_y))
519
520     # Calculate main metrics
521     roc_auc = round(roc_auc_score(h2o_y_bin, h2o_predict_bin), 4)
522     accuracy = round(accuracy_score(h2o_y_bin, h2o_predict_bin), 4)
523     kappa = round(cohen_kappa_score(h2o_y, h2o_predict), 4)
524     rmse = round(mean_squared_error(h2o_y_bin, h2o_predict_bin), 4)
525     mae = round(mean_absolute_error(h2o_y_bin, h2o_predict_bin), 4)
526     r2 = round(r2_score(h2o_y_bin, h2o_predict_bin), 4)
527     f1 = round(f1_score(h2o_y_bin, h2o_predict_bin, average='macro'), 4)
528
529     # Build multiindex table
530     df = pd.DataFrame([['ROC AUC:', roc_auc], ['Accuracy:', accuracy], ['Kappa:', kappa],
531                        ['RMSE:', rmse], ['MAE:', mae], ['R2:', r2], ['F1:', f1]],
532                       columns=('metric', 'value'))
533
534     if styled:
535         title = f'{h2o_model.key} Training'
536         df.columns = pd.MultiIndex.from_tuples([(title, col) for col in df.columns])
537         return df.style.hide()
538     else:
539         return df
540
541

```

```

542 # Function: Table for recall & precision, sensitivity & specificity
543 def h2o_sens_spec(h2o_model, h2o_test, styled=False):
544     # Predictions
545     h2o_pred = pd.Series(label_encoder.fit_transform(h2o_model.predict(h2o_test))['predict']
546                          ].as_data_frame())
547     h2o_y = pd.Series(label_encoder.fit_transform(h2o_test['y'].as_data_frame()))
548     # Recall & Precision values
549     recall = round(recall_score(h2o_y, h2o_pred, average='macro'), 4)
550     precision = round(precision_score(h2o_y, h2o_pred, average='macro'), 4)
551     # Confusion matrix
552     conf_matrix = confusion_matrix(h2o_y, h2o_pred)
553     # List compression for sens & spec values calculation
554     sensitivity, specificity = zip(*[(round(recall_score(h2o_y, h2o_pred, labels=[i],
555     average='macro'), 4),
556                                     round(conf_matrix[i, i] / sum(conf_matrix[:, i]), 4))
557                                     for i in range(len(conf_matrix))])
558     # Labels and indexes
559     column_labels = np.unique(h2o_test['y'].as_data_frame())
560     index_1 = ['Recall:', 'Precision:']
561     index_2 = [recall, precision]
562     index_ = [' - ', ' - ']
563     index_3 = ['Sensitivity:', 'Specificity:']
564     # Build multiindex table
565     df = pd.DataFrame([sensitivity, specificity])
566     df.columns = column_labels
567     df.index = [index_1, index_2, index_, index_3]
568     # Dataframe style
569     if styled:
570         title = f'{h2o_model.key} Model'
571         df.columns = pd.MultiIndex.from_tuples([(title, col) for col in df.columns])
572         return df.style.set_table_styles([{'selector': 'th.col_heading',
573         'props': 'text-align: center;'}], overwrite=
574 False)
575     else:
576         return df
577
578
579
580
581

```