

```
1 import DataPackage as dp
2 import DataExperimentSupport as des
3 import copy
4 import pickle
5
6
7 class DataExperiment:
8
9     def __init__(self,
10                  projectName,
11                  experimentName,
12                  origData,
13                  uniqueColumn,
14                  targetColumn,
15                  classifier,
16                  processDataPackage=False
17                  ):
18         self.projectName = projectName
19         self.experimentName = experimentName
20         self.__setDataPackage(origData=origData,
21                               uniqueColumn=uniqueColumn
22                               ,
23                               targetColumn=targetColumn
24                               )
25         self.__setClassifier(classifier)
26
27         # Should really consider putting these into a
28         function
29         # Following are default values on init for
30         stuff set later
31
32         self.isBaseModelLoaded = False
33         self.baseModel = None
34         self.isFinalModelLoaded = False
35         self.finalModel = None
36
37         self.isBaseModelPredicted = False
38         self.baseModelPrediction = None
39         self.baseModelAccuracy = None
```

```
35         self.baseModelPrecision = None
36         self.baseModelRecall = None
37         self.baseModelF1 = None
38         self.baseModelCohenKappa = None
39
40         self.isFinalModelPredicted = False
41         self.finalModelPrediction = None
42         self.finalModelPrediction = None
43         self.finalModelAccuracy = None
44         self.finalModelPrecision = None
45         self.finalModelRecall = None
46         self.finalModelF1 = None
47         self.finalModelCohenKappa = None
48
49         self.isBaseModelLearningCurveCreated = False
50         self.baseModel_train_sizes = None
51         self.baseModel_train_scores = None
52         self.baseModel_test_scores = None
53         self.baseModel_fit_times = None
54         self.baseModelROCAUC = None
55
56         self.isFinalModelLearningCurveCreated = False
57         self.finalModel_train_sizes = None
58         self.finalModel_train_scores = None
59         self.finalModel_test_scores = None
60         self.finalModel_fit_times = None
61
62         self.isBaseModelROCAUCCalculated = False
63         self.baseModelROCAUC = None
64         self.isFinalModelROCAUCCalculated = False
65         self.finalModelROCAUC = None
66
67         self.finalFeaturesAll = None
68         self.finalFeatures = None
69
70         # =====
71         if processDataPackage:
```

```
72         self.processDataPackage()
73
74         self.display()
75
76     def display(self):
77         indent = '---> '
78         print(f'DataExperiment summary:')
79         print(f'{indent}projectName: {self.projectName
80 }')
81         print(f'{indent}experimentName: {self.
experimentName}')
82         print(f'{indent}isDataPackageLoaded: {self.
isDataPackageLoaded}')
83         print(f'{indent}isBaseModelLoaded: {self.
isBaseModelLoaded}')
84         print(f'{indent}isBaseModelPredicted: {self.
isBaseModelPredicted}')
85         print(f'{indent}
isBaseModelLearningCurveCreated: {self.
isBaseModelLearningCurveCreated}')
86
87         print(f'{indent}isFinalModelLoaded: {self.
isFinalModelLoaded}')
88         print(f'{indent}isFinalModelPredicted: {self.
isFinalModelPredicted}')
89         print(f'{indent}
isFinalModelLearningCurveCreated: {self.
isFinalModelLearningCurveCreated}')
90
91         print(f'{indent}isClassifierLoaded: {self.
isClassifierLoaded}')
92         print(self.getClassifier())
93         print('')
94         self.dataPackage.display()
95
96     def getClassifier(self):
97         return copy.deepcopy(self.classifier)
```

```
98
99     def __setClassifier(self, classifier):
100         self.classifier = classifier
101         self.isClassifierLoaded = True
102
103     def __setDataPackage(self,
104                           origData,
105                           uniqueColumn,
106                           targetColumn):
107
108         self.dataPackage = dp.DataPackage(origData=
109 origData,
110                                           uniqueColumn
111 =uniqueColumn,
112                                           targetColumn
113 =targetColumn)
114         self.isDataPackageLoaded = True
115
116     def processDataPackage(self, verbose=False):
117         self.dataPackage.classBalanceUndersample()
118         self.dataPackage.splitTrainTest()
119
120         if verbose:
121             self.dataPackage.display()
122
123     def createBaseModel(self):
124         model = des.createModel(data=self.dataPackage.
125 getTrainData(),
126                               uniqueColumn=self.
127 dataPackage.uniqueColumn,
128                               targetColumn=self.
129 dataPackage.targetColumn,
130                               classifier=self.
131 getClassifier())
132
133         self.__setBaseModel(model)
134
135     def createFinalModel(self,
```

```

129                                     featureImportanceThreshold=0.
    002):
130         # TO DO:
131         # need to get list of features we wish to use
    to subset full data
132
133         impFeatureListFull = self.
    __getFinalModelFeatures(returnAbove=
    featureImportanceThreshold,
134
    includeUniqueAndTarget=True)
135
136         impFeatureList = self.__getFinalModelFeatures(
    returnAbove=featureImportanceThreshold,
137
    includeUniqueAndTarget=False)
138
139         # get full training dataframe
140         df = self.dataPackage.getTrainData()
141
142         model = des.createModel(data=df[
    impFeatureListFull],
143
    uniqueColumn=self.
    dataPackage.uniqueColumn,
144
    targetColumn=self.
    dataPackage.targetColumn,
145
    classifier=self.
    getClassifier())
146         self.__setFinalModel(model=model,
147
    finalFeatures=
    impFeatureList,
148
    finalFeaturesAll=
    impFeatureListFull)
149
150     def __setBaseModel(self, model):
151         self.baseModel = model
152         self.isBaseModelLoaded = True
153

```

```

154         # when you set base model invalidate some
155         things
156         self.isBaseModelPredicted = False
157         self.baseModelPrediction = None
158
159     def __setFinalModel(self,
160                        model,
161                        finalFeatures,
162                        finalFeaturesAll):
163         self.finalModel = model
164         self.isFinalModelLoaded = True
165         self.finalFeaturesAll = finalFeaturesAll
166         self.finalFeatures = finalFeatures
167
168         # when you set base model invalidate some
169         things
170         self.isFinalModelPredicted = False
171         self.finalModelPrediction = None
172
173     def getBaseModel(self):
174         return self.baseModel
175
176     def getFinalModel(self):
177         return self.finalModel
178
179     def __setBaseModelPrediction(self,
180                                predictionData,
181                                colActual,
182                                colPredict,
183                                average='weighted',
184                                sigDigs=2):
185         self.baseModelPrediction = predictionData
186         self.isBaseModelPredicted = True
187         self.baseModelPredictionColActual = colActual
188         self.baseModelPredictionColPredict =
189         colPredict
190
191         self.baseModelAccuracy = round(des.

```

```
188 getModelAccuracy(data=predictionData,
189
        colActual=colActual,
190
        colPredict=colPredict), sigDigs)
191
192     self.baseModelPrecision = round(des.
    getModelPrecision(data=predictionData,
193
        colActual=colActual,
194
        colPredict=colPredict,
195
        average=average), sigDigs)
196
197     self.baseModelRecall = round(des.
    getModelRecall(data=predictionData,
198
        colActual=colActual,
199
        colPredict=colPredict,
200
        average=average), sigDigs)
201
202     self.baseModelF1 = round(des.getModelF1(data=
    predictionData,
203
        colActual=colActual,
204
        colPredict=colPredict,
205
        average=average), sigDigs)
206
207     self.baseModelCohenKappa = round(des.
    getModelCohenKappa(data=predictionData,
208
        colActual=colActual,
209
```

```
209         colPredict=colPredict), sigDigs)
210
211     def __setFinalModelPrediction(self,
212                                   predictionData,
213                                   colActual,
214                                   colPredict,
215                                   average='weighted',
216                                   sigDigs=2):
217         self.finalModelPrediction = predictionData
218         self.isFinalModelPredicted = True
219         self.finalModelPredictionColActual = colActual
220         self.finalModelPredictionColPredict =
221         colPredict
222         self.finalModelAccuracy = round(des.
223         getModelAccuracy(data=predictionData,
224
225         colActual=colActual,
226
227         colPredict=colPredict), sigDigs)
228
229         self.finalModelPrecision = round(des.
230         getModelPrecision(data=predictionData,
231
232         colActual=colActual,
233
234         colPredict=colPredict,
235
236         average=average), sigDigs)
```



```

235
236         self.finalModelF1 = round(des.getModelF1(data=
predictionData,
237         colActual=colActual,
238         colPredict=colPredict,
239         average=average), sigDigs)
240
241         self.finalModelCohenKappa = round(des.
getModelCohenKappa(data=predictionData,
242         colActual=colActual,
243         colPredict=colPredict), sigDigs)
244
245     def showBaseModelStats(self):
246         print(f'Base Model Stats:')
247         print(f'Accuracy: {self.baseModelAccuracy}')
248         print(f'Precision: {self.baseModelPrecision}')
249         print(f'Recall: {self.baseModelRecall}')
250         print(f'F1 Score: {self.baseModelF1}')
251         print(f'Cohen kappa: {self.
baseModelCohenKappa}')
252
253     def showFinalModelStats(self):
254         print(f'Final Model Stats:')
255         print(f'Accuracy: {self.finalModelAccuracy}')
256         print(f'Precision: {self.finalModelPrecision}')
257         print(f'Recall: {self.finalModelRecall}')
258         print(f'F1 Score: {self.finalModelF1}')
259         print(f'Cohen kappa: {self.
finalModelCohenKappa}')
260
261     def getBaseModelPrediction(self):
262         if self.isBaseModelPredicted:

```

```
263         return self.baseModelPrediction
264     else:
265         print(f'No base model predictions
calculated.')
```

```
266         return None
267
268     def getFinalModelPrediction(self):
269         if self.isFinalModelPredicted:
270             return self.finalModelPrediction
271         else:
272             print(f'No final model predictions
calculated.')
```

```
273         return None
274
275     def predictBaseModel(self, average='weighted'):
276         if self.isBaseModelPredicted:
277             display("Base model already predicted.
Displaying results:")
278             self.showBaseModelStats()
279             return
280
281         tDf, colActual, colPredict = des.predictModel(
model=self.getBaseModel(),
282
data=self.dataPackage.getTestData(),
283
uniqueColumn=self.dataPackage.uniqueColumn,
284
targetColumn=self.dataPackage.targetColumn)
285
286         self.__setBaseModelPrediction(predictionData=
tDf,
287
colActual=
colActual,
288
colPredict=
colPredict,
289
average=average)
290
```

```

291         self.showBaseModelStats()
292
293     def predictFinalModel(self, average='weighted'):
294         if self.isFinalModelPredicted:
295             display("Final model already predicted.
296             Displaying results:")
297             self.showFinalModelStats()
298             return
299
300         testData = self.dataPackage.getTestData()
301         testData = testData[self.finalFeaturesAll].
302         copy()
303         tDf, colActual, colPredict = des.predictModel(
304             model=self.getFinalModel(),
305             data=testData,
306             uniqueColumn=self.dataPackage.uniqueColumn,
307             targetColumn=self.dataPackage.targetColumn)
308
309         self.__setFinalModelPrediction(predictionData=
310             tDf,
311             colActual=
312             colActual,
313             colPredict=
314             colPredict,
315             average=average
316         )
317
318         self.showFinalModelStats()
319
320     def analyzeBaseModelFeatureImportance(self,
321         returnAbove=
322         0.002,
323         startValue=0
324         .0001,
325         increment=0.

```

```

316 0001,
317                                     upperValue=0
    .01,
318                                     showSummary=
    True,
319                                     showPlot=
    True):
320
321         df, featureLabel, valueLabel = des.
getModelFeatureImportance(self.getBaseModel())
322
323         retDf = des.analyzeModelFeatureImportance(data
=df,
324
    valueLabel=valueLabel,
325
    startValue=startValue,
326
    increment=increment,
327
    upperValue=upperValue,
328
    returnAbove=returnAbove,
329
    showSummary=showSummary,
330
    showPlot=showPlot)
331         return retDf
332
333     def analyzeFinalModelFeatureImportance(self,
334                                     returnAbove
=0.002,
335
    startValue=
0.0001,
336
    increment=0
.0001,
337
    upperValue=
0.01):

```

```
338
339         df, featureLabel, valueLabel = des.
getModelFeatureImportance(self.getFinalModel())
340
341         retDf = des.analyzeModelFeatureImportance(data
=df,
342
valueLabel=valueLabel,
343
startValue=startValue,
344
increment=increment,
345
upperValue=upperValue,
346
returnAbove=returnAbove,
347
showSummary=True)
348         return retDf
349
350     def showBaseModelFeatureImportance(self,
351                                         startValue=0.
0001,
352                                         increment=0.
0001,
353                                         upperValue=0.01
):
354
355         df, featureLabel, valueLabel = des.
getModelFeatureImportance(self.getBaseModel())
356
357         des.analyzeModelFeatureImportance(data=df,
358                                         startValue=
startValue,
359                                         increment=
increment,
360                                         upperValue=
upperValue,
```

```
361 showSummary=
    False)
362
363     des.showAllModelFeatureImportance(data=df,
364                                         featureLabel
365                                         =featureLabel,
366                                         valueLabel=
367                                         valueLabel
368                                         )
369
370     def showFinalModelFeatureImportance(self,
371                                         startValue=0.
372                                         0001,
373                                         increment=0.
374                                         0001,
375                                         upperValue=0.
376                                         01):
377
378         df, featureLabel, valueLabel = des.
379         getModelFeatureImportance(self.getFinalModel())
380
381         des.analyzeModelFeatureImportance(data=df,
382                                             startValue=
383                                             startValue,
384                                             increment=
385                                             increment,
386                                             upperValue=
387                                             upperValue,
388                                             showSummary=
389                                             False)
390
391         des.showAllModelFeatureImportance(data=df,
392                                             featureLabel
393                                             =featureLabel,
394                                             valueLabel=
395                                             valueLabel
396                                             )
397
```

```

386     def showBaseModelReport(self,
387                             axisLabels
388                             ):
389         self.showBaseModelStats()
390
391         des.showReport(data=self.
getBaseModelPrediction(),
392                       colNameActual=self.
baseModelPredictionColActual,
393                       colNamePredict=self.
baseModelPredictionColPredict,
394                       axisLabels=axisLabels,
395                       titleSuffix=self.experimentName
        )
396
397         self.showBaseModelLearningCurve()
398         self.showBaseModelROCAUC(axisLabels=axisLabels
        )
399
400     def showBaseModelROCAUC(self, axisLabels,
useStored=False):
401         if useStored and self.
isBaseModelROCAUCCalculated:
402             print('Base model ROCAUC already
calculated. Displaying stored results')
403             tViz = self.__getBaseModelROCAUC()
404             tViz.show()
405         else:
406             print('Base model ROCAUC not calculated.
Starting now')
407             viz = des.showROCAUC(dataTrain=self.
dataPackage.getTrainData(),
408                                dataTest=self.
dataPackage.getTestData(),
409                                classifier=self.
getClassifier(),
410                                axisLabels=axisLabels
        ,

```

```

411                 colNameActual=self.
dataPackage.targetColumn,
412                 features=self.
getBaseFeatures())
413                 self.__setBaseModelROCAUC(visualizer=viz)
414                 viz.show()
415
416     def __setBaseModelROCAUC(self,
417                               visualizer):
418         self.isBaseModelROCAUCCalculated = True
419         self.baseModelROCAUC = pickle.dumps(visualizer
    )
420
421     def __getBaseModelROCAUC(self):
422         return pickle.loads(self.baseModelROCAUC)
423
424     def showFinalModelROCAUC(self, axisLabels,
useStored=False):
425         if useStored and self.
isFinalModelROCAUCCalculated:
426             print('Final model ROCAUC already
calculated. Displaying stored results')
427             tViz = self.__getBaseModelROCAUC()
428             tViz.show()
429         else:
430             print('Final model ROCAUC not calculated.
Starting now')
431             viz = des.showROCAUC(dataTrain=self.
dataPackage.getTrainData(),
432                                 dataTest=self.
dataPackage.getTestData(),
433                                 classifier=self.
getClassifier(),
434                                 axisLabels=axisLabels
,
435                                 colNameActual=self.
dataPackage.targetColumn,
436                                 features=self.

```



```
436 getFinalFeatures())
437         self.__setFinalModelROCAUC(visualizer=viz)
438         viz.show()
439
440     def __getFinalModelROCAUC(self):
441         return pickle.loads(self.finalModelROCAUC)
442
443     def __setFinalModelROCAUC(self,
444                               visualizer):
445         self.isFinalModelROCAUCCalculated = True
446         self.finalModelROCAUC = pickle.dumps(
visualizer)
447
448     def showFinalModelReport(self,
449                             axisLabels
450                             ):
451         self.showFinalModelStats()
452
453         des.showReport(data=self.
getFinalModelPrediction(),
454                       colNameActual=self.
finalModelPredictionColActual,
455                       colNamePredict=self.
finalModelPredictionColPredict,
456                       axisLabels=axisLabels,
457                       titleSuffix=self.experimentName
)
458
459         self.showFinalModelLearningCurve()
460         self.showFinalModelROCAUC(axisLabels=
axisLabels)
461
462     def getBaseFeatures(self):
463         return self.dataPackage.dataFeatures
464
465     def getFinalFeatures(self):
466         return self.finalFeatures
467
```

```

468     def createBaseModelLearningCurve(self,
469                                     cv=None,
470                                     n_jobs=None,
471                                     train_sizes=None,
472                                     verbose=4):
473         # If it is already predicted just show it
474         if self.isBaseModelLearningCurveCreated:
475             print('Base model learning curve already
calculated. Displaying results:')
476             self.showBaseModelLearningCurve()
477         else:
478             df = self.dataPackage.getTrainData()
479             train_sizes, train_scores, test_scores,
fit_times = des.create_learning_curve(
480                 estimator=self.getClassifier(),
481                 X=df[self.dataPackage.dataFeatures],
482                 y=df[self.dataPackage.targetColumn],
483                 cv=cv,
484                 n_jobs=n_jobs,
485                 train_sizes=train_sizes,
486                 verbose=verbose)
487
488             self.__setBaseModelLearningData(
train_sizes=train_sizes,
489             train_scores=train_scores,
490             test_scores=test_scores,
491                                     fit_times=
fit_times)
492
493     def createFinalModelLearningCurve(self,
494                                     cv=None,
495                                     n_jobs=None,
496                                     train_sizes=None
,
497                                     verbose=4):
498         # If it is already predicted just show it

```

```

499         if self.isFinalModelLearningCurveCreated:
500             print('Final model learning curve already
calculated. Displaying results:')
501             self.showFinalModelLearningCurve()
502         else:
503
504             df = self.dataPackage.getTrainData()
505             train_sizes, train_scores, test_scores,
fit_times = des.create_learning_curve(
506                 estimator=self.getClassifier(),
507                 X=df[self.finalFeatures],
508                 y=df[self.dataPackage.targetColumn],
509                 cv=cv,
510                 n_jobs=n_jobs,
511                 train_sizes=train_sizes,
512                 verbose=verbose)
513             self.__setFinalModelLearningData(
train_sizes=train_sizes,
514
train_scores=train_scores,
515
test_scores=test_scores,
516
fit_times
=fit_times)
517
518         def __setBaseModelLearningData(self,
519                                         train_sizes,
520                                         train_scores,
521                                         test_scores,
522                                         fit_times):
523             self.isBaseModelLearningCurveCreated = True
524             self.baseModel_train_sizes = train_sizes
525             self.baseModel_train_scores = train_scores
526             self.baseModel_test_scores = test_scores
527             self.baseModel_fit_times = fit_times
528
529         def __setFinalModelLearningData(self,
530                                         train_sizes,

```

```

531                 train_scores,
532                 test_scores,
533                 fit_times):
534         self.isFinalModelLearningCurveCreated = True
535         self.finalModel_train_sizes = train_sizes
536         self.finalModel_train_scores = train_scores
537         self.finalModel_test_scores = test_scores
538         self.finalModel_fit_times = fit_times
539
540     def showBaseModelLearningCurve(self,
541                                   axes=None,
542                                   ylim=(0.0, 1.01)
543                                   ):
544         if self.isBaseModelLearningCurveCreated:
545
546             des.plot_learning_curve(train_sizes=self.
547 baseModel_train_sizes,
548                                   train_scores=self.
549 baseModel_train_scores,
550                                   test_scores=self.
551 baseModel_test_scores,
552                                   fit_times=self.
553 baseModel_fit_times,
554                                   title=self.
555 experimentName,
556                                   axes=axes,
557                                   ylim=ylim
558                                   )
559         else:
560             display('Base model Learning curve has not
561 yet been calculated')
562
563     def showFinalModelLearningCurve(self,
564                                     axes=None,
565                                     ylim=(0.0, 1.01)
566                                     ):
567         if self.isFinalModelLearningCurveCreated:
568             des.plot_learning_curve(train_sizes=self.

```

```

562 finalModel_train_sizes,
563                                     train_scores=self.
        finalModel_train_scores,
564                                     test_scores=self.
        finalModel_test_scores,
565                                     fit_times=self.
        finalModel_fit_times,
566                                     title=self.
        experimentName,
567                                     axes=axes,
568                                     ylim=ylim
569                                     )
570     else:
571         display('Final model Learning curve has
        not yet been calculated')
572
573     def __getFinalModelFeatures(self,
574                                     returnAbove=0.002,
575                                     includeUniqueAndTarget
        =False):
576         # get a list of the features that have been
        deemed important
577         # Get full list of features
578         features = self.dataPackage.dataFeatures
579
580         df, featureLabel, valueLabel = des.
        getModelFeatureImportance(self.getBaseModel())
581
582         retDf = des.analyzeModelFeatureImportance(data
        =df,
583         valueLabel=valueLabel,
584         returnAbove=returnAbove,
585         showSummary=False,
586         showPlot=False)

```

```
587
588         keepFeatures = retDf[featureLabel].to_list()
589
590         # Initialize important features list
591         features_important = []
592
593         for x in keepFeatures:
594             features_important.append(features[x])
595
596         if includeUniqueAndTarget:
597             # Feature list doesn't include target and
unique
598             features_important.append(self.dataPackage
599                                     .uniqueColumn)
600             features_important.append(self.dataPackage
601                                     .targetColumn)
602
603         return features_important
```