

```
1 # -*- coding: utf-8 -*-
2
3 import os
4 import pandas as pd
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 import matplotlib.gridspec as gridspec
8 import matplotlib.ticker as mtick
9 import numpy as np
10 import math
11 from yellowbrick.target import ClassBalance
12
13
14 # Start dataframe functions
15 def exploreDataframe(data, numRecords=1):
16     print("dataframe shape: " + str(data.shape))
17     print("\ndataframe info: ")
18     print(data.info())
19     print("\nTop " + str(numRecords) + " in dataframe")
20     display(data.head(numRecords))
21     print("\nBottom " + str(numRecords) + " in
    dataframe")
22     display(data.tail(numRecords))
23
24 def showUniqueColVals(series):
25     i = 0
26     for l in np.unique(series):
27         i += 1
28         print(l)
29     print("\n-->Unique values: " + str(i))
30
31
32 # End dataframe functions
33
34
35 # Start generic ML functions
36
37 # Create a summary frame for us to append results
```

```

38 def initSummaryFrame():
39     column_names = ["analysisStage",
40                     "columnName",
41                     "totalDocuments",
42                     "totalNulls",
43                     "totalCategories",
44                     "upperPerc",
45                     "lowerPerc",
46                     "percTotalDocuments",
47                     "percTotalCategories"
48                     ]
49     fnDf = pd.DataFrame(columns=column_names)
50     return fnDf
51
52
53 # Add columns for upper and lower boundaries
54 def addUpperAndLowerPercColumns(data,
55                                 upperPerc,
56                                 lowerPerc,
57                                 upperPercColName='
    upperPerc',
58                                 lowerPercColName='
    lowerPerc'):
59     fnDf = data
60     fnDf[upperPercColName] = upperPerc
61     fnDf[lowerPercColName] = lowerPerc
62     return fnDf
63
64
65 # Add results data to summary frame
66 def appendResultsData(resultsFrame,
67                        analysisStage,
68                        analysisColName,
69                        totalDocuments,
70                        totalNulls,
71                        totalCategories,
72                        upperPerc,
73                        lowerPerc,

```

```

74         upperPercColName='upperPerc',
75         lowerPercColName='lowerPerc',
76         displayAppendedFrame=False):
77     if len(resultsFrame) == 0:
78         # this is the first row (should be original)
79         # add the perc total documents/columns
80         percTotalDocuments = 1.0
81         percTotalCategories = 1.0
82     else:
83         allCategories = resultsFrame['totalCategories'
84 ].values[0]
85         allDocuments = resultsFrame['totalDocuments'].
86 values[0]
87         percTotalDocuments = round(totalDocuments /
88 allDocuments, 2)
89         percTotalCategories = round(totalCategories /
90 allCategories, 2)
91
92     new_row = {'analysisStage': analysisStage,
93               'columnName': analysisColName,
94               'totalDocuments': totalDocuments,
95               'totalNulls': totalNulls,
96               'totalCategories': totalCategories,
97               upperPercColName: upperPerc,
98               lowerPercColName: lowerPerc,
99               'percTotalDocuments':
100 percTotalDocuments,
101               'percTotalCategories':
102 percTotalCategories
103             }
104     # append row to the dataframe
105     resultsFrame = resultsFrame.append(new_row,
106 ignore_index=True)
107     if displayAppendedFrame:
108         print('Results appended. New frame:')
109         display(resultsFrame)
110

```

```
105     return resultsFrame
106
107
108 def buildReportingFrame(df,
109                         analysisColName,
110                         summaryColName,
111                         cumulativeColName,
112                         colOrderName,
113                         percTotalColName,
114                         percCumulativeColName,
115                         ):
116     tDf = df.copy()
117
118     tDf[analysisColName] = tDf[analysisColName].astype
119     ('string')
120
121     tDf = df.groupby([analysisColName]).size().
122     to_frame(summaryColName)
123     tDf = tDf.sort_values(by=summaryColName, ascending
124     =False)
125     tDf[cumulativeColName] = tDf[summaryColName].
126     cumsum()
127
128     totalRows = tDf[summaryColName].sum()
129
130     tDf[percTotalColName] = round(tDf[summaryColName
131     ] / totalRows, 2)
132     tDf[percCumulativeColName] = round(tDf[
133     cumulativeColName] / totalRows, 2)
134     tDf[colOrderName] = np.arange(len(tDf)) + 1
135
136     tDf.reset_index(inplace=True)
137     tDf.index = np.arange(1, len(tDf) + 1)
138
139     tDf = tDf[
140     [colOrderName, analysisColName, summaryColName
141     , cumulativeColName, percTotalColName,
142     percCumulativeColName]]
```

```

135
136     return tDf
137
138
139 def showColumnSummary(df,
140                        analysisColName
141                        ):
142     tDf = df.copy()
143     totalDocuments = len(tDf)
144     totalNulls = tDf[analysisColName].isnull().sum()
145     totalNAs = tDf[analysisColName].isna().sum()
146     totalCategories = tDf[analysisColName].nunique()
147
148     print(f'Dataframe shape {str(df.shape)}')
149     print(f'Analysis column: {analysisColName}')
150     print(f'Distinct values (incl. null): {str(
totalCategories)}')
151     print(f'Number of na values: {str(totalNAs)}')
152     print(f'Number of null values: {str(totalNulls)}')
153     print(f'Total documents in corpus: {str(
totalDocuments)}', end='\n\n')
154
155
156 def analyzeReportingFrame(df,
157                           summaryColName,
158                           analysisColName,
159                           resultsFrame,
160                           analysisName="original",
161                           upperPerc=1.0, # use
default for initial run
162                           lowerPerc=0.0, # use
default for initial run
163                           showRows=2):
164     tDf = df.copy()
165
166     totalDocuments = tDf[summaryColName].sum()
167     totalCategories = tDf[analysisColName].nunique()
168     totalNulls = tDf[analysisColName].isnull().sum()

```

```

169
170     print(f'Top {str(showRows)} for summary of column
      {analysisColName}')
171     display(tDf[:showRows])
172     print("", end='\n\n')
173
174     print(f'Last {str(showRows)} for summary of column
      {analysisColName}')
175     display(tDf.tail(showRows))
176     print("", end='\n\n')
177
178     resultsFrame = appendResultsData(resultsFrame,
179                                     analysisName,
180                                     analysisColName,
181                                     totalDocuments,
182                                     totalNulls,
183                                     totalCategories,
184                                     upperPerc=
      upperPerc,
185                                     lowerPerc=
      lowerPerc
186                                     )
187     return resultsFrame
188
189
190 def getFocusedDf(data,
191                 upperPerc,
192                 lowerPerc):
193     fnDf = addUpperAndLowerPercColumns(data=data,
194                                       upperPerc=
      upperPerc,
195                                       lowerPerc=
      lowerPerc
196                                       )
197
198     fnDf['upperInclude'] = np.where(fnDf['
      percCumulative'] <= fnDf['upperPerc'], 1, 0)
199     fnDf['lowerInclude'] = np.where(fnDf['percTotal']

```

```

199 ] >= fnDf['lowerPerc'], 1, 0)
200     fnDf = fnDf[(fnDf['upperInclude'] == 1) & (fnDf['
    lowerInclude'] == 1)]
201
202     return fnDf
203
204
205 def plotColumnAnalysis(df, xColName,
206                        percTotalColName,
207                        percCumulativeColName,
208                        analysisColName,
209                        summaryColName,
210                        upperPerc,
211                        lowerPerc,
212                        upperPercColName='upperPerc',
213                        lowerPercColName='lowerPerc'):
214     sns.set(rc={'figure.figsize': (20, 8)})
215     fnDf = addUpperAndLowerPercColumns(data=df,
216                                       upperPerc=
217     upperPerc,
218                                       lowerPerc=
219     lowerPerc
220                                       )
221
222     print(f'Lineplot showing {analysisColName}
    distribution')
223     fig = sns.lineplot(x=xColName, y='value', hue='
    variable', data=pd.melt(fnDf2, [xColName]))
224     fig.set(xlabel='Item rank', ylabel='Percent of all
    documents in corpus')
225     fig.set_title(f'Lineplot summary for: {
    analysisColName}')
226     fig.yaxis.set_major_formatter(mtick.
    PercentFormatter(1.0))
227     plt.show()

```

```

228     print("\n\n")
229
230     print(f'Barplot showing {analysisColName}
distribution')
231     if len(df) < 100:
232         plt.xticks(rotation='vertical')
233         fig = sns.barplot(x=analysisColName, y=
summaryColName, data=fnDf)
234         fig.set(xlabel=f'Items in summary column: {
analysisColName}', ylabel='Document Count')
235         fig.set_title(f'Barplot summary for: {
analysisColName}')
236         plt.show()
237         print("\n\n")
238     else:
239         print(f'--->Dataset too large for barchart
visibility: {str(len(df))}')
240         print('\n\n')
241
242
243 def plotSummaryFrame(dataFrame):
244     plotOne = ['percTotalDocuments', '
percTotalCategories']
245     plotTwo = ['totalDocuments', 'totalNulls', '
totalCategories']
246
247     tDf = pd.melt(dataFrame, id_vars=['analysisStage'
], value_vars=plotOne)
248     sns.barplot(x='variable', y='value', hue='
analysisStage', data=tDf)
249     plt.show()
250
251     print('\n\n')
252
253     tDf = pd.melt(dataFrame, id_vars=['analysisStage'
], value_vars=plotTwo)
254     sns.barplot(x='variable', y='value', hue='
analysisStage', data=tDf)

```



```
255     plt.show()
256
257
258 def columnExplore(dataFrame,
259                   analysisColName,
260                   upperPerc=0.9,
261                   lowerPerc=0.01,
262                   summaryColName='docCount',
263                   cumulativeColName='cumulativeCount',
264                   colOrderName='order',
265                   percTotalColName='percTotal',
266                   percCumulativeColName='
percCumulative'):
267     resultsFrame = initSummaryFrame()
268
269     print("Beginning analysis on 'Main' frame...")
270
271     showColumnSummary(df=dataFrame,
272                      analysisColName=analysisColName)
273
274     fnDf = buildReportingFrame(df=dataFrame,
275                               analysisColName=
analysisColName,
276                               summaryColName=
summaryColName,
277                               cumulativeColName=
cumulativeColName,
278                               colOrderName=
colOrderName,
279                               percTotalColName=
percTotalColName,
280                               percCumulativeColName=
percCumulativeColName)
281
282     resultsFrame = analyzeReportingFrame(df=fnDf,
283                                         resultsFrame=
resultsFrame,
284
```

```

284 summaryColName=summaryColName,
285     analysisColName=analysisColName)
286
287     plotColumnAnalysis(df=fnDf,
288                        xColName=colOrderName,
289                        analysisColName=analysisColName
290                        ,
291                        summaryColName=summaryColName,
292                        upperPerc=upperPerc,
293                        lowerPerc=lowerPerc,
294                        percTotalColName=
295                        percTotalColName,
296                        percCumulativeColName=
297                        percCumulativeColName)
298
299     print("Beginning analysis on 'Focused' frame
300     .....")
301     focusDf = getFocusedDf(data=fnDf,
302                            upperPerc=upperPerc,
303                            lowerPerc=lowerPerc)
304
305     resultsFrame = analyzeReportingFrame(df=focusDf,
306                                         resultsFrame=
307                                         resultsFrame,
308                                         analysisName=
309                                         "Trimmed",
310                                         upperPerc=0.9
311                                         ,
312                                         lowerPerc=0.
313                                         01,
314                                         summaryColName=summaryColName,
315                                         analysisColName=analysisColName)
316
317     plotColumnAnalysis(df=focusDf,
318                        xColName=colOrderName,

```

```

311         analysisColName=analysisColName
312     ,
313         summaryColName=summaryColName,
314         upperPerc=upperPerc,
315         lowerPerc=lowerPerc,
316         percTotalColName=
percTotalColName,
317         percCumulativeColName=
percCumulativeColName)
318     display(resultsFrame)
319     plotSummaryFrame(resultsFrame)
320
321     return resultsFrame
322
323
324 def setPlotSize(plotsize):
325     if plotsize == 5:
326         sns.set(rc={'figure.figsize': (20, 8)})
327     elif plotsize == 4:
328         sns.set(rc={'figure.figsize': (15, 8)})
329     elif plotsize == 3:
330         sns.set(rc={'figure.figsize': (10, 8)})
331     elif plotsize == 2:
332         sns.set(rc={'figure.figsize': (8, 8)})
333     elif plotsize == 1:
334         sns.set(rc={'figure.figsize': (4, 8)})
335     else: # Should be size 1
336         # should only be one but catch it and default
to size 1
337         sns.set(rc={'figure.figsize': (4, 4)})
338
339
340 def examineColumnNumeric(df, colName,
341                         binsize=1000,
342                         verbose=False,
343                         zoom=False,
344                         minZoomLevel=0,

```

```

345             maxZoomLevel=0,
346             plotsize=1,
347             numRecords=10
348         ):
349             binColName = f'bin_at_{str(binsize)}'
350             binnedCountName = 'binnedCount'
351
352             #Make a copy of the incoming frame as we will be
manipulating it
353             tDf = df.copy()
354
355             # Parameter checking
356             if (binsize <= 0):
357                 print(f'binsize of {str(binsize)} given. Must
358                 be > 0 and evenly divisible by 10 or = 1')
359                 return
360
361             if (binsize % 10 != 0) and (binsize != 1):
362                 print(f'binsize of {str(binsize)} given. Must
363                 be evenly divisible by 10 or = 1')
364                 return
365
366             if zoom:
367                 if maxZoomLevel < minZoomLevel:
368                     print(f'maxZoomLevel given as {str(
369                     maxZoomLevel)} which must ' +
370                     f'be >= minZoomLevel given as {str(
371                     minZoomLevel)}')
372                     return
373
374             if (maxZoomLevel % 10 != 0) or (minZoomLevel
375             % 10 != 0):
376                 print(f'both maxZoomLevel given as {str(
377                 maxZoomLevel)} ' +
378                 f'and minZoomLevel given as {str(
379                 minZoomLevel)} ' +
380                 f'must be evenly divisible by 10')
381                 return

```

```

375
376     #Set size of displayed plot
377     if plotsize < 1 or plotsize > 5:
378         print(f'plotsize given as {str(plotsize)} must
    be between 1 and 5')
379         return
380     else:
381         setPlotSize(plotsize)
382
383     #Null values make it flunk out.
384     numNullValues = tDf[colName].isnull().sum()
385     if numNullValues > 0:
386         print(f'Warning: {numNullValues} null values
    detected in column. Removing for analysis')
387         tDf = tDf.dropna(subset=[colName], axis=0)
388
389     #Groupby and summarize dataframe
390     tDf = round(tDf[[colName]], 0).astype(int)
391     tDf[binColName] = [int(math.trunc(val / binsize
    ) * binsize) for val in tDf[colName]]
392     tDf = tDf.groupby(binColName).size().to_frame(
    binnedCountName).sort_values([binColName], ascending=
    False)
393     tDf.reset_index(inplace=True)
394
395     #Zoom to applicable level
396     if zoom:
397         tDf = tDf.loc[(tDf[binColName] >= minZoomLevel
    ) & (tDf[binColName] <= maxZoomLevel)]
398         tDf.reset_index(drop=True, inplace=True)
399
400     #Show me how it went
401     plt.xticks(rotation='vertical')
402     fig = sns.barplot(x=binColName, y=binnedCountName
    , data=tDf, palette="crest")
403     fig.set_xlabel('Summary column: {0} (binned at {1
    })'.format(colName, str(binsize)), fontsize=15)
404     fig.xaxis.labelpad=20

```

```

405     fig.set_ylabel('Document Count', fontsize=15)
406
407     if zoom:
408         titleTail = '\nZoom factor [{0}:{1}]'.format(
minZoomLevel, maxZoomLevel)
409     else:
410         titleTail = ''
411     fig.set_title(f'Data dispersion summary for: {
colName} (binned at {str(binsize)}){titleTail}',
        fontsize=20)
412
413     plt.show()
414
415     if verbose:
416         exploreDataframe(tDf, numRecords=numRecords)
417
418
419 def classBalanceUndersample(df, columnName):
420     ttlColName = 'ttlCol'
421     tDf = df.copy()
422
423     visualizer = ClassBalance()
424     visualizer.fit(tDf[columnName])
425     visualizer.show()
426
427     tDfSize = tDf.groupby([columnName]).size().
to_frame(ttlColName).sort_values(by=ttlColName)
428     tDfSize.reset_index(inplace=True)
429     sample_size = pd.to_numeric(tDfSize[ttlColName][0
])
430     # display(tDfSize.head(5))
431     sample_class = tDfSize[columnName][0]
432     print(f'Undersampling data to match min class: {
str(sample_class)} of size: {sample_size}')
433     tDf = tDf.groupby(columnName, group_keys=False).
apply(lambda x: x.sample(sample_size))
434     tDf.reset_index(drop=True, inplace=True)
435

```

```
436     visualizer2 = ClassBalance()
437     visualizer2.fit(tDf[columnName])
438     visualizer2.show()
439
440     return tDf
441
442
443 def displayClassBalance(df, columnName, showRecords=5
444 ):
445     ttlColName = 'ttlCol'
446     tDf = df.copy()
447
448     visualizer = ClassBalance()
449     visualizer.fit(df[columnName]) # Fit the data to
450     the visualizer
451     visualizer.show() # Finalize and render the
452     figure
453
454     tDfSize = tDf.groupby([columnName]).size().
455     to_frame(ttlColName).sort_values(by=ttlColName)
456     tDfSize.reset_index(inplace=True)
457     tDfSize.head(5)
458
```