OL take home exam

March 16, 2021

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
from sklearn.metrics import recall_score, roc_auc_score,␣
 ↪classification_report,confusion_matrix,accuracy_score,precision_score,f1_score
from sklearn.ensemble import RandomForestClassifier,␣
 ↪AdaBoostClassifier,GradientBoostingClassifier,BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.combine import SMOTETomek


import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#load data and verify first 5 rows
df = pd.read_csv('ol_takehome_exam.csv')
df.head()
```

[2]:

	job_aptitude_exam	same_industry	classes_missed	hs_gpa	job_offered	\
0	89.827919	0	3	3.351572	1	
1	104.328788	0	1	2.653762	1	
2	94.331629	0	1	3.626603	1	
3	95.889929	1	3	0.732220	1	
4	102.692694	0	1	3.005002	1	

	good_behavior	high_school	illness	instructor	sensitive_680	…	\
0	NaN	211	0	inst_2	0.465284	…	
1	1.0	22	0	inst_6	1.368163	…	

```
2            0.0          310        0     inst_2      -0.149146  …
3            0.0          214        0     inst_6      -1.710868  …
4            0.0           55        0     inst_1       0.659796  …

   sensitive_343  sensitive_111  sensitive_455  sensitive_249  sensitive_446  \
0            NaN       0.048310       0.092214       1.702569       0.913870
1            NaN       1.041406      -1.611823       0.874068       0.717489
2            NaN       0.957701       1.030393       1.434821       1.181024
3      -0.743999      -0.063537      -2.393425      -0.158986       0.660245
4            NaN      -0.144108       0.334806       0.400220       0.275324

   sensitive_271  sensitive_663  sensitive_609  sensitive_151  sensitive_467
0       0.682981      -0.357153       0.002772       0.638673      -0.049358
1      -1.532020      -0.201669      -0.062293       1.294990      -2.469716
2       0.466697      -0.350472       1.199847       0.034484       0.926317
3       1.750983       1.298697      -0.944087       0.043960      -0.452384
4      -0.508607       0.203111       1.281182      -0.090057      -0.720526

[5 rows x 26 columns]
```

0.0.1 Univariate Analysis

[3]:
```
df.describe().T
```

[3]:

	count	mean	std	min	25%	\
job_aptitude_exam	5000.0	98.636325	11.827747	54.350631	90.807389	
same_industry	5000.0	0.021600	0.145388	0.000000	0.000000	
classes_missed	5000.0	1.366200	1.261117	0.000000	0.000000	
hs_gpa	4902.0	3.091835	0.619559	0.096073	2.783842	
job_offered	5000.0	0.896000	0.305291	0.000000	1.000000	
good_behavior	4492.0	0.239760	0.426984	0.000000	0.000000	
high_school	5000.0	191.944000	121.220357	0.000000	87.000000	
illness	5000.0	0.012000	0.108896	0.000000	0.000000	
sensitive_680	5000.0	0.040900	1.071373	-3.961977	-0.671667	
sensitive_128	5000.0	-0.022025	1.098761	-3.877101	-0.759034	
sensitive_503	5000.0	-0.024546	1.132415	-4.200902	-0.772006	
sensitive_572	5000.0	0.475690	0.563106	-0.910920	-0.019662	
sensitive_607	5000.0	0.496873	0.559451	-0.923998	0.002832	
sensitive_489	5000.0	-0.015480	1.133934	-4.201456	-0.790821	
sensitive_335	5000.0	-0.022330	1.122605	-4.650949	-0.767717	
sensitive_343	1951.0	-0.017148	1.113438	-3.689808	-0.774335	
sensitive_111	5000.0	0.512848	0.558641	-0.811995	-0.003494	
sensitive_455	5000.0	-0.032936	1.096461	-4.690186	-0.773552	
sensitive_249	5000.0	0.496444	0.556691	-0.768549	-0.002686	
sensitive_446	5000.0	0.497203	0.557031	-0.862406	-0.000928	
sensitive_271	5000.0	0.005603	1.130537	-4.081710	-0.759945	
sensitive_663	5000.0	0.516709	0.553950	-0.783504	0.015503	

```
sensitive_609      5000.0   -0.001314    1.124887  -3.851571  -0.730187
sensitive_151      5000.0    0.511728    0.560316  -0.881341   0.007054
sensitive_467      5000.0   -0.035830    1.131217  -5.665216  -0.785098

                         50%         75%         max
job_aptitude_exam   98.681432  106.993986  138.374073
same_industry        0.000000    0.000000    1.000000
classes_missed       1.000000    2.000000    8.000000
hs_gpa               3.174528    3.510580    4.646815
job_offered          1.000000    1.000000    1.000000
good_behavior        0.000000    0.000000    1.000000
high_school        184.500000  287.000000  442.000000
illness              0.000000    0.000000    1.000000
sensitive_680        0.033464    0.759104    3.733875
sensitive_128       -0.004123    0.731397    3.911621
sensitive_503       -0.031687    0.735283    4.003386
sensitive_572        0.410552    0.986352    1.800856
sensitive_607        0.462817    0.998094    1.818274
sensitive_489        0.004279    0.787556    3.751892
sensitive_335       -0.027841    0.707499    4.594052
sensitive_343       -0.013131    0.731907    3.406843
sensitive_111        0.571665    1.006714    1.795773
sensitive_455       -0.036700    0.710009    4.095858
sensitive_249        0.504463    0.992035    2.004538
sensitive_446        0.486210    0.991916    1.947355
sensitive_271       -0.001040    0.787328    4.083509
sensitive_663        0.572544    1.006449    1.801121
sensitive_609        0.004394    0.726331    4.357126
sensitive_151        0.554192    1.014138    1.834701
sensitive_467       -0.037292    0.710321    3.939717
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   job_aptitude_exam  5000 non-null   float64
 1   same_industry      5000 non-null   int64
 2   classes_missed     5000 non-null   int64
 3   hs_gpa             4902 non-null   float64
 4   job_offered        5000 non-null   int64
 5   good_behavior      4492 non-null   float64
 6   high_school        5000 non-null   int64
 7   illness            5000 non-null   int64
 8   instructor         4764 non-null   object
 9   sensitive_680      5000 non-null   float64
```

```
 10  sensitive_128      5000 non-null   float64
 11  sensitive_503      5000 non-null   float64
 12  sensitive_572      5000 non-null   float64
 13  sensitive_607      5000 non-null   float64
 14  sensitive_489      5000 non-null   float64
 15  sensitive_335      5000 non-null   float64
 16  sensitive_343      1951 non-null   float64
 17  sensitive_111      5000 non-null   float64
 18  sensitive_455      5000 non-null   float64
 19  sensitive_249      5000 non-null   float64
 20  sensitive_446      5000 non-null   float64
 21  sensitive_271      5000 non-null   float64
 22  sensitive_663      5000 non-null   float64
 23  sensitive_609      5000 non-null   float64
 24  sensitive_151      5000 non-null   float64
 25  sensitive_467      5000 non-null   float64
dtypes: float64(20), int64(5), object(1)
memory usage: 1015.8+ KB
```

good_behavior, hs_gpa, instructor and sensitive_343 have missing values.

sensitive_343 is missing ~61% of it's values.

good_behavior is missing 508 values ~10%.

hs_gpa is missing in 98 records ~2%.

instructor is missing in 236 racords ~4.75%

[5]:
```
for col in df.columns:
    print(col + ':')
    print(df[col].unique())
    print(df[col].nunique())
```

```
job_aptitude_exam:
[ 89.82791882 104.32878751  94.33162867 … 120.53428526  74.65327219
 107.56946268]
5000
same_industry:
[0 1]
2
classes_missed:
[3 1 5 0 2 4 6 7 8]
9
hs_gpa:
[3.35157189 2.65376223 3.62660276 … 3.23150804 2.91004794 2.16073077]
4902
job_offered:
```

```
[1 0]
2
good_behavior:
[nan  1.  0.]
2
high_school:
[211  22 310 214  55 277 159 309 399  94  93 151 186 389  67 273 303   2
 203  24  13 139 300 239 330 114 421 126 143 269 281 366   7 315 256 348
  97 413 147  91 157   9 108  18  62 240 123   0 412 306 213 404 387  31
  30 247  10 125   4 238 290 221 189 437  54  37 105  23  35 381  17 267
  63  34 338 210  57 295  77  49 232 372 284 137 379 305  61 173 199  96
 175 112   1  99 102  26 235 182  58 375 298 321  27 430  40 200 144 225
 322 252  11 129  87 365 149 230 313 187 208 343 219 156  28 414 113 439
 195 344 111 106 249 415  74 169  41 127 161 229 367 382  60 292 342 119
 251 141 180 244  83 201 318 312 320 228 431  32 226   8 347 266 438 353
 246 356 442  29 422 197  50  51 411 133  85 231 436 409 100 291   3  95
 332   6 135  12  76 418 115  73 360 250 276 218 188 334  44 416 314 202
  75 134 118  82  89 346 288  98 242 275  81 154 299  14 168 109 354 280
  80 216 283  71 319 158 101 304 164 417 153  43 326 373  19 392  88 110
 190 386 138 167 248 217  79 193  86 237 142 419 136 255 427  25 170 432
 254 335  68 352 196 270 241  69 174 103 124 307 376 148 302 397 393 311
  56 183 146 104   5  70 395 308 408 181 282 163 236 364  84  90 407  45
 287 424 212 261 224 260 337  16 441 145 339 165  46 223 297  92 324 293
 206 401 279 140 272 130 336 263 194  42 371 410 204 262  52 398 121 385
 166 368 345 383 258 150  38 192  21 172 171 160 420 162  39 405  78  64
 131 259 132 341 253 184 391 357 350 220 402 370 377  47 179 316 294  36
 265 191 406 215  20 178 122 286 268 205 355 323 378 394 429 423 359 285
 177 120 271 327 396  59  66 233 222 361 388  53 329 278 351 257 362 176
 128 426 185 390 301 117 155  65 331 380  72 207  15  48 274 227 434 152
 435 264 325 209 245 107 363 374 234 317 428 425 328 116 358 243 369 198
  33 289 400 340 333 296 349 403 433 440 384]
443
illness:
[0 1]
2
instructor:
['inst_2' 'inst_6' 'inst_1' 'inst_4' 'inst_5' 'inst_7' 'inst_3' 'inst_10'
 'inst_8' 'inst_12' 'inst_9' nan 'inst_0' 'inst_11' 'inst_13']
14
sensitive_680:
[ 0.46528368  1.36816295 -0.14914628 … -1.2875998   0.54519974
  0.86646256]
5000
sensitive_128:
[-1.34471661  0.33536711  1.25960361 … -1.21386567 -0.90800648
 -0.23946397]
5000
sensitive_503:
```

```
[-0.76860387 -0.45087832  0.6396518  …  0.60393106 -0.874631
  1.37021193]
5000
sensitive_572:
[-0.03399402 -0.11798781  0.30125164 …  0.27478971  1.11028668
  0.16981123]
5000
sensitive_607:
[-0.28089761 -0.08165145  1.22569012 …  0.9684104  -0.53418552
  0.73191511]
5000
sensitive_489:
[ 0.29773166 -0.85710309  0.36436941 … -0.44963768 -0.26046757
  0.37851283]
5000
sensitive_335:
[-1.08711659  1.3266578   1.29032117 …  0.28212512 -0.15021403
  0.42849385]
5000
sensitive_343:
[        nan -0.74399851  1.09155644 …  1.60621404  0.34966264
 -0.16672945]
1951
sensitive_111:
[ 0.04831037  1.04140603  0.95770107 … -0.24424774 -0.02283043
  1.30257112]
5000
sensitive_455:
[ 0.0922145  -1.6118234   1.03039343 … -1.42197382 -0.79606973
 -1.31541351]
5000
sensitive_249:
[1.70256941 0.87406784 1.43482135 … 1.00360471 0.2241468  1.44987119]
5000
sensitive_446:
[0.91386996 0.71748851 1.18102406 … 0.4674336  0.09041776 0.84809794]
5000
sensitive_271:
[ 0.68298078 -1.53201956  0.46669712 …  1.97129742  0.4040221
 -0.75109089]
5000
sensitive_663:
[-0.3571529  -0.20166891 -0.35047177 …  0.05230231  0.03599499
  0.77458743]
5000
sensitive_609:
[ 0.00277214 -0.06229297  1.19984721 … -2.66248086 -1.11984581
 -0.07077162]
```

```
5000
sensitive_151:
[ 0.63867321  1.29498958  0.03448389 … -0.23497795  0.61440613
  1.09977904]
5000
sensitive_467:
[-0.0493575  -2.46971581  0.92631722 … -2.62842472  0.45408469
  0.620044  ]
5000
```

0.0.2 Null instructor (object) values will be dropped, since it is a lower percentage of total data and there is no reasonable way to substitute a value for this variable.

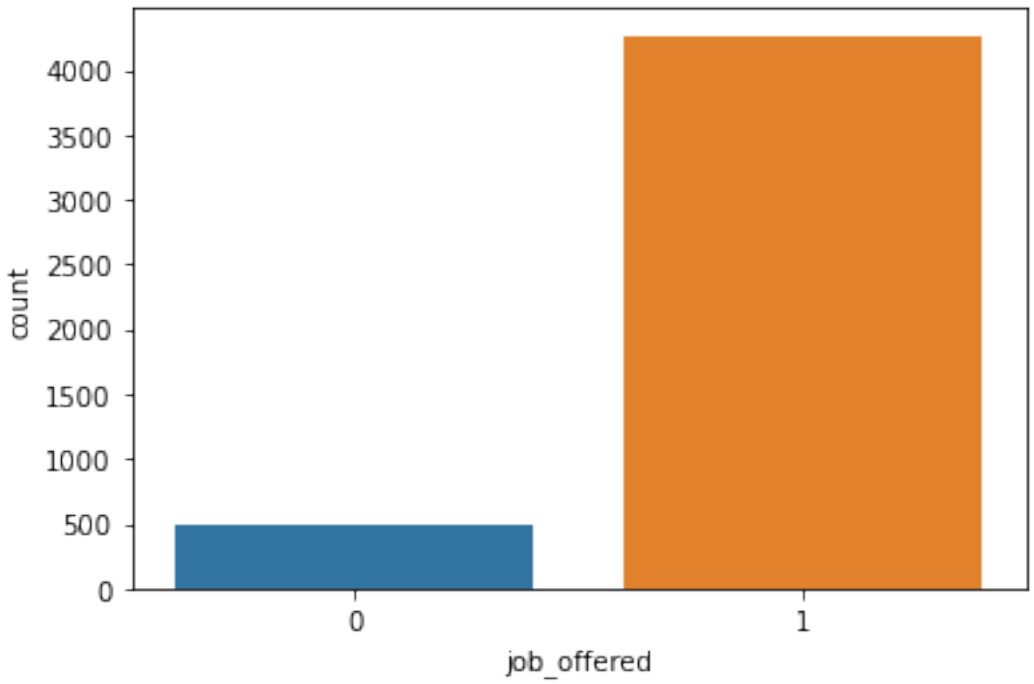
[6]:
```
#drop null instructor values.  could not reasonably substitute data for this␣
 ↪column
df.dropna(subset=['instructor'], inplace=True,)
df[df['instructor'].isnull()==True]
```

[6]:
```
Empty DataFrame
Columns: [job_aptitude_exam, same_industry, classes_missed, hs_gpa, job_offered,
good_behavior, high_school, illness, instructor, sensitive_680, sensitive_128,
sensitive_503, sensitive_572, sensitive_607, sensitive_489, sensitive_335,
sensitive_343, sensitive_111, sensitive_455, sensitive_249, sensitive_446,
sensitive_271, sensitive_663, sensitive_609, sensitive_151, sensitive_467]
Index: []

[0 rows x 26 columns]
```

[7]:
```
data_shape = df.shape
#check imbalance of data.  I suspect job_offered = 1 dominates the dataset
print('persons who were offered the job account for {}% of the dataset'.
 ↪format(100*round(df[df['job_offered'] == 1]['job_offered'].count()/
 ↪data_shape[0], 3)))
sns.countplot(df['job_offered'])
```

```
persons who were offered the job account for 89.5% of the dataset
```

[7]:
```
<AxesSubplot:xlabel='job_offered', ylabel='count'>
```



[ ]:

0.0.3 Replace null values for continuous variables with median values for both training set. Testing set has no null values

[8]:
```
df.loc[df['sensitive_343'].isnull() == True, 'sensitive_343'] =␣
 ↪df['sensitive_343'].median()
df.loc[df['hs_gpa'].isnull() == True, 'hs_gpa'] = df['hs_gpa'].median()
df.loc[df['good_behavior'].isnull() == True, 'good_behavior'] =␣
 ↪df['good_behavior'].median()
df.isnull().sum()
```

[8]:
```
job_aptitude_exam    0
same_industry        0
classes_missed       0
hs_gpa               0
job_offered          0
good_behavior        0
high_school          0
illness              0
instructor           0
sensitive_680        0
sensitive_128        0
sensitive_503        0
```

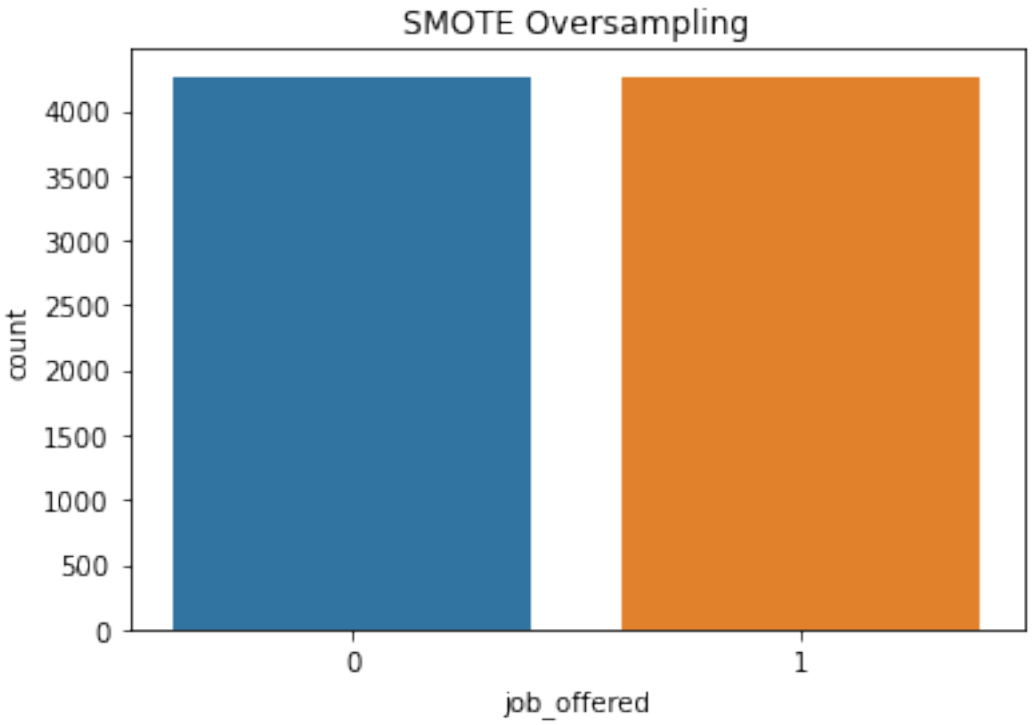
```
sensitive_572        0
sensitive_607        0
sensitive_489        0
sensitive_335        0
sensitive_343        0
sensitive_111        0
sensitive_455        0
sensitive_249        0
sensitive_446        0
sensitive_271        0
sensitive_663        0
sensitive_609        0
sensitive_151        0
sensitive_467        0
dtype: int64
```

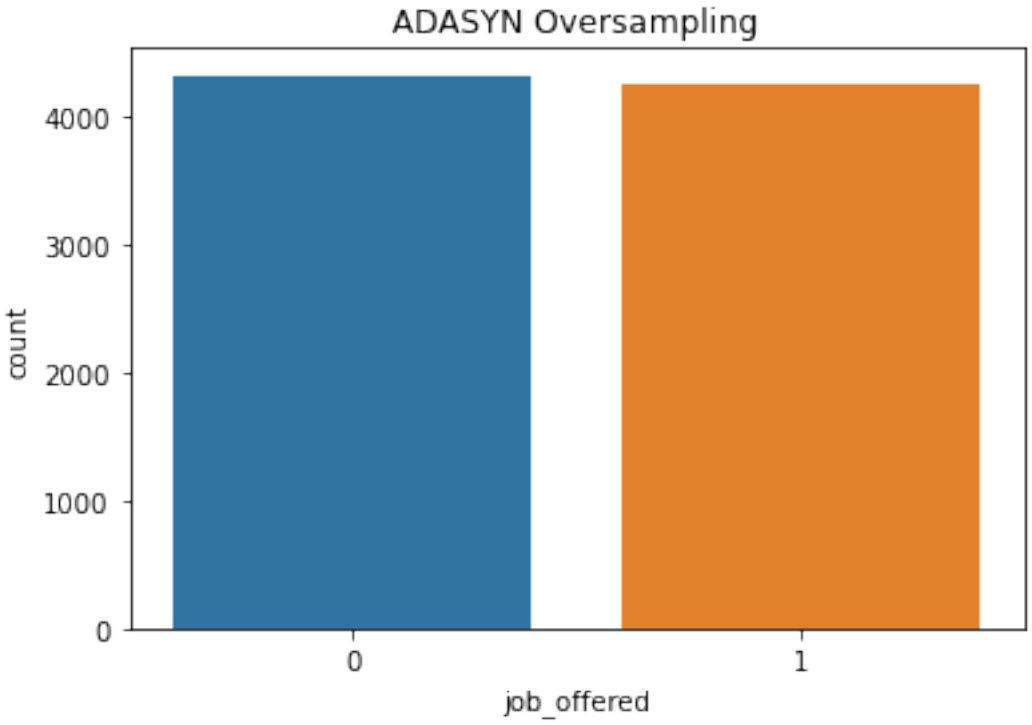
[9]:
```
#convert instructor to numerical value for inclusion in pairplot/visualization. Column will be converted to one-hot prior to model building in favor of one-hot encoding
df.loc[df['instructor'] == 'inst_0', 'instructor'] = 0
df.loc[df['instructor'] == 'inst_1', 'instructor'] = 1
df.loc[df['instructor'] == 'inst_2', 'instructor'] = 2
df.loc[df['instructor'] == 'inst_3', 'instructor'] = 3
df.loc[df['instructor'] == 'inst_4', 'instructor'] = 4
df.loc[df['instructor'] == 'inst_5', 'instructor'] = 5
df.loc[df['instructor'] == 'inst_6', 'instructor'] = 6
df.loc[df['instructor'] == 'inst_7', 'instructor'] = 7
df.loc[df['instructor'] == 'inst_8', 'instructor'] = 8
df.loc[df['instructor'] == 'inst_9', 'instructor'] = 9
df.loc[df['instructor'] == 'inst_10', 'instructor'] = 10
df.loc[df['instructor'] == 'inst_11', 'instructor'] = 11
df.loc[df['instructor'] == 'inst_12', 'instructor'] = 12
df.loc[df['instructor'] == 'inst_13', 'instructor'] = 13
df['instructor'] = df['instructor'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4764 entries, 0 to 4999
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   job_aptitude_exam  4764 non-null   float64
 1   same_industry      4764 non-null   int64
 2   classes_missed     4764 non-null   int64
 3   hs_gpa             4764 non-null   float64
 4   job_offered        4764 non-null   int64
 5   good_behavior      4764 non-null   float64
 6   high_school        4764 non-null   int64
```

```
 7   illness            4764 non-null   int64
 8   instructor         4764 non-null   int32
 9   sensitive_680      4764 non-null   float64
 10  sensitive_128      4764 non-null   float64
 11  sensitive_503      4764 non-null   float64
 12  sensitive_572      4764 non-null   float64
 13  sensitive_607      4764 non-null   float64
 14  sensitive_489      4764 non-null   float64
 15  sensitive_335      4764 non-null   float64
 16  sensitive_343      4764 non-null   float64
 17  sensitive_111      4764 non-null   float64
 18  sensitive_455      4764 non-null   float64
 19  sensitive_249      4764 non-null   float64
 20  sensitive_446      4764 non-null   float64
 21  sensitive_271      4764 non-null   float64
 22  sensitive_663      4764 non-null   float64
 23  sensitive_609      4764 non-null   float64
 24  sensitive_151      4764 non-null   float64
 25  sensitive_467      4764 non-null   float64
dtypes: float64(20), int32(1), int64(5)
memory usage: 1.1 MB
```

[10]:
```
#oversample minority class
y=df['job_offered']
X=df.drop('job_offered', axis=1)

# SMOTE Oversampling
smt = SMOTE(sampling_strategy='auto')
X_smt, y_smt = smt.fit_resample(X, y)

sns.countplot(y_smt)
plt.title('SMOTE Oversampling')
plt.show()
df_oversampled = pd.merge(X_smt, y_smt, left_index=True, right_index=True)

# ADASYN
adasyn = ADASYN(sampling_strategy='auto')
X_adasyn, y_adasyn = adasyn.fit_resample(X, y)
sns.countplot(y_adasyn)
plt.title('ADASYN Oversampling')
plt.show()

df_oversampled2 = pd.merge(X_adasyn, y_adasyn, left_index=True,␣
 ↪right_index=True)
```


SMOTE Oversampling
4000
3500
3000
2500
2000
1500
1000
500
0
count
0
1
job_offered


ADASYN Oversampling
4000
3000
2000
1000
0
count
0
1
job_offered

0.1 Visualize data

[11]:
```
df.head()
```

[11]:

	job_aptitude_exam	same_industry	classes_missed	hs_gpa	job_offered	\
0	89.827919	0	3	3.351572	1	
1	104.328788	0	1	2.653762	1	
2	94.331629	0	1	3.626603	1	
3	95.889929	1	3	0.732220	1	
4	102.692694	0	1	3.005002	1	

	good_behavior	high_school	illness	instructor	sensitive_680	…	\
0	0.0	211	0	2	0.465284	…	
1	1.0	22	0	6	1.368163	…	
2	0.0	310	0	2	-0.149146	…	
3	0.0	214	0	6	-1.710868	…	
4	0.0	55	0	1	0.659796	…	

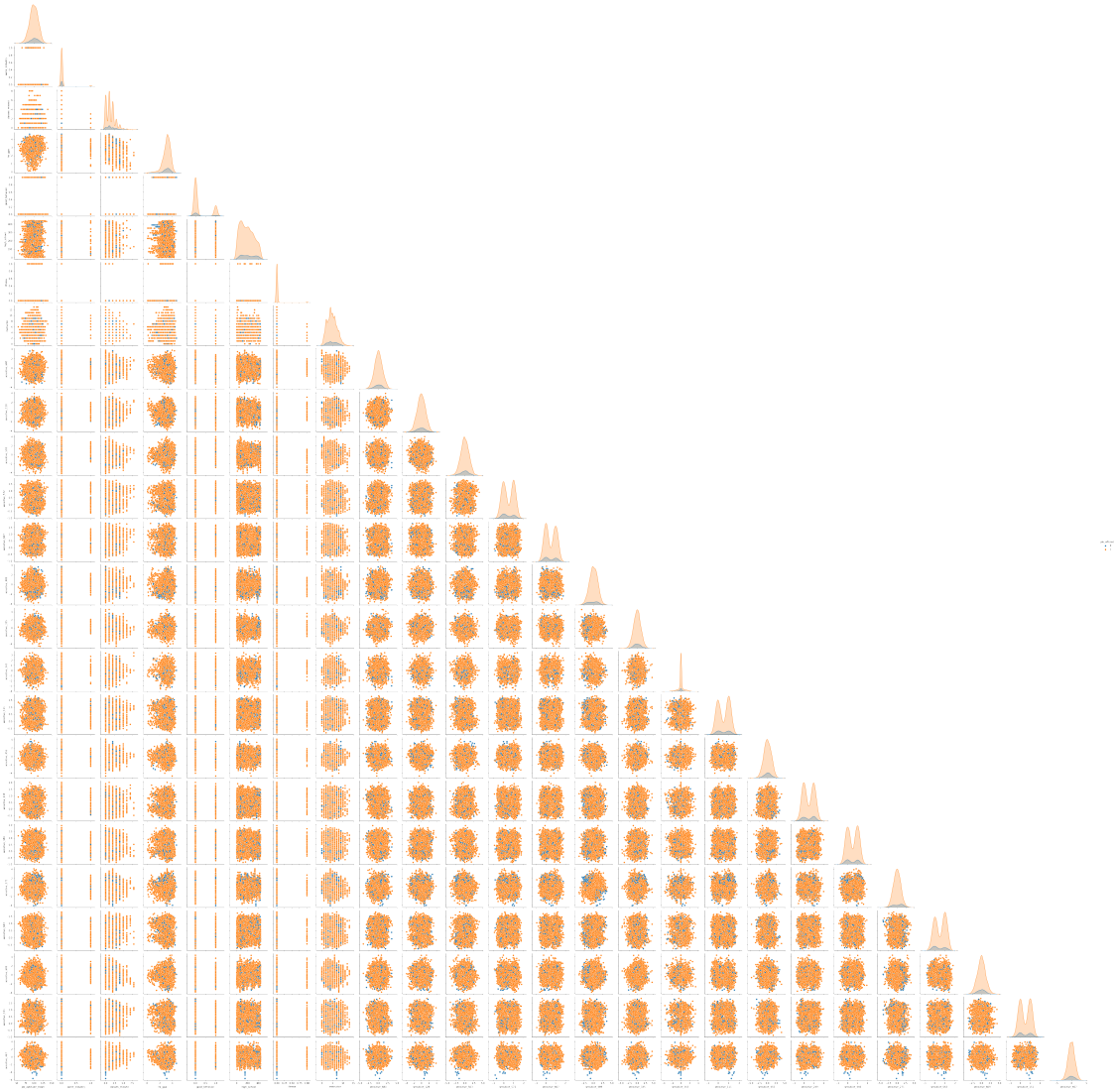
	sensitive_343	sensitive_111	sensitive_455	sensitive_249	sensitive_446	\
0	-0.001687	0.048310	0.092214	1.702569	0.913870	
1	-0.001687	1.041406	-1.611823	0.874068	0.717489	
2	-0.001687	0.957701	1.030393	1.434821	1.181024	
3	-0.743999	-0.063537	-2.393425	-0.158986	0.660245	
4	-0.001687	-0.144108	0.334806	0.400220	0.275324	

	sensitive_271	sensitive_663	sensitive_609	sensitive_151	sensitive_467
0	0.682981	-0.357153	0.002772	0.638673	-0.049358
1	-1.532020	-0.201669	-0.062293	1.294990	-2.469716
2	0.466697	-0.350472	1.199847	0.034484	0.926317
3	1.750983	1.298697	-0.944087	0.043960	-0.452384
4	-0.508607	0.203111	1.281182	-0.090057	-0.720526

[5 rows x 26 columns]

[12]:
```
#plot overview of densitys/histos and scatterplots.
plt.figure(figsize=(12,12));
sns.pairplot(df, hue='job_offered', corner=True);
```

```
<Figure size 864x864 with 0 Axes>
```



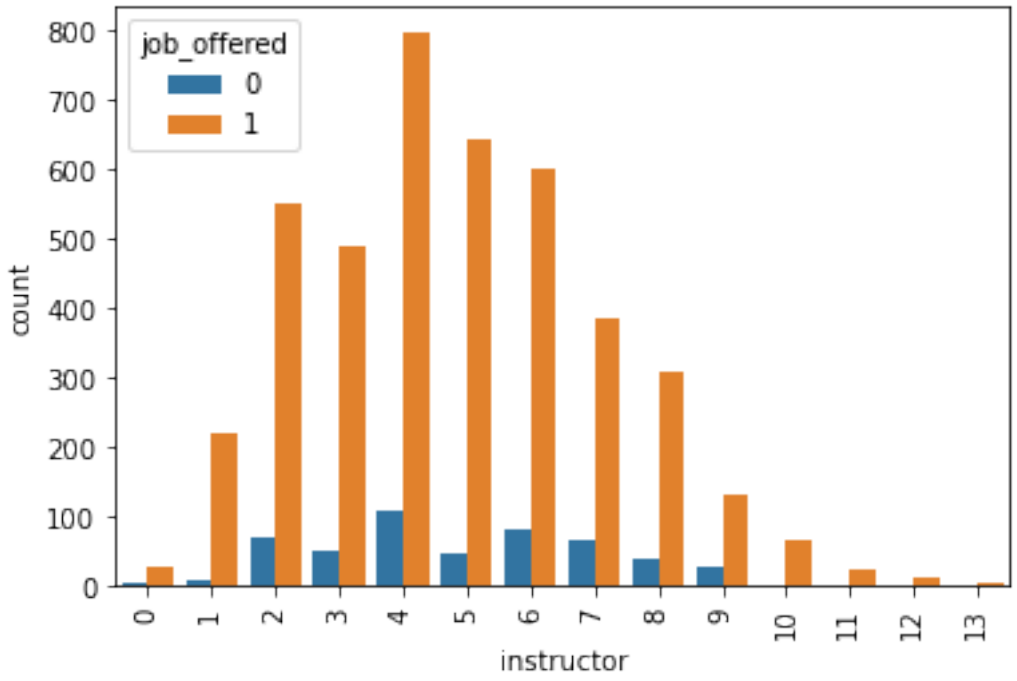
1 Verify passing rate by instructor

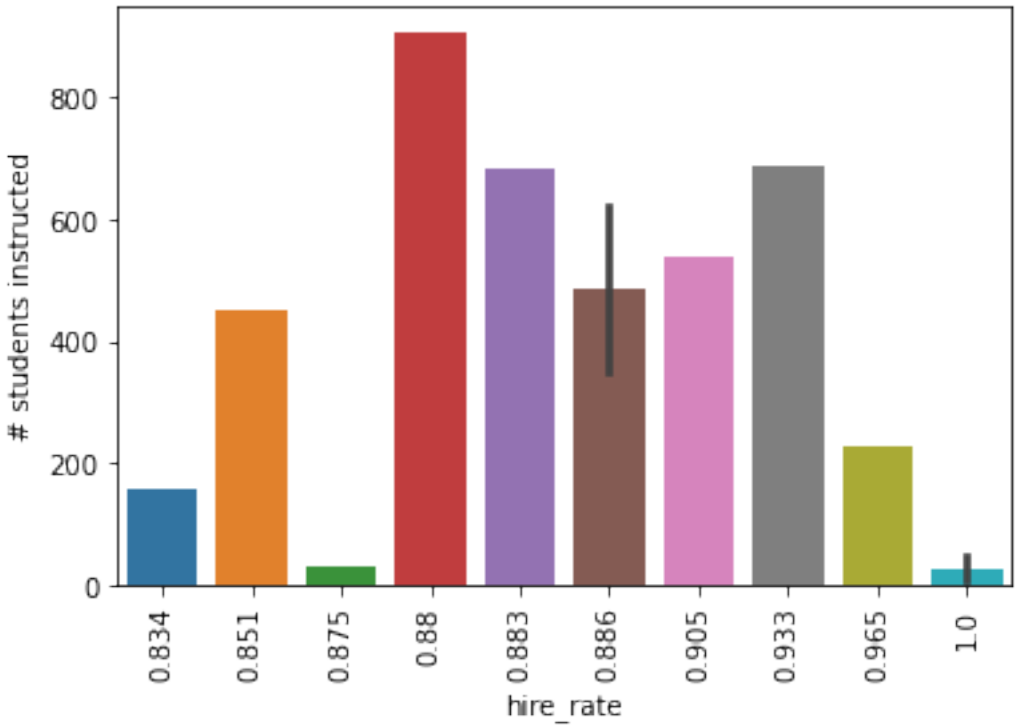
the above pairplot shows a bimodal distribution for job offered but appears to have a single mode for job not offered. Simply put which instructor the student had seems to have an effect on whether they were offered the job or not.

drilling down further:

[13]:
```
sns.countplot(df['instructor'], hue=df['job_offered'])
plt.xticks(rotation=90)
plt.show()

num_students = []
hire_rate = []
```

```
instructors = df['instructor'].unique()
for instructor in instructors:
    hire_rate.append(round(df[df['instructor']== instructor]['job_offered'].
 ↪sum()/df[df['instructor']==instructor]['instructor'].count(),3))
    num_students.append(df[df['instructor']==instructor]['instructor'].count())

output_df = pd.DataFrame()
output_df['instructors'] = instructors
output_df['hire_rate'] = hire_rate
output_df['# students instructed'] = num_students
output_df.sort_values(inplace=True, by='hire_rate')
sns.barplot(output_df['hire_rate'], y=output_df['# students instructed'])
plt.xticks(rotation=90)
plt.show()
output_df
```





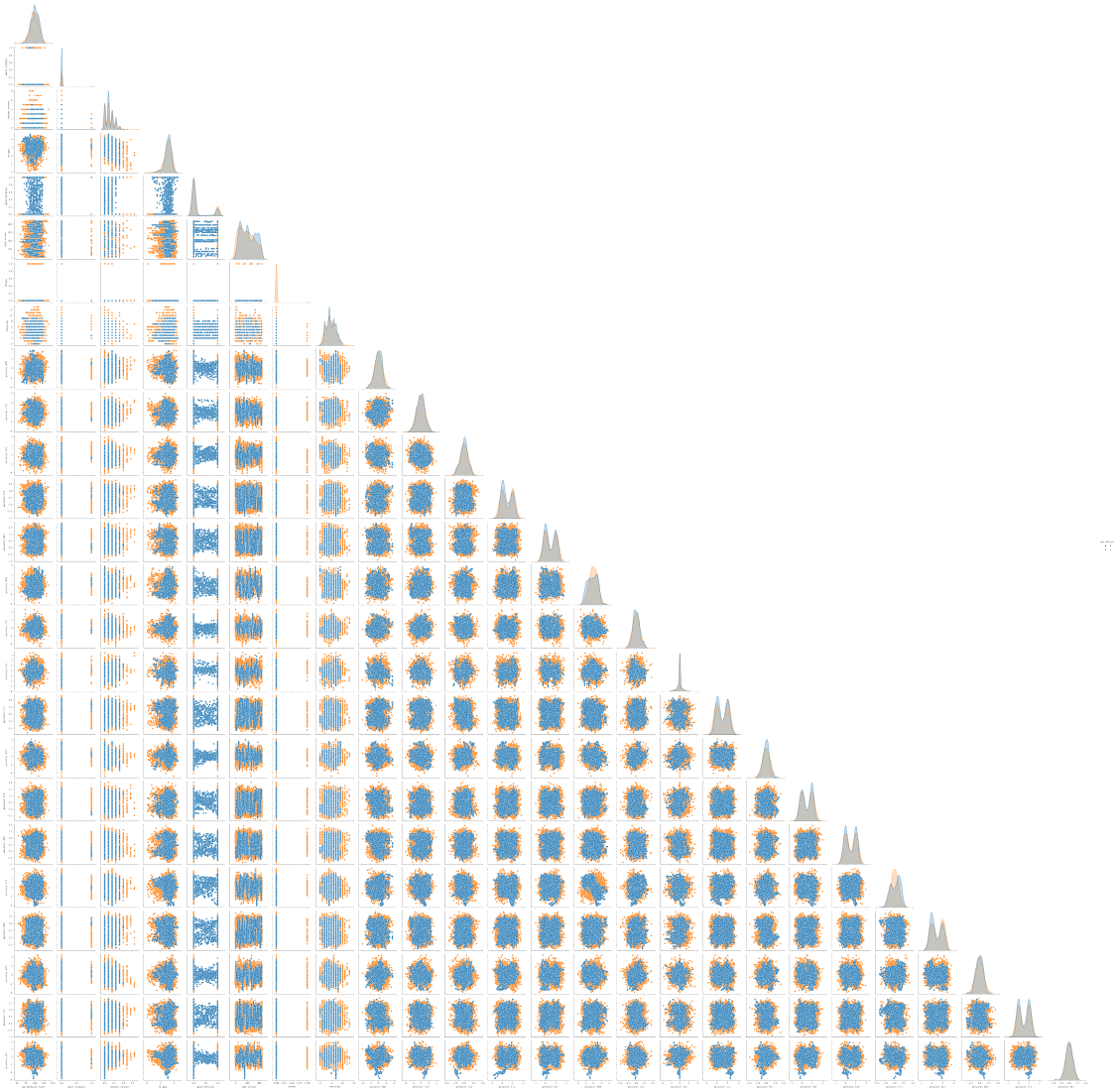
[13]:

	instructors	hire_rate	# students instructed
10	9	0.834	157
5	7	0.851	451
11	0	0.875	32
3	4	0.880	907
1	6	0.883	683
0	2	0.886	623
8	8	0.886	350
6	3	0.905	539
4	5	0.933	690
2	1	0.965	229
7	10	1.000	64
9	12	1.000	11
12	11	1.000	23
13	13	1.000	5

1.0.1 students who had instructors 10, 11, 12, and 13 have the highest job offer rates 100%. Students who had instructors 0, 2, 4, 6, 7, 8, 9 received offers less than 90% of the time. Overall, instructors with the lowest number of students had the highest offer rate. Instructors 0 and 9 have relatively low number of students, but have low acceptance rate.

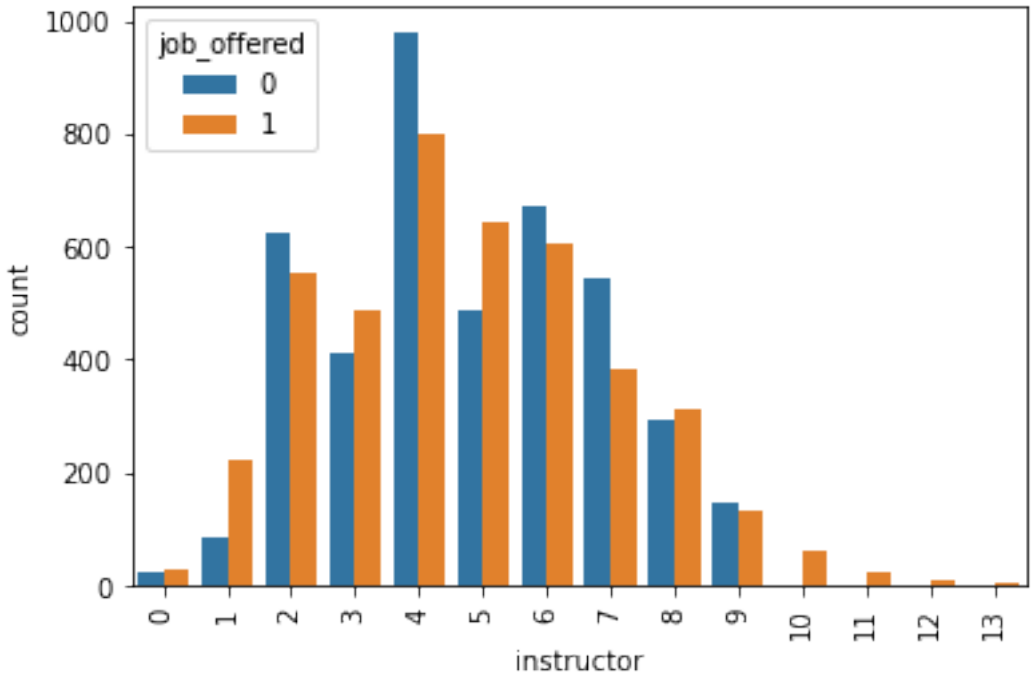
[14]:
```
#plot overview of densitys/histos and scatterplots.
plt.figure(figsize=(12,12));
sns.pairplot(df_oversampled, hue='job_offered', corner=True);
```

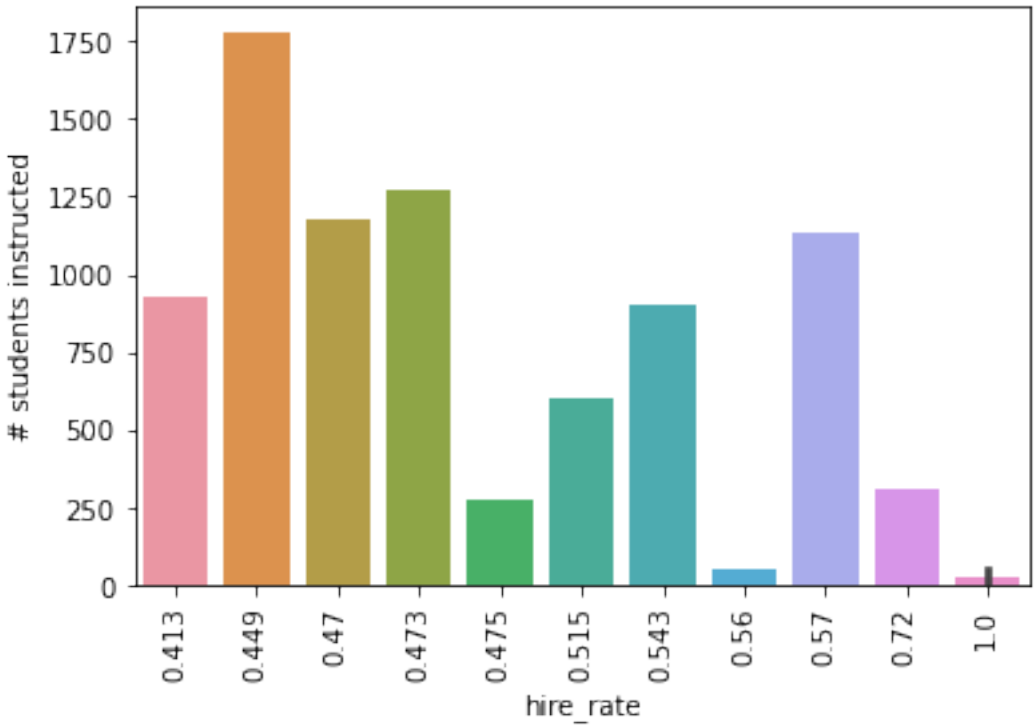
```
<Figure size 864x864 with 0 Axes>
```



[15]:
```
sns.countplot(df_oversampled['instructor'], hue=df_oversampled['job_offered'])
plt.xticks(rotation=90)
```

```
plt.show()

num_students = []
hire_rate = []
instructors = df_oversampled['instructor'].unique()
for instructor in instructors:
    hire_rate.append(round(df_oversampled[df_oversampled['instructor']==␣
 ↪instructor]['job_offered'].sum()/
 ↪df_oversampled[df_oversampled['instructor']==instructor]['instructor'].
 ↪count(),3))
    num_students.
 ↪append(df_oversampled[df_oversampled['instructor']==instructor]['instructor'].
 ↪count())

output_df = pd.DataFrame()
output_df['instructors'] = instructors
output_df['hire_rate'] = hire_rate
output_df['# students instructed'] = num_students
output_df.sort_values(inplace=True, by='hire_rate')
sns.barplot(output_df['hire_rate'], y=output_df['# students instructed'])
plt.xticks(rotation=90)
plt.show()
output_df
```



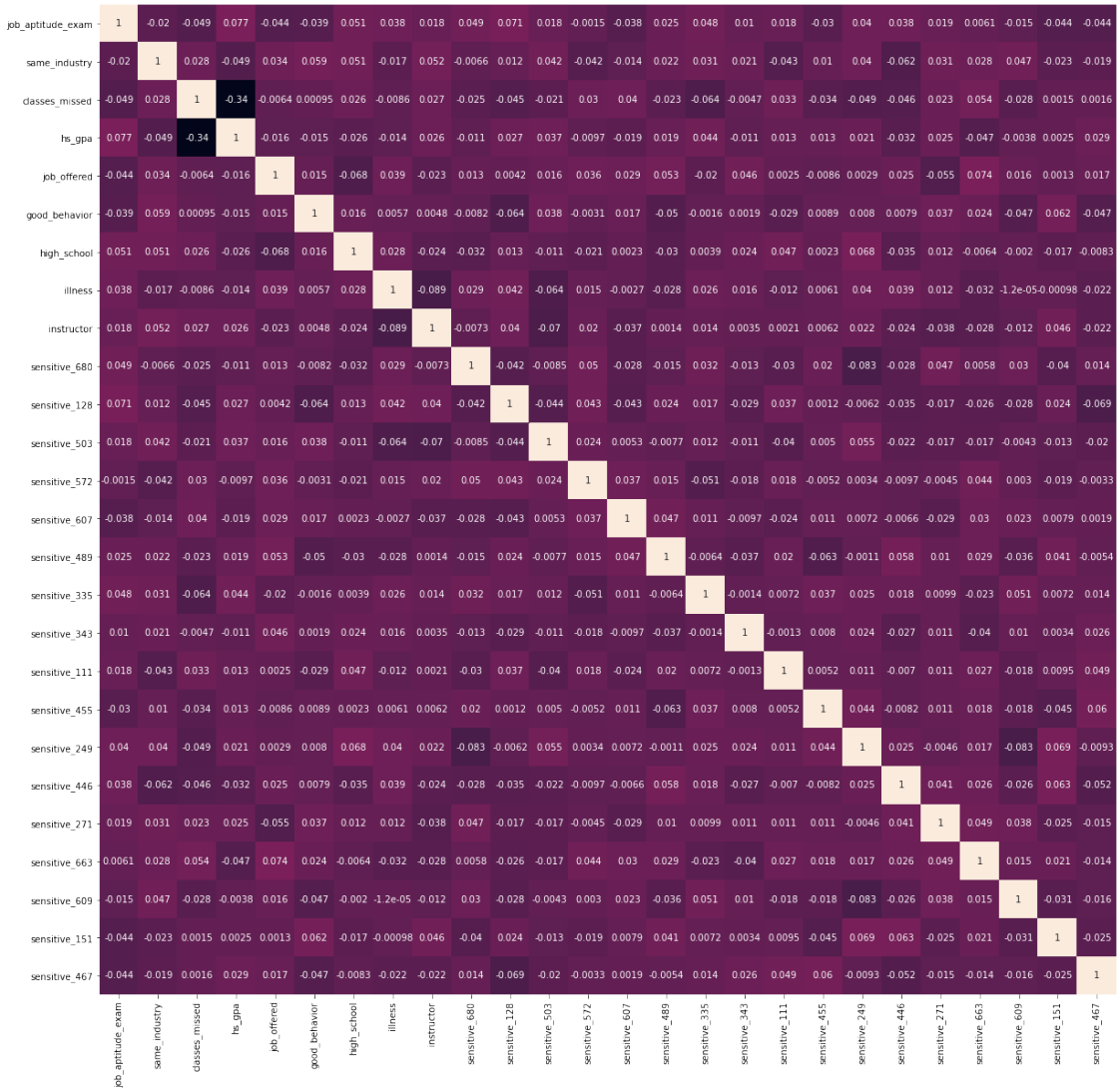


[15]:

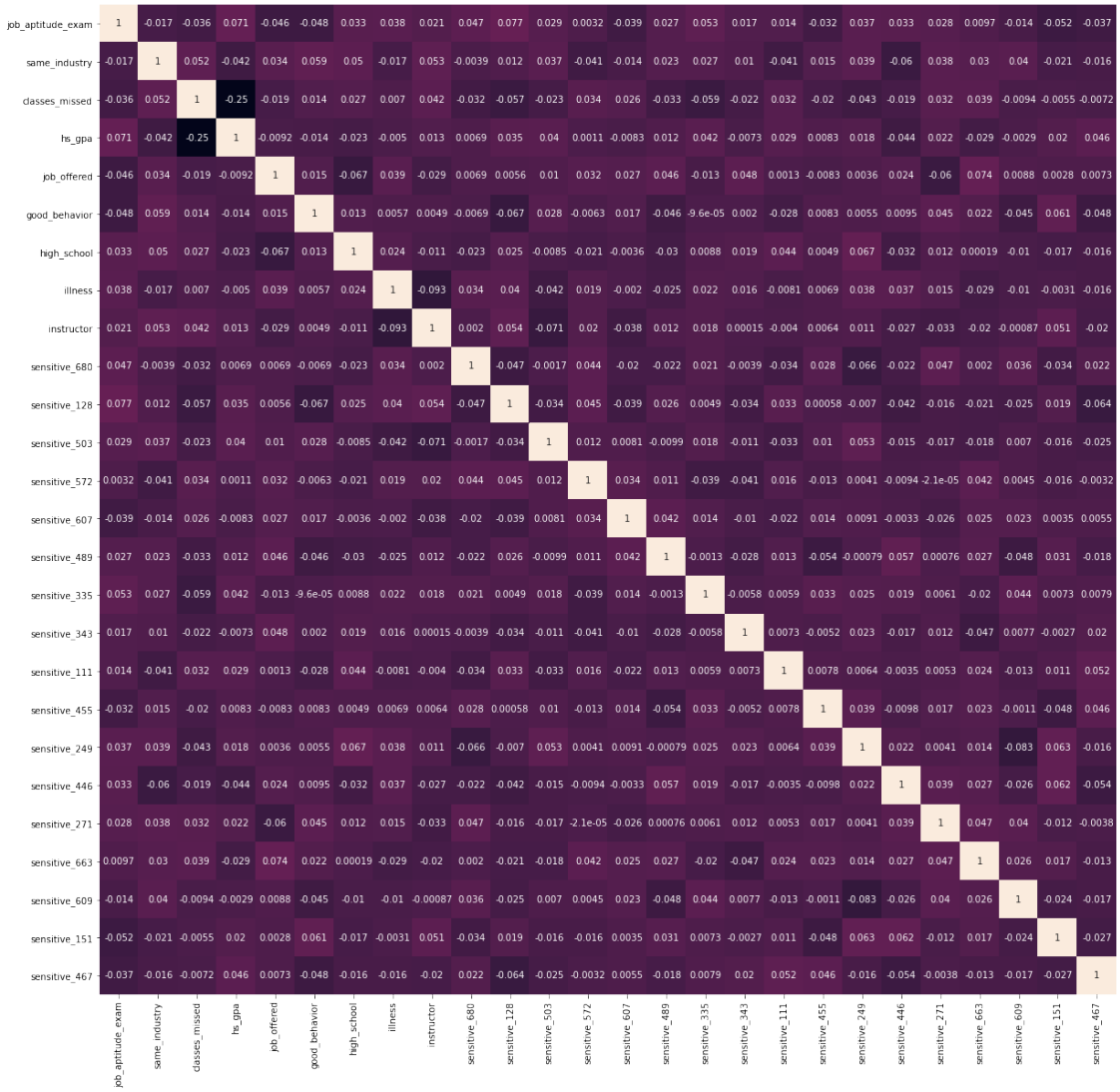
	instructors	hire_rate	# students instructed
5	7	0.413	930
3	4	0.449	1777
0	2	0.470	1175
1	6	0.473	1275
10	9	0.475	276
8	8	0.515	602
6	3	0.543	899
11	0	0.560	50
4	5	0.570	1130
2	1	0.720	307
7	10	1.000	64
9	12	1.000	11
12	11	1.000	23
13	13	1.000	5

[ ]:

[16]:
```
plt.figure(figsize=(20,20));
sns.heatmap(df.corr(), cbar=False, annot=True, );
```



[17]:
```
plt.figure(figsize=(20,20));
sns.heatmap(df.corr('spearman'), cbar=False, annot=True);
```



1.0.2 There is some inverse correlation between classes missed and hs_gpa (-0.34), unlikely to cause issues in building a model. Simply put, it appears attendance and gpa are mildly related, better attendance leads to better grades. The effect size is related to the magnitude of correlation. Attendance and hs_gpa have a mild association

[18]:
```
#perform one hot encoding on instructor value
df_temp = pd.get_dummies(df['instructor'])
df = pd.merge(df, df_temp, left_index=True, right_index=True)
df.drop('instructor', inplace=True, axis=1)
```

[19]:
```
df.columns
```

[19]:
```
Index(['job_aptitude_exam',     'same_industry',    'classes_missed',
                   'hs_gpa',       'job_offered',     'good_behavior',
              'high_school',          'illness',     'sensitive_680',
             'sensitive_128',     'sensitive_503',     'sensitive_572',
             'sensitive_607',     'sensitive_489',     'sensitive_335',
             'sensitive_343',     'sensitive_111',     'sensitive_455',
             'sensitive_249',     'sensitive_446',     'sensitive_271',
             'sensitive_663',     'sensitive_609',     'sensitive_151',
             'sensitive_467',                   0,                   1,
                           2,                   3,                   4,
                           5,                   6,                   7,
                           8,                   9,                  10,
                          11,                  12,                  13],
      dtype='object')
```

[20]:
```
#split the dataset
x_train, x_test, y_train, y_test = train_test_split(X,y,random_state=6,␣
 ↪test_size = 0.3)
#verify splits
print('total data size', len(X))
print('training data as a percentage of total data', round(len(x_train)/len(X)␣
 ↪*100,2))
print('testing data as a percentage of total data', round(len(x_test)/len(X)␣
 ↪*100,2))

print('\n\ntraining results as a percentage of total data', round(len(y_train)/
 ↪len(y) *100,2))
print('testing results as a percentage of total data', round(len(y_test)/len(y)␣
 ↪*100,2))
```

```
total data size 4764
training data as a percentage of total data 69.98
testing data as a percentage of total data 30.02


training results as a percentage of total data 69.98
testing results as a percentage of total data 30.02
```

[21]:
```
y_oversampled = df_oversampled['job_offered']
X_oversampled = df_oversampled.drop('job_offered', axis=1)
#split the dataset
x_train_oversampled, x_test_oversampled, y_train_oversampled,␣
 ↪y_test_oversampled =␣
 ↪train_test_split(X_oversampled,y_oversampled,random_state=6, test_size = 0.3)
#verify splits
print('total data size', len(X_oversampled))
```

```
print('training data as a percentage of total data',␣
 ↪round(len(x_train_oversampled)/len(X_oversampled) *100,2))
print('testing data as a percentage of total data',␣
 ↪round(len(x_test_oversampled)/len(X_oversampled) *100,2))

print('\n\ntraining results as a percentage of total data',␣
 ↪round(len(y_train_oversampled)/len(y_oversampled) *100,2))
print('testing results as a percentage of total data',␣
 ↪round(len(y_test_oversampled)/len(y_oversampled) *100,2))
```

```
total data size 8524
training data as a percentage of total data 69.99
testing data as a percentage of total data 30.01


training results as a percentage of total data 69.99
testing results as a percentage of total data 30.01
```

[22]:
```
y_oversampled2 = df_oversampled2['job_offered']
X_oversampled2 = df_oversampled2.drop('job_offered', axis=1)
#split the dataset
x_train_oversampled2, x_test_oversampled2, y_train_oversampled2,␣
 ↪y_test_oversampled2 =␣
 ↪train_test_split(X_oversampled2,y_oversampled2,random_state=6, test_size = 0.
 ↪3)
#verify splits
print('total data size', len(X_oversampled))
print('training data as a percentage of total data',␣
 ↪round(len(x_train_oversampled2)/len(X_oversampled2) *100,2))
print('testing data as a percentage of total data',␣
 ↪round(len(x_test_oversampled2)/len(X_oversampled2) *100,2))

print('\n\ntraining results as a percentage of total data',␣
 ↪round(len(y_train_oversampled2)/len(y_oversampled2) *100,2))
print('testing results as a percentage of total data',␣
 ↪round(len(y_test_oversampled2)/len(y_oversampled2) *100,2))
```

```
total data size 8524
training data as a percentage of total data 69.99
testing data as a percentage of total data 30.01


training results as a percentage of total data 69.99
testing results as a percentage of total data 30.01
```

2 Model Building

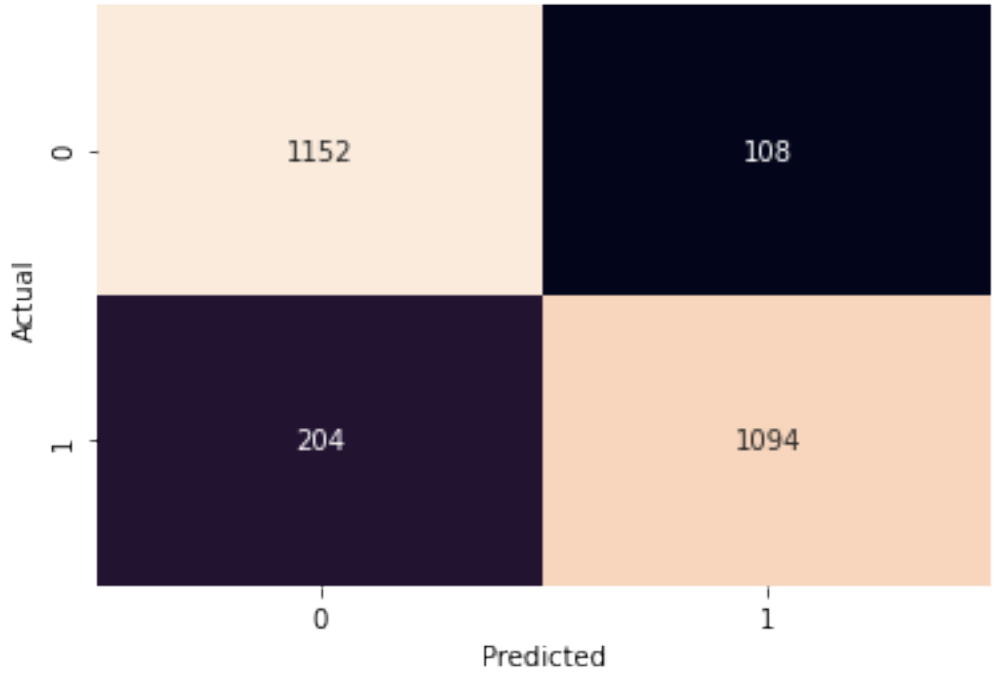
[23]:
```
def scoreModel(param_grid, classifier, x_train, x_test, y_train, y_test):
    pipe = Pipeline([('scaler', RobustScaler()),('pca', PCA()),␣
 ↪('classifier',classifier)])
    model = GridSearchCV(pipe, param_grid, cv=5, scoring='recall', n_jobs=8)
    model.fit(x_train, y_train)

    print("Best parameters (CV score=%0.3f):" % model.best_score_)
    print(model.best_params_)

    y_pred = model.predict(x_test)
    print_results(y_test, y_pred)

def print_results(y_test, y_pred):
    print(classification_report(y_test, y_pred), '\n')
    sns.heatmap(confusion_matrix(y_test, y_pred), cbar = False, annot = True,␣
 ↪fmt='.5g')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

def scoreModel_noPCA(param_grid, classifier, x_train, x_test, y_train, y_test):
    pipe = Pipeline([('scaler', RobustScaler()), ('classifier',classifier)])
    model = GridSearchCV(pipe, param_grid, cv=5, scoring='recall', n_jobs=8)
    model.fit(x_train, y_train)

    print("Best parameters (CV score=%0.3f):" % model.best_score_)
    print(model.best_params_)

    y_pred = model.predict(x_test)
    print_results(y_test, y_pred)
```

3 Vanilla Decision Tree

[24]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'classifier__max_features' : ['auto', 'sqrt', 'log2'],
    'classifier__max_depth': [None, 10, 50, 100, 150, 200, 250, 300],
}

regressor = DecisionTreeClassifier()
scoreModel_noPCA(param_grid, regressor, x_train_oversampled,␣
 ↪x_test_oversampled, y_train_oversampled, y_test_oversampled)
```

```
Best parameters (CV score=0.858):
```

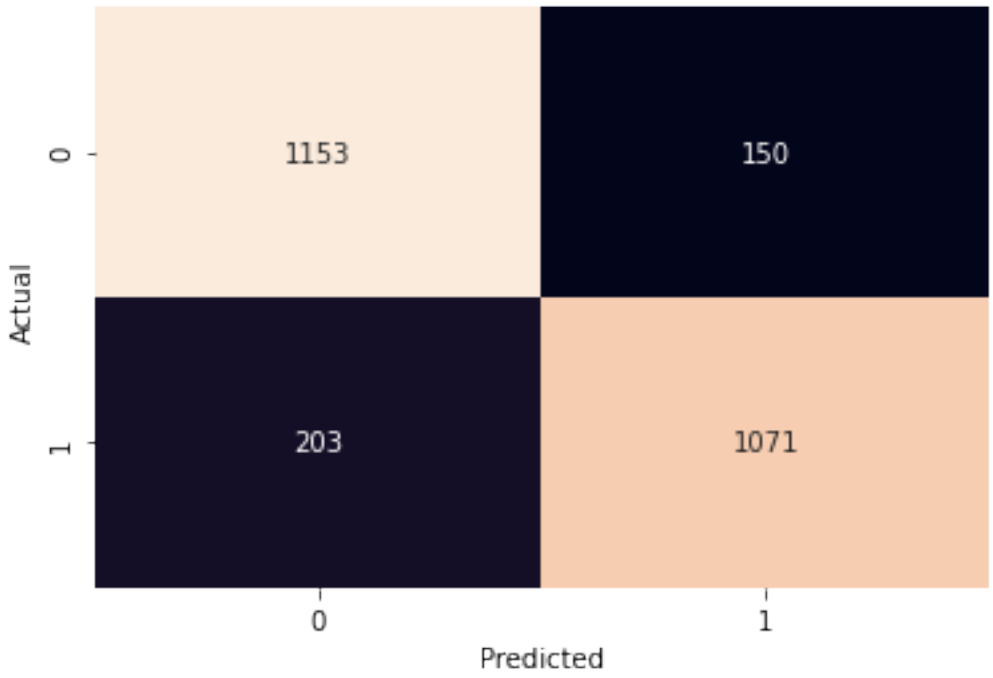
```
{'classifier__max_depth': 300, 'classifier__max_features': 'sqrt'}
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1260
           1       0.91      0.84      0.88      1298

    accuracy                           0.88      2558
   macro avg       0.88      0.88      0.88      2558
weighted avg       0.88      0.88      0.88      2558
```



[25]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'classifier__max_features' : ['auto', 'sqrt', 'log2'],
    'classifier__max_depth': [None, 10, 50, 100, 150, 200, 250, 300],
}

regressor = DecisionTreeClassifier()
scoreModel_noPCA(param_grid, regressor, x_train_oversampled2,␣
 ↪x_test_oversampled2, y_train_oversampled2, y_test_oversampled2)
```

```
Best parameters (CV score=0.833):
{'classifier__max_depth': 250, 'classifier__max_features': 'sqrt'}
```

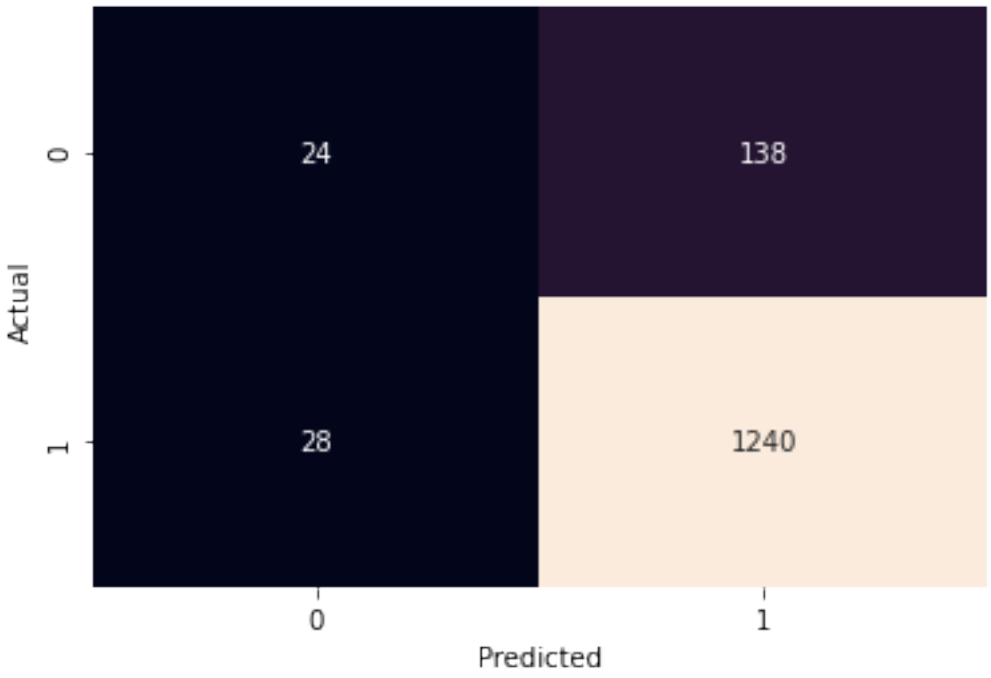
```
                precision    recall  f1-score   support

           0       0.85      0.88      0.87      1303
           1       0.88      0.84      0.86      1274

    accuracy                           0.86      2577
   macro avg       0.86      0.86      0.86      2577
weighted avg       0.86      0.86      0.86      2577
```



[26]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'classifier__max_features' : ['auto', 'sqrt', 'log2'],
    'classifier__max_depth': [None, 10, 50, 100, 150, 200, 250, 300],
}

regressor = DecisionTreeClassifier()
scoreModel_noPCA(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.973):
{'classifier__max_depth': 10, 'classifier__max_features': 'log2'}
              precision    recall  f1-score   support
```

```
              0       0.46      0.15      0.22       162
              1       0.90      0.98      0.94      1268

    accuracy                            0.88      1430
   macro avg        0.68      0.56      0.58      1430
weighted avg        0.85      0.88      0.86      1430
```

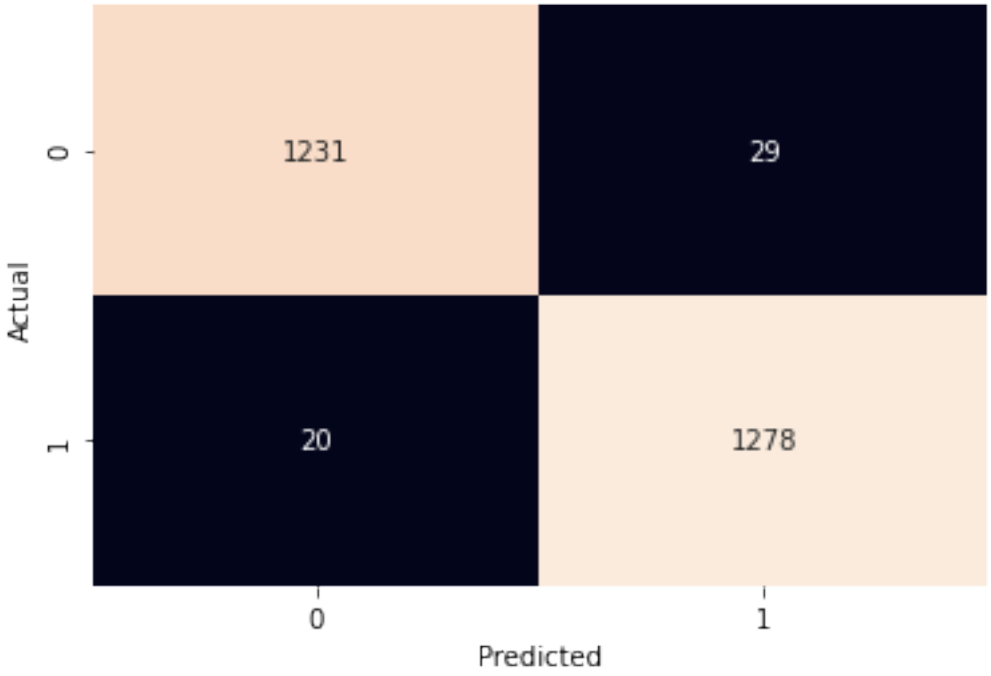


4 RandomForest

[27]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'classifier__max_features' : ['auto', 'sqrt', 'log2'],
    'classifier__max_depth': [None, 10, 50, 100, 150, 200],
    'classifier__n_estimators':[150, 200, 250, 300],
}

regressor = RandomForestClassifier()
scoreModel_noPCA(param_grid, regressor, x_train_oversampled,␣
 ↪x_test_oversampled, y_train_oversampled, y_test_oversampled)
```

```
Best parameters (CV score=0.978):
{'classifier__max_depth': None, 'classifier__max_features': 'log2',
```

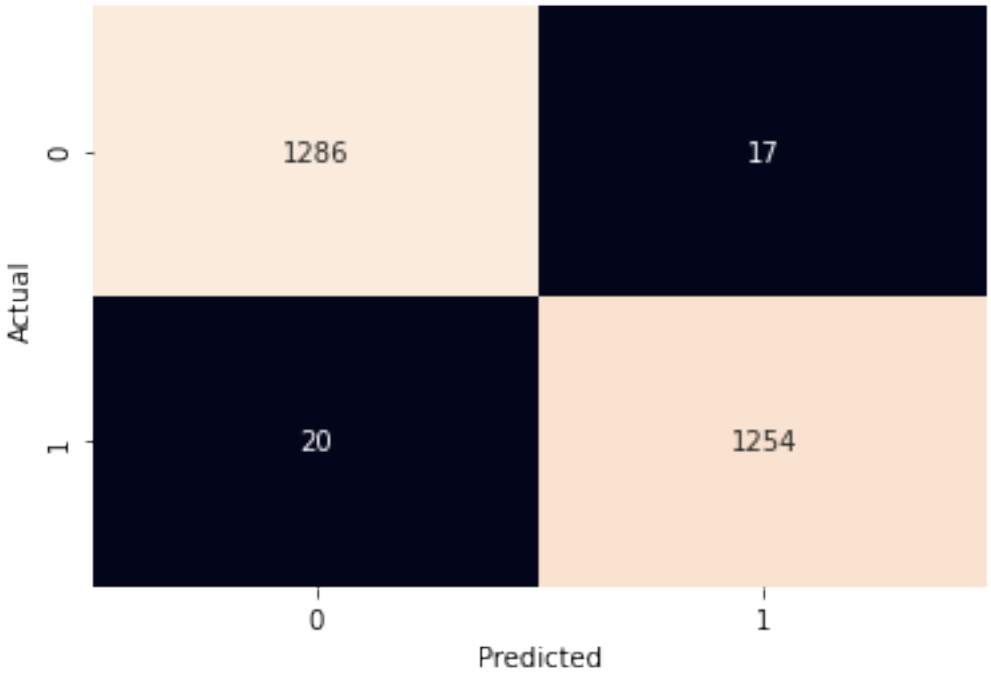
```
'classifier__n_estimators': 250}
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1260
           1       0.98      0.98      0.98      1298

    accuracy                           0.98      2558
   macro avg       0.98      0.98      0.98      2558
weighted avg       0.98      0.98      0.98      2558
```



[28]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'classifier__max_features' : ['auto', 'sqrt', 'log2'],
    'classifier__max_depth': [None, 10, 50, 100, 150, 200],
    'classifier__n_estimators':[150, 200, 250, 300, 350, 400],
}

regressor = RandomForestClassifier()
scoreModel_noPCA(param_grid, regressor, x_train_oversampled2,␣
 ↪x_test_oversampled2, y_train_oversampled2, y_test_oversampled2)
```

```
Best parameters (CV score=0.979):
```

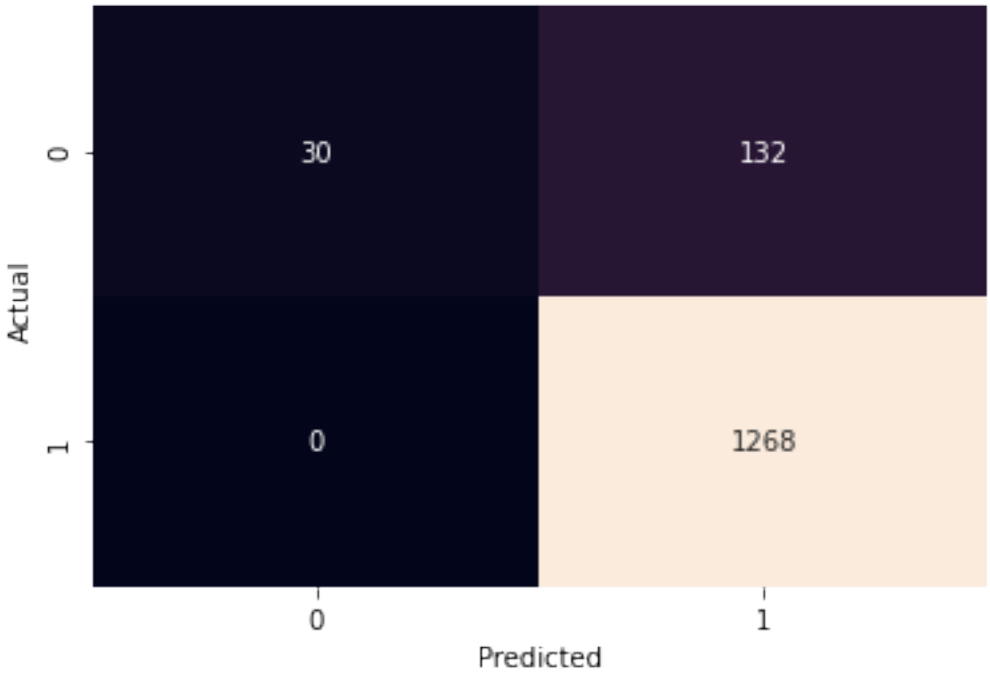
```
{'classifier__max_depth': 100, 'classifier__max_features': 'log2',
'classifier__n_estimators': 350}
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1303
           1       0.99      0.98      0.99      1274

    accuracy                           0.99      2577
   macro avg       0.99      0.99      0.99      2577
weighted avg       0.99      0.99      0.99      2577
```



[29]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'classifier__max_features' : ['auto', 'sqrt', 'log2'],
    'classifier__max_depth': [None, 10, 50, 100, 150, 200],
    'classifier__n_estimators':[150, 200, 250, 300],
}

regressor = RandomForestClassifier()
scoreModel_noPCA(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=1.000):
```

```
{'classifier__max_depth': None, 'classifier__max_features': 'auto',
'classifier__n_estimators': 200}
              precision    recall  f1-score   support

           0       1.00      0.19      0.31       162
           1       0.91      1.00      0.95      1268

    accuracy                           0.91      1430
   macro avg       0.95      0.59      0.63      1430
weighted avg       0.92      0.91      0.88      1430
```

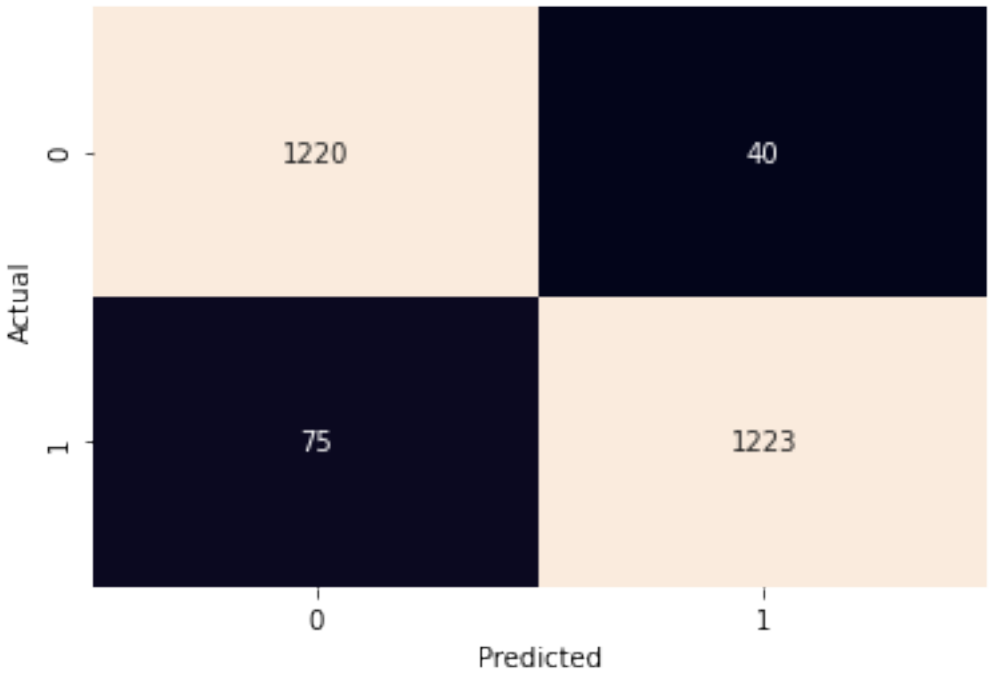


5 GradientBoosting

[30]:
```
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    #'pca__n_components': [5, 8, 9, 10, None],
    'classifier__loss':['deviance', 'exponential'],
    'classifier__n_estimators':[300, 350, 400, 450, 500],
    'classifier__criterion':['mse', 'friedman_mse'],
}

classifier = GradientBoostingClassifier()
```

```
scoreModel_noPCA(param_grid, classifier, x_train_oversampled,␣
 ↪x_test_oversampled, y_train_oversampled, y_test_oversampled)
```

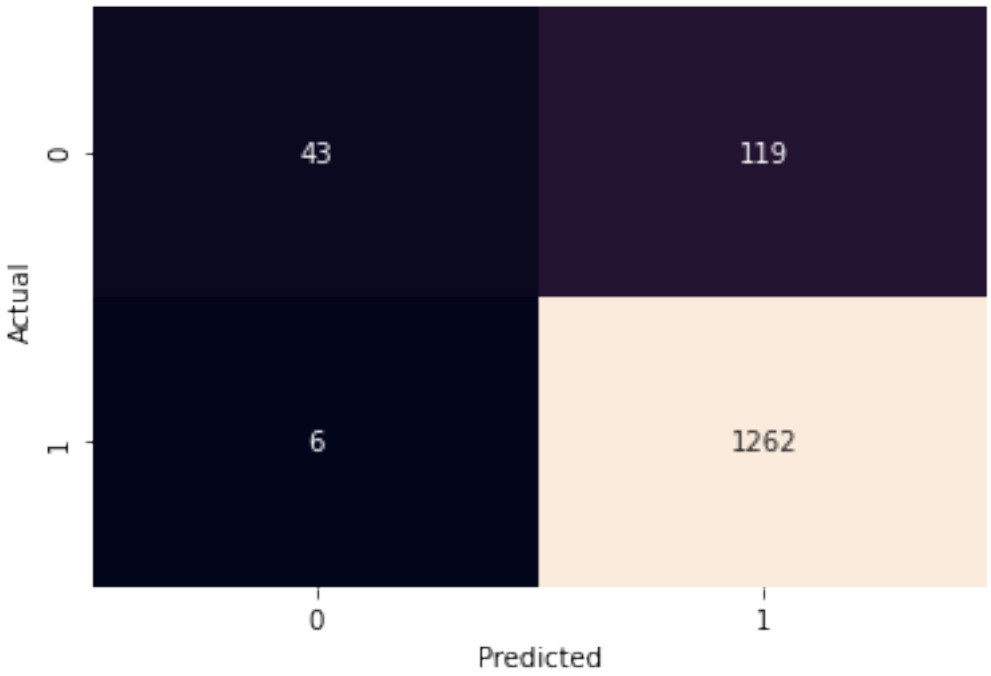
```
Best parameters (CV score=0.926):
{'classifier__criterion': 'mse', 'classifier__loss': 'deviance',
'classifier__n_estimators': 500}
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      1260
           1       0.97      0.94      0.96      1298

    accuracy                           0.96      2558
   macro avg       0.96      0.96      0.96      2558
weighted avg       0.96      0.96      0.96      2558
```



[31]:
```
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    #'pca__n_components': [5, 8, 9, 10, None],
    'classifier__loss':['deviance', 'exponential'],
    'classifier__n_estimators':[200, 250, 300, 350, 400],
    'classifier__criterion':['mse', 'friedman_mse'],
}
```

```
classifier = GradientBoostingClassifier()
scoreModel_noPCA(param_grid, classifier, x_train, x_test, y_train, y_test)
```

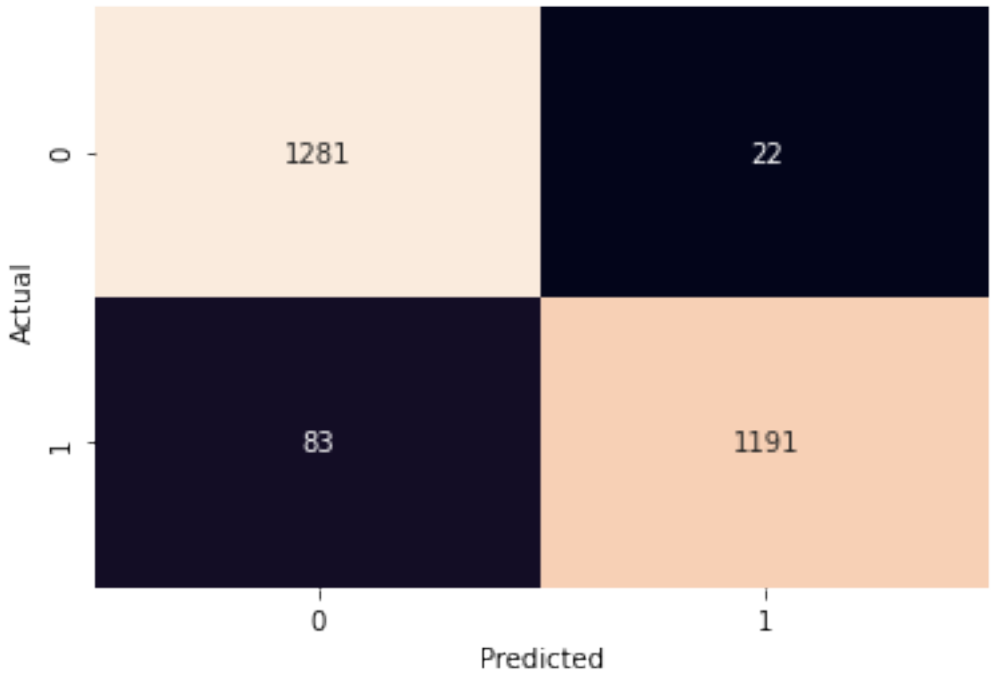
```
Best parameters (CV score=0.996):
{'classifier__criterion': 'mse', 'classifier__loss': 'exponential',
'classifier__n_estimators': 200}
              precision    recall  f1-score   support

           0       0.88      0.27      0.41       162
           1       0.91      1.00      0.95      1268

    accuracy                           0.91      1430
   macro avg       0.90      0.63      0.68      1430
weighted avg       0.91      0.91      0.89      1430
```



[32]:
```
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    #'pca__n_components': [5, 8, 9, 10, None],
    'classifier__loss':['deviance', 'exponential'],
    'classifier__n_estimators':[200, 250, 300, 350, 400],
    'classifier__criterion':['mse', 'friedman_mse'],
}
```

```
classifier = GradientBoostingClassifier()
scoreModel_noPCA(param_grid, classifier,x_train_oversampled2,␣
↪x_test_oversampled2, y_train_oversampled2, y_test_oversampled2)
```

```
Best parameters (CV score=0.914):
{'classifier__criterion': 'mse', 'classifier__loss': 'deviance',
'classifier__n_estimators': 400}
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      1303
           1       0.98      0.93      0.96      1274

    accuracy                           0.96      2577
   macro avg       0.96      0.96      0.96      2577
weighted avg       0.96      0.96      0.96      2577
```

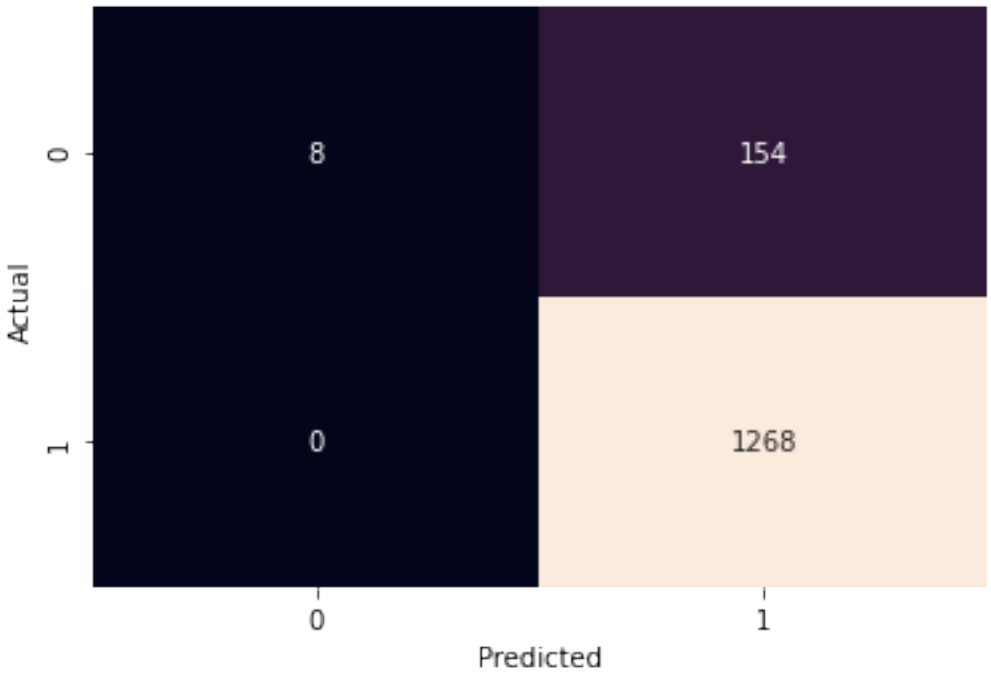


6 XGBoost

[33]:
```
param_grid = {"classifier__learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.
↪30 ] ,
              "classifier__max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
              "classifier__min_child_weight" : [ 1, 3, 5, 7 ],
```

```
        "classifier__gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
        "classifier__colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ],
}

classifier = XGBClassifier(eval_metric='logloss', use_label_encoder=False)
scoreModel_noPCA(param_grid, classifier, x_train, x_test, y_train, y_test)
```

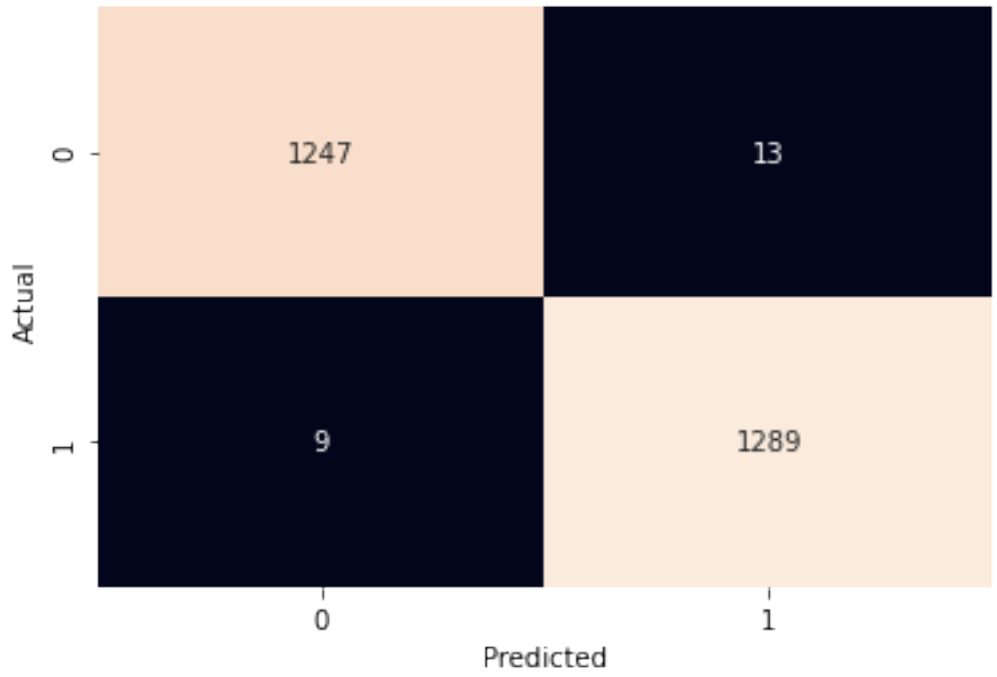
```
Best parameters (CV score=1.000):
{'classifier__colsample_bytree': 0.3, 'classifier__gamma': 0.0,
'classifier__learning_rate': 0.05, 'classifier__max_depth': 3,
'classifier__min_child_weight': 1}
              precision    recall  f1-score   support

           0       1.00      0.05      0.09       162
           1       0.89      1.00      0.94      1268

    accuracy                           0.89      1430
   macro avg       0.95      0.52      0.52      1430
weighted avg       0.90      0.89      0.85      1430
```



[34]:
```
param_grid = {"classifier__learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.
↪30 ] ,
        "classifier__max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
```

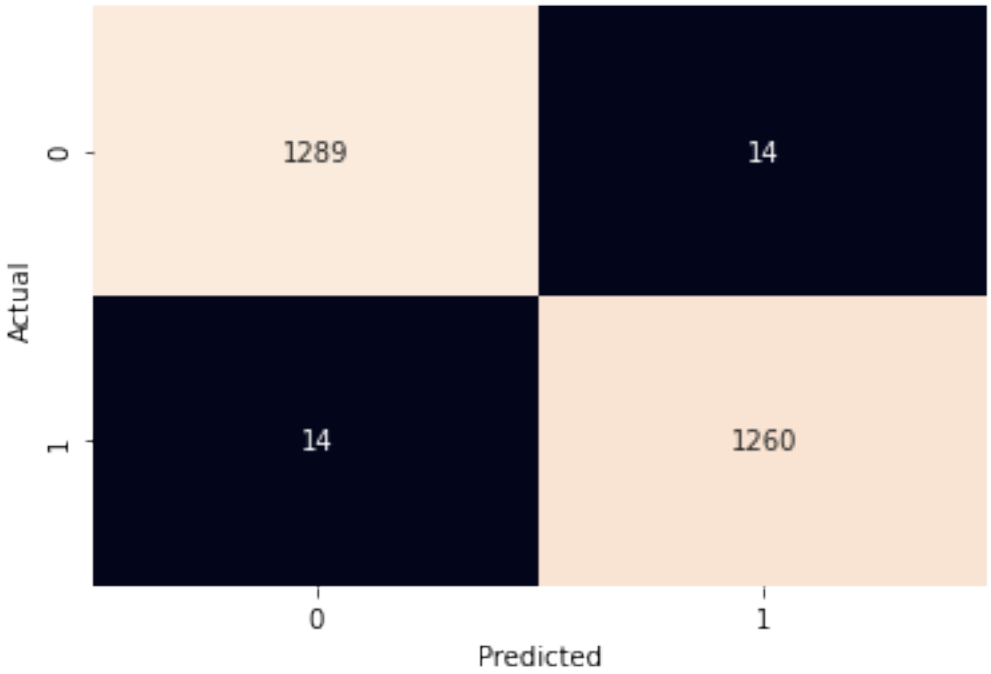
```
        "classifier__min_child_weight" : [ 1, 3, 5, 7 ],
        "classifier__gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
        "classifier__colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ],
}

classifier = XGBClassifier(eval_metric='logloss', use_label_encoder=False)
scoreModel_noPCA(param_grid, classifier, x_train_oversampled,␣
 ↪x_test_oversampled, y_train_oversampled, y_test_oversampled)
```

```
Best parameters (CV score=0.985):
{'classifier__colsample_bytree': 0.3, 'classifier__gamma': 0.0,
'classifier__learning_rate': 0.25, 'classifier__max_depth': 15,
'classifier__min_child_weight': 1}
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1260
           1       0.99      0.99      0.99      1298

    accuracy                           0.99      2558
   macro avg       0.99      0.99      0.99      2558
weighted avg       0.99      0.99      0.99      2558
```



[35]:
```
param_grid = {"classifier__learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.
 ↪30 ] ,
        "classifier__max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
        "classifier__min_child_weight" : [ 1, 3, 5, 7 ],
        "classifier__gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
        "classifier__colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ],
}

classifier = XGBClassifier(eval_metric='logloss', use_label_encoder=False)
scoreModel_noPCA(param_grid, classifier, x_train_oversampled2,␣
 ↪x_test_oversampled2, y_train_oversampled2, y_test_oversampled2)
```

```
Best parameters (CV score=0.984):
{'classifier__colsample_bytree': 0.3, 'classifier__gamma': 0.0,
'classifier__learning_rate': 0.15, 'classifier__max_depth': 15,
'classifier__min_child_weight': 1}
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1303
           1       0.99      0.99      0.99      1274

    accuracy                           0.99      2577
   macro avg       0.99      0.99      0.99      2577
weighted avg       0.99      0.99      0.99      2577
```



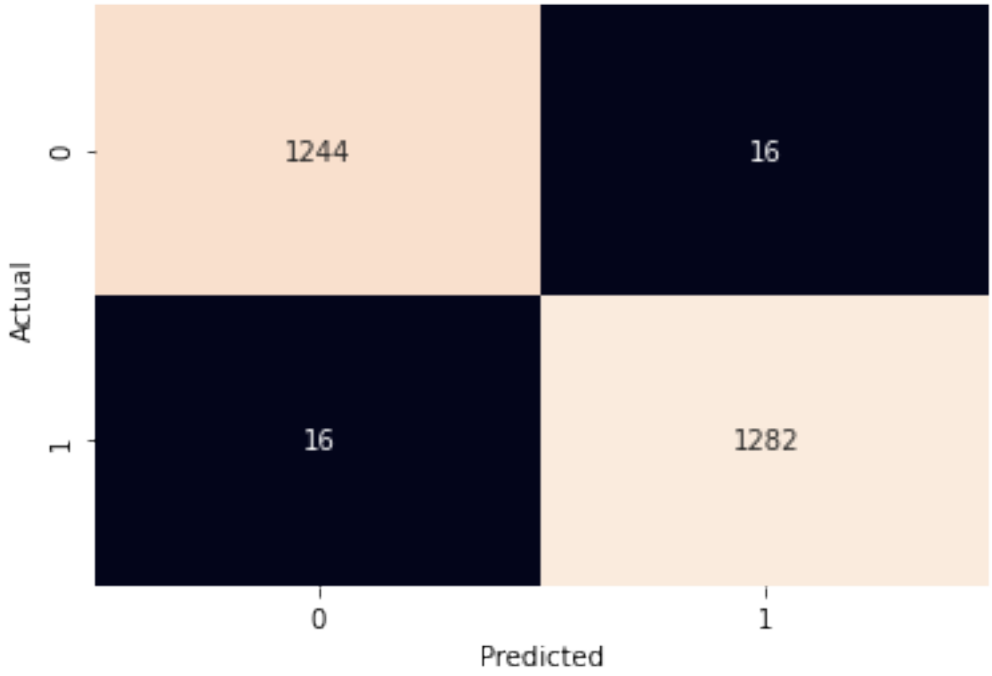
7 Best performing models

[46]:
```
output = pd.DataFrame()

classifier = RandomForestClassifier(max_depth= 100, max_features= 'log2',␣
 ↪n_estimators= 350)
model = Pipeline([('scaler', RobustScaler()), ('classifier',classifier)])
model.fit(x_train_oversampled2,y_train_oversampled2)

#compute model weights to assess contributions from each feature
feature_weights = model.steps[1][1].feature_importances_
output['feature_rating'] =  feature_weights[0:]
output['features'] = x_train.columns
print(output.sort_values(by=['feature_rating'], ascending = False,␣
 ↪ignore_index=True))
```

```
    feature_rating           features
0         0.083367        high_school
1         0.083154      sensitive_489
2         0.074284      sensitive_271
3         0.057692      sensitive_343
4         0.047316      sensitive_663
5         0.042505      sensitive_455
6         0.041814      sensitive_572
7         0.041766      sensitive_249
8         0.040516  job_aptitude_exam
9         0.039925      sensitive_111
10        0.039153      sensitive_151
11        0.039104      sensitive_503
12        0.037915             hs_gpa
13        0.037257      sensitive_609
14        0.037202      sensitive_335
15        0.037104      sensitive_467
16        0.036802      sensitive_607
17        0.035887      sensitive_128
18        0.033895         instructor
19        0.033638      sensitive_446
20        0.033532      sensitive_680
21        0.023302     classes_missed
22        0.018855      good_behavior
23        0.002839      same_industry
24        0.001176            illness
```

[47]:
```
y_pred = model.predict(x_test_oversampled)
print_results(y_test_oversampled, y_pred)
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1260
1	0.99	0.99	0.99	1298
accuracy			0.99	2558
macro avg	0.99	0.99	0.99	2558
weighted avg	0.99	0.99	0.99	2558

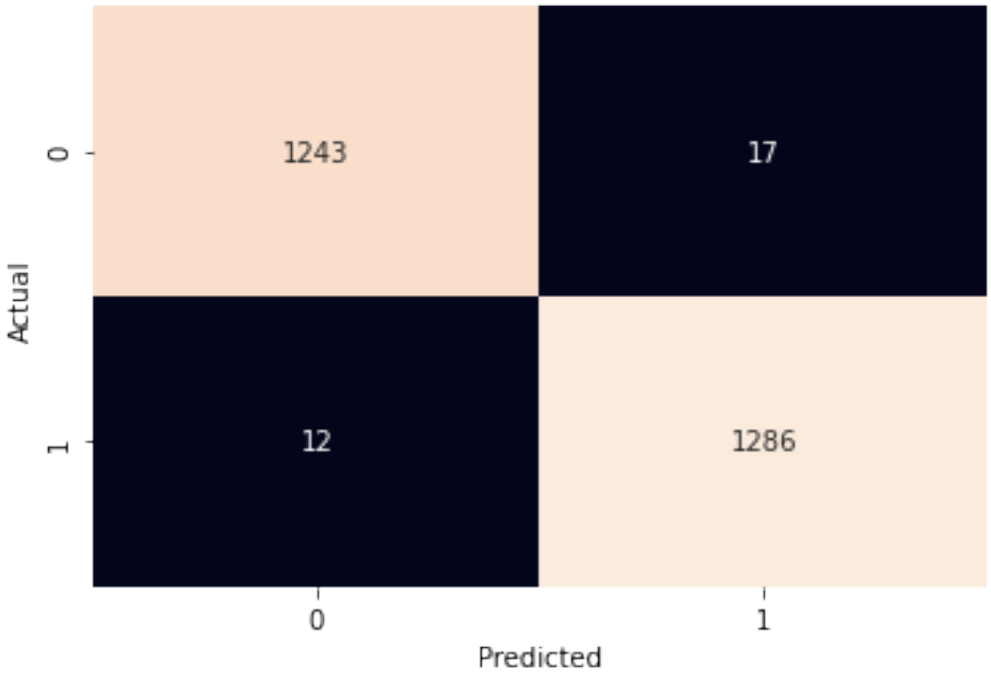


[48]:
```
output = pd.DataFrame()

classifier = XGBClassifier(eval_metric='logloss', use_label_encoder=False,␣
 ↪colsample_bytree = 0.3, gamma= 0.0, learning_rate = 0.15, max_depth = 15,␣
 ↪min_child_weight = 1)
model = Pipeline([('scaler', RobustScaler()), ('classifier',classifier)])
model.fit(x_train_oversampled2,y_train_oversampled2)

#compute model weights to assess contributions from each feature
feature_weights = model.steps[1][1].feature_importances_
output['feature_rating'] =  feature_weights[0:]
output['features'] = x_train.columns
print(output.sort_values(by=['feature_rating'], ascending = False,␣
 ↪ignore_index=True))
```

	feature_rating	features
0	0.095262	good_behavior
1	0.077959	high_school
2	0.071415	sensitive_343
3	0.066671	illness
4	0.051929	instructor
5	0.047519	sensitive_271
6	0.045290	sensitive_151
7	0.040660	sensitive_111
8	0.038793	sensitive_335
9	0.038104	sensitive_609
10	0.037474	sensitive_446
11	0.034472	sensitive_607
12	0.034446	job_aptitude_exam
13	0.031266	sensitive_503
14	0.030492	sensitive_489
15	0.030317	hs_gpa
16	0.028777	sensitive_467
17	0.027913	same_industry
18	0.027288	sensitive_572
19	0.026922	sensitive_455
20	0.026735	sensitive_663
21	0.026678	sensitive_128
22	0.024671	sensitive_249
23	0.021303	classes_missed
24	0.017643	sensitive_680

[49]:
```
y_pred = model.predict(x_test_oversampled)
print_results(y_test_oversampled, y_pred)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1260
           1       0.99      0.99      0.99      1298

    accuracy                           0.99      2558
   macro avg       0.99      0.99      0.99      2558
weighted avg       0.99      0.99      0.99      2558
```



8 Model Discussion

With oversampling, both XGBoost and RandomForest classifiers preformed admirably. Each acheived accuracy in the 99% range with precision, recall and f1 scores all greater than or equal to 98%. XGBoost is a more robust boosting algo, which takes time to properly tune and train. RandomForest is a bagging algo, which is leaner and much easier to train. Clearly, both algos are suited to this task, but XGBoost a bit more robust.

[ ]:

9 Open Discussion Section

(1) Let's say a client comes to you and asks for "the effect of attendance (one of the variables in the dataset) on being offered a job." What are some ways you might go about providing that? How would you communicate uncertainty around the 'effect size'?

There is some inverse correlation between classes missed and hs_gpa (-0.34) (unlikely to cause issues in building a model). Simply put, it appears attendance and gpa are mildly related, better attendance leads to better grades. The effect size is related to the magnitude of correlation, mild(0.2-0.4), medium(0.4-0.75), strong(0.75-1.0). Attendance and hs_gpa have a mild association

(2) Let's say your client is interested in understanding the uncertainty of the predictions produced by your model at the individual level. As an example, your model might say that person i has probability of being offered a job 0.78. How do you calculate uncertainty on that quantity?

In a classification models performace can be computed in different ways, depending on the specific needs of the client. The most common are overall accuracy, precision, recall, and f1 scores.

Accuracy represents the models overall accuracy accuracy = correct_predictions/total_predictions.

recall represents the models ability to capture the signal in question. recall = true_positives/(true_positives + false_negatives)

precision represents how accurately the model is able to identify the target variable in question. precision = true_positives/(true_positives + false_positives)

f1 score is the harmonic of precision and recall. f1 = 2*precision*recall/(precision+recall)

Given the wording of the question, the client probably cares how precise the model is in classifying the probabilty at .78. Therefore, precision would be the best metric to provide the client.

(3) Let's say a domain expert thinks that the model is not performing well for a subset of the population (e.g. folks with a low GPA). How would you check to see if the model is performing well among subpopulations of your training data?

a. Ideally, I would collect data from the population subset in question, input it into the model, then compute performace metrics. This type of experimentation would allow me to 1. identify model performace for the group in question. 2. it would provide me with additional training data to update and improve model performance for the group in question.

b. If collecting additional data is not possible, I would identify records from the testing data for the subset in question, input it into the model, then compute performance metrics.

c. If a and b are not possible, then I would try and get the data which the expert is using, input it into the model and then compute performance metrics.

(4) Let's say you discover that the training data provided here are a biased subset of all of the data in the actual jobs program (you were only able to get data in one geography, for example, or the data you were able to acquire tends to be folks with higher GPAs). Let's also say you are able to acquire the features for the population at large (but you don't have the outcome variable), and your new task is to provide prediction on the entire population and not your biased subset. What, if anything, would you change about your analysis?

a. If I knew the source of the bias (say high GPA), I could oversample the under represented class (low GPA) in an attempt to balance the data accordingly. This may bring the model preformance back in line on the unseen data.

b. I could perform clustering analysis on the whole dataset. if a clear pattern emerged from clustering, I could then use the clustering information to comment on the unseen data. Including, possibly even tuning the model to correctly predict the under represented classes.

[ ]: