Predict occurrence of kidney stones using Classification Algorithms

We eventually want to predict some target based on numerous inputs . In this situation, we wish to predict the presence of Kidney Stones, where '1 = we observe a presence' and '0 = we detect no presence.' As a result, this is a 'Binary Classification' issue in which something exists or it does not.

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBo
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold,
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow import keras
import pickle
# NN models
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.wrappers.scikit_learn import KerasClassifier
from keras.callbacks import EarlyStopping, ModelCheckpoint
import warnings
warnings.filterwarnings('ignore')
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/playground-series-s3e12/sample_submission.csv
/kaggle/input/playground-series-s3e12/train.csv
/kaggle/input/playground-series-s3e12/test.csv
```

Loading the csv file

In [2]:
```
df=pd.read_csv('/kaggle/input/playground-series-s3e12/train.csv')
```

In [3]:
```
df
```

Out[3]:

	id	gravity	ph	osmo	cond	urea	calc	target
0	0	1.013	6.19	443	14.8	124	1.45	0
1	1	1.025	5.40	703	23.6	394	4.18	0
2	2	1.009	6.13	371	24.5	159	9.04	0
3	3	1.021	4.91	442	20.8	398	6.63	1
4	4	1.021	5.53	874	17.8	385	2.21	1
...	...	...	...	...	...	...	...	...
409	409	1.011	5.21	527	21.4	75	1.53	0
410	410	1.024	5.53	577	19.7	224	0.77	0
411	411	1.018	6.28	455	22.2	270	7.68	1
412	412	1.008	7.12	325	12.6	75	1.03	1
413	413	1.011	6.13	364	9.9	159	0.27	0

414 rows × 8 columns

Data Preprocessing

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   id       414 non-null    int64  
 1   gravity  414 non-null    float64
 2   ph       414 non-null    float64
 3   osmo     414 non-null    int64  
 4   cond     414 non-null    float64
 5   urea     414 non-null    int64  
 6   calc     414 non-null    float64
 7   target   414 non-null    int64  
dtypes: float64(4), int64(4)
memory usage: 26.0 KB
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
id         0
gravity    0
ph         0
osmo       0
cond       0
urea       0
calc       0
target     0
dtype: int64
```

In [6]:
```
df.duplicated().sum()
```

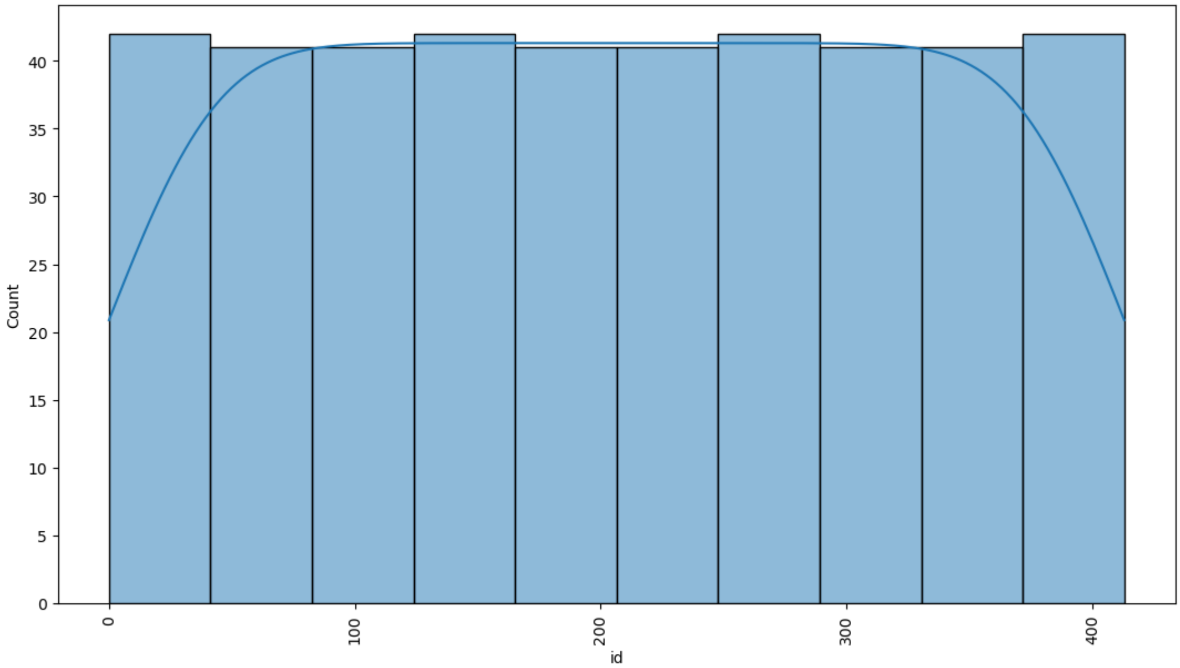
Out[6]: 0

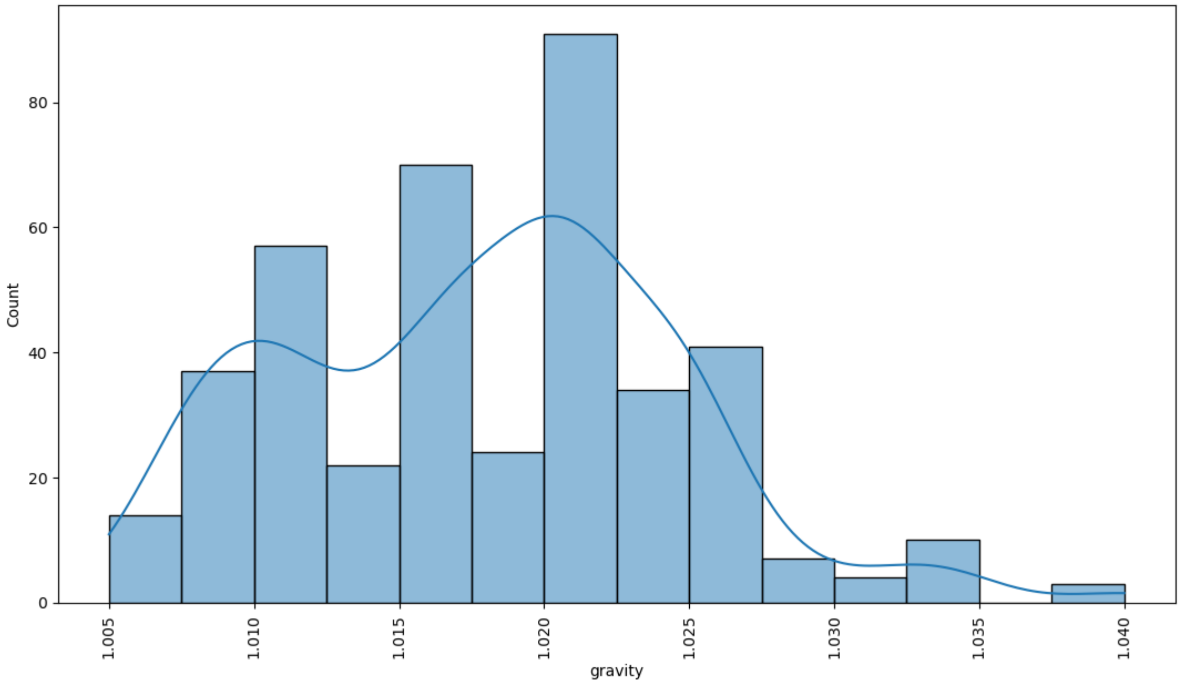
In [7]:
```
df['target'].value_counts()
```

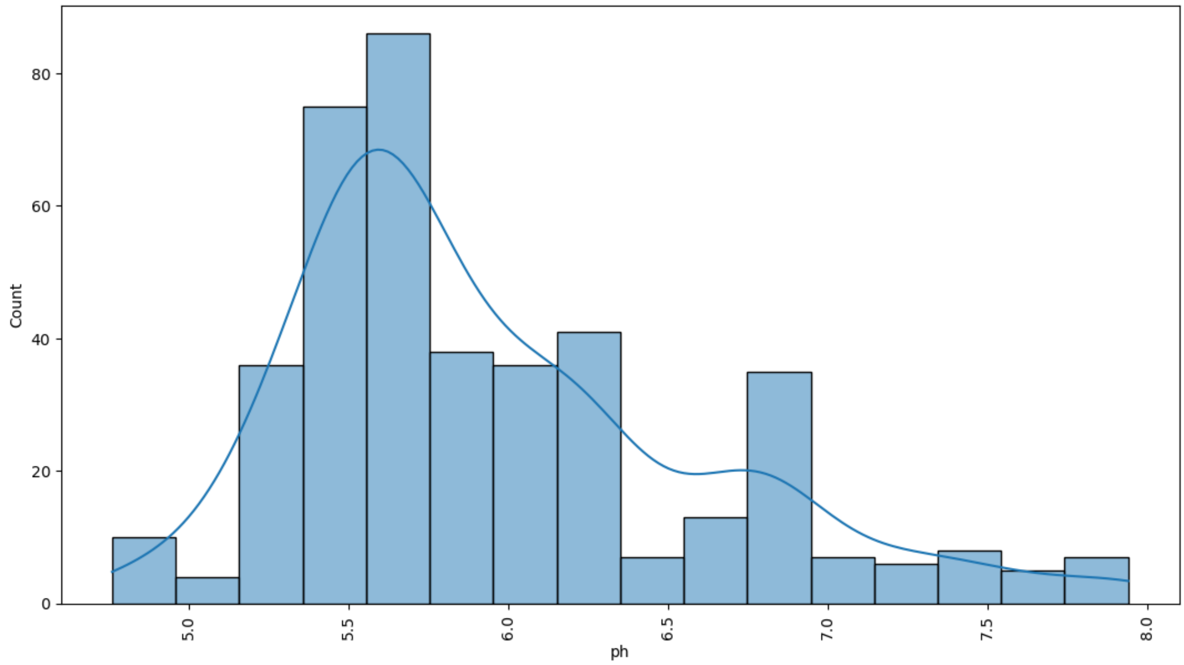
Out[7]:
```
0    230
1    184
Name: target, dtype: int64
```

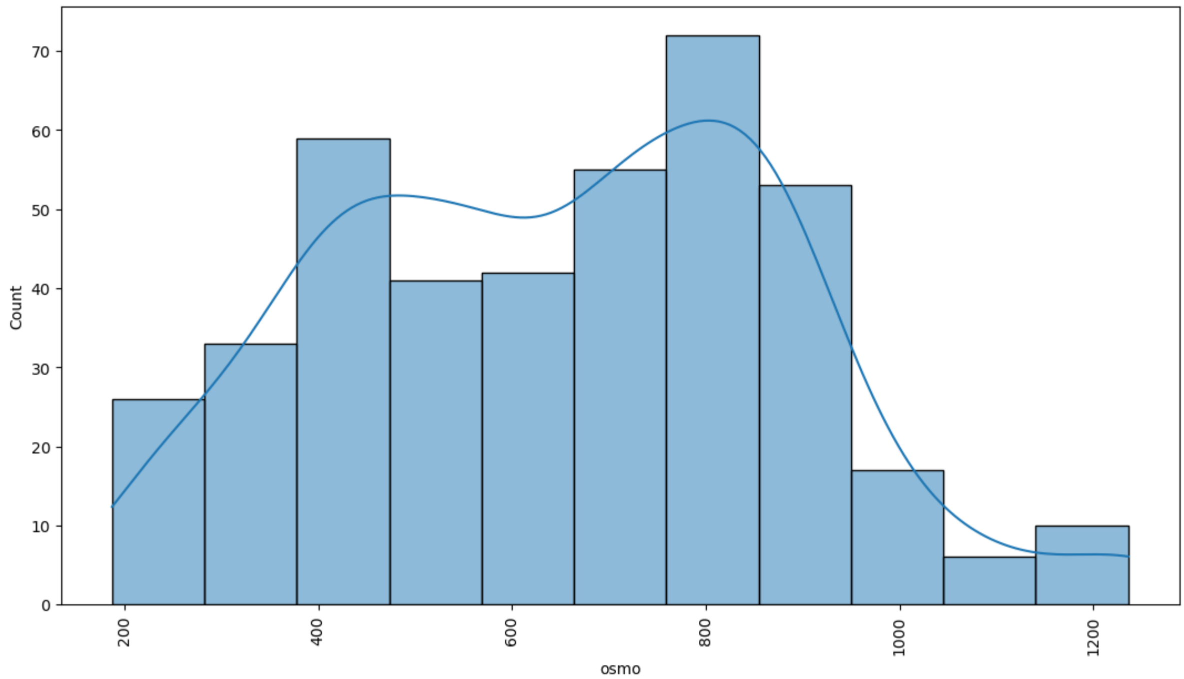
Data Visualization

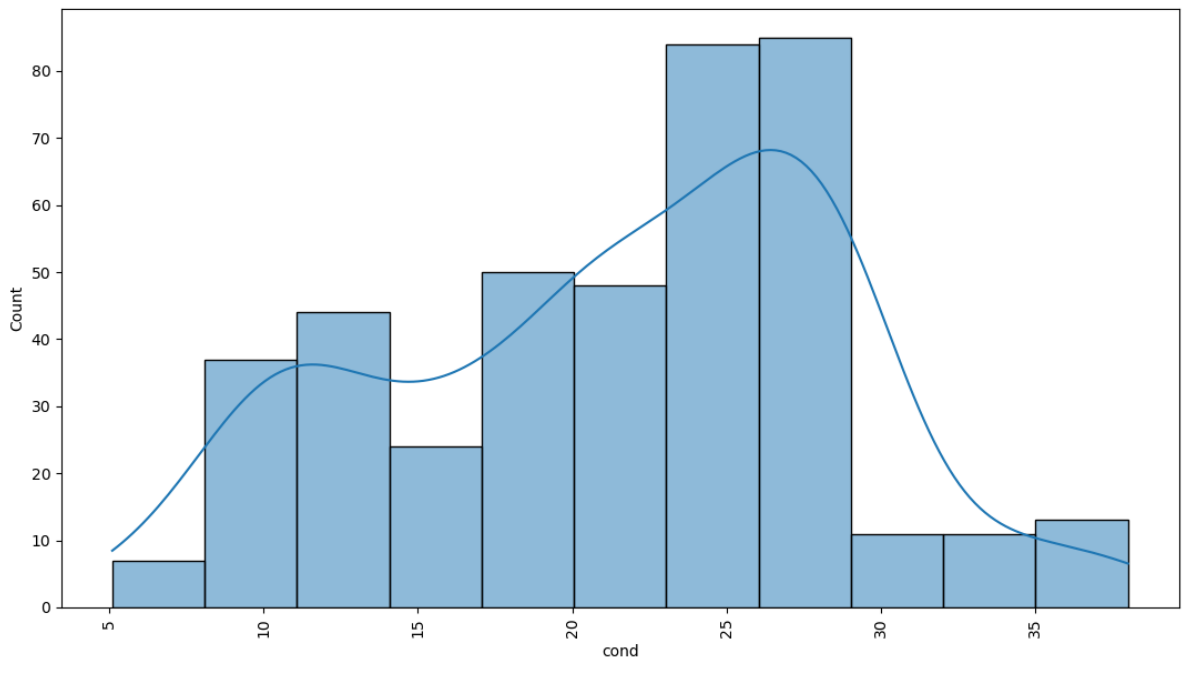
In [8]:
```
for i in df.columns:
    plt.figure(figsize=(13,7))
    sns.histplot(data = df[i], kde=True, multiple='stack')
    plt.xticks(rotation=90)
    plt.show()
```

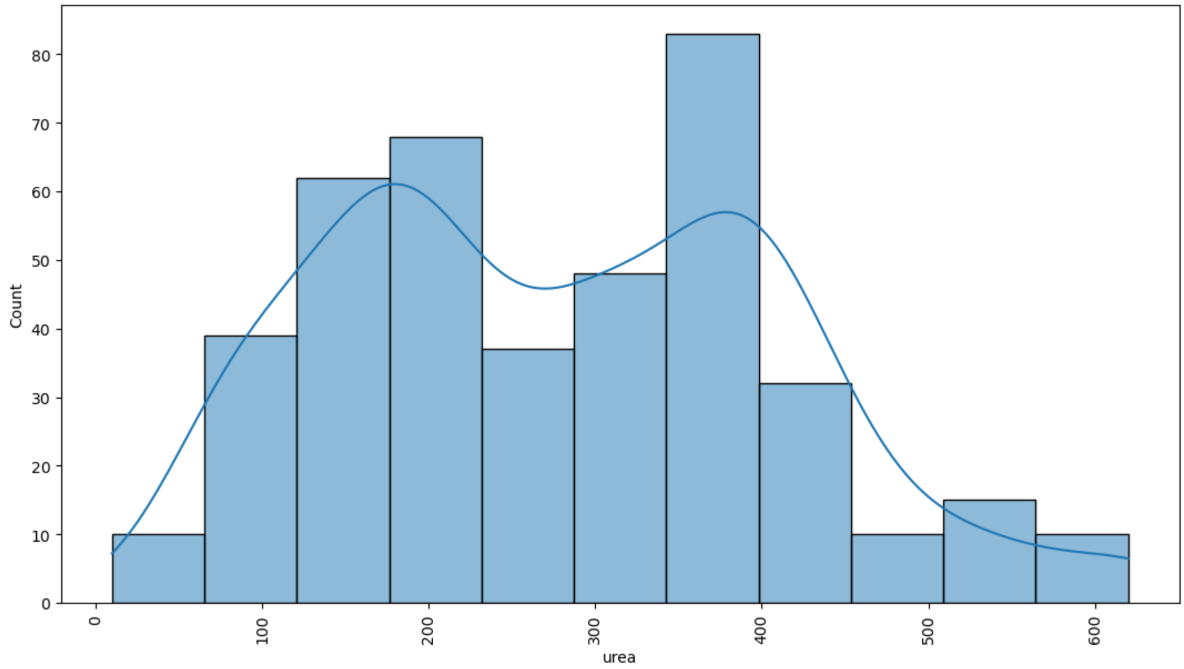


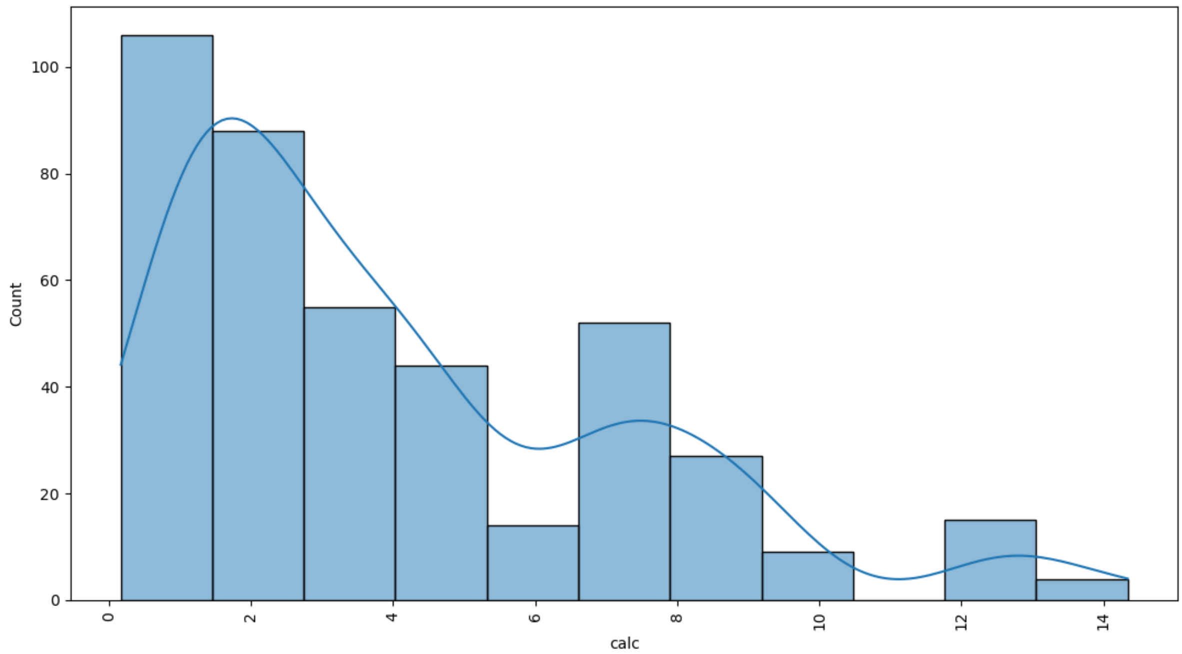


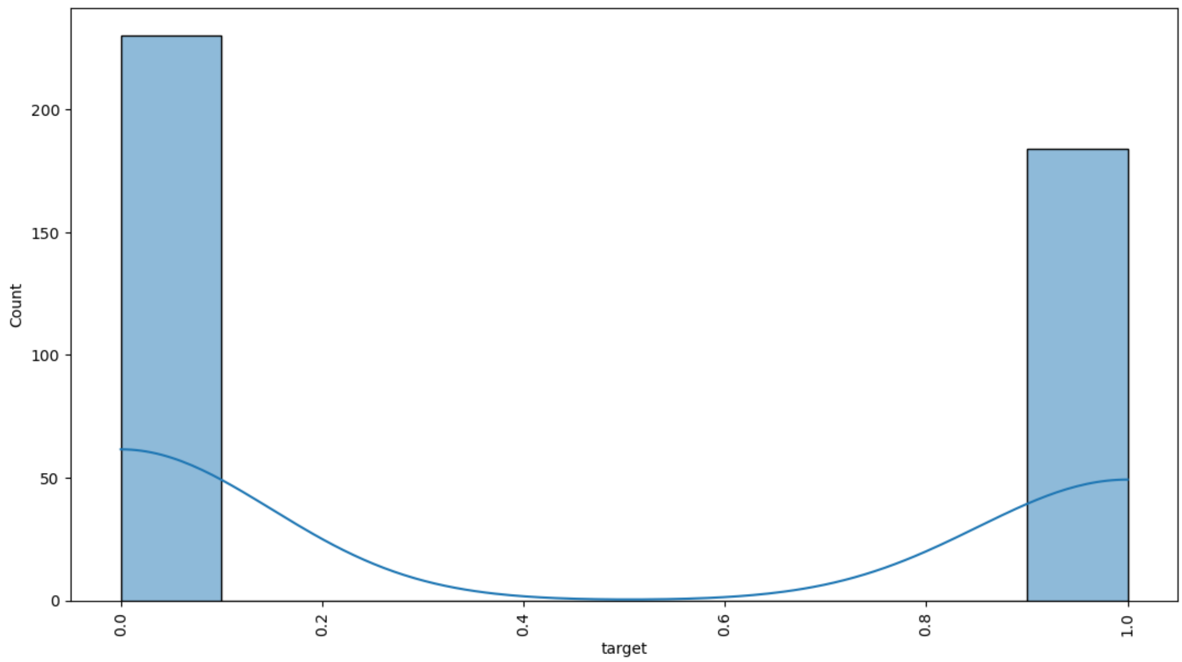

Count
0
20
40
60
80
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ph


Count
0
10
20
30
40
50
60
70
200
400
600
800
1000
1200
osmo


Count
0
10
20
30
40
50
60
70
80
5
10
15
20
25
30
35
cond

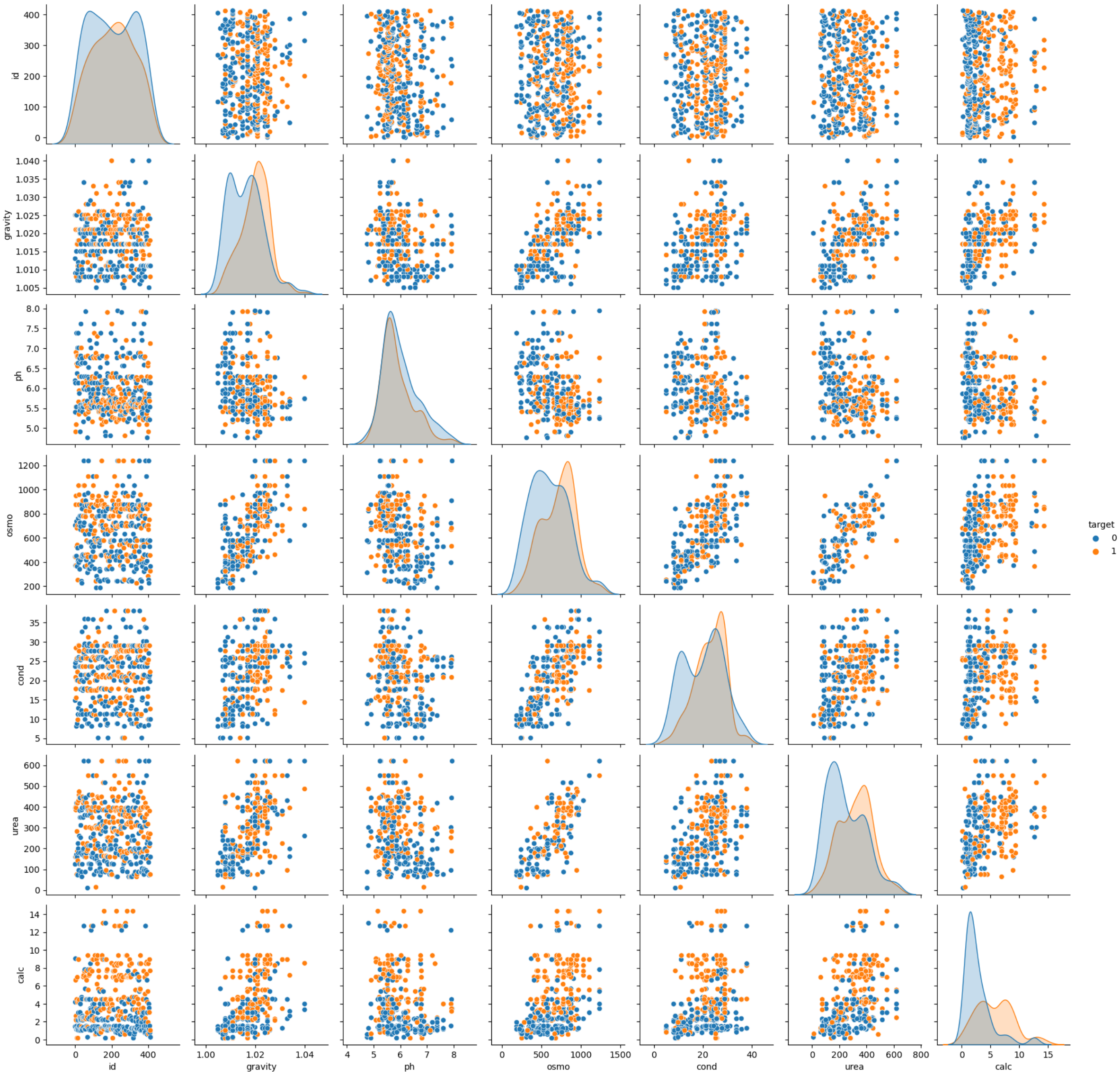

Count
urea
0
10
20
30
40
50
60
70
80
100
200
300
400
500
600


Count
calc
0
20
40
60
80
100
2
4
6
8
10
12
14

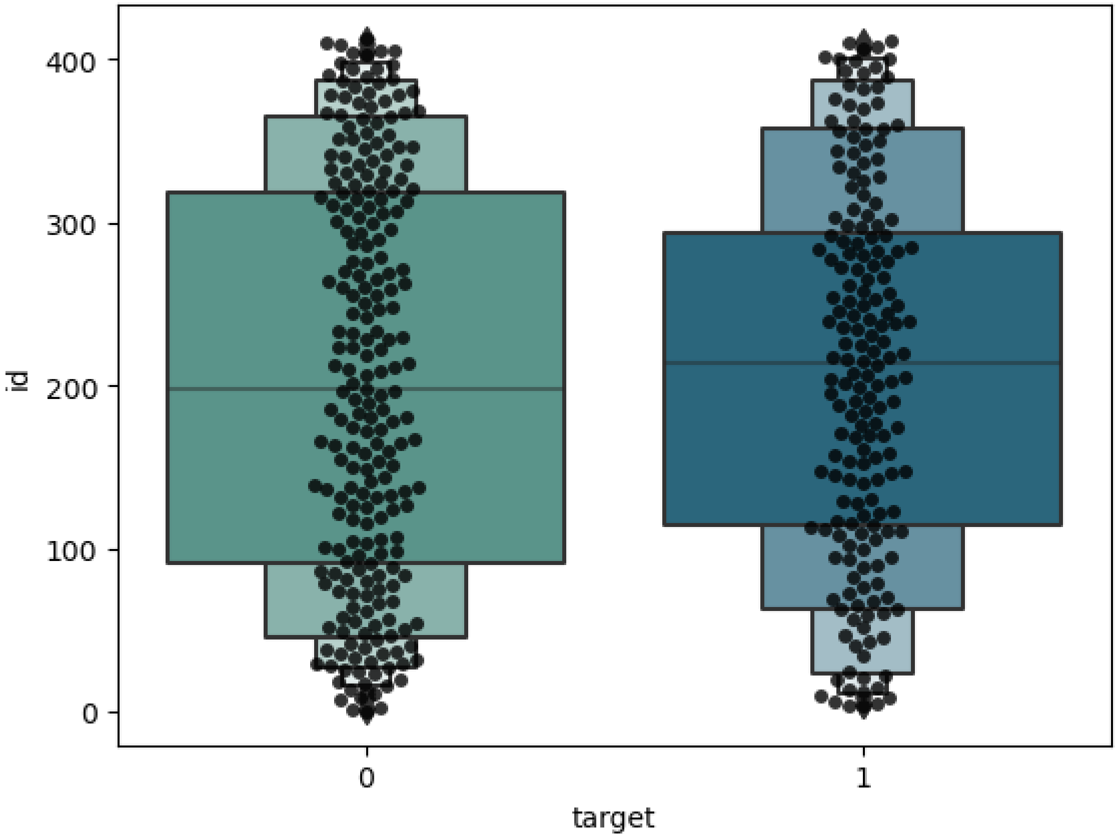

Count
target
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8
1.0

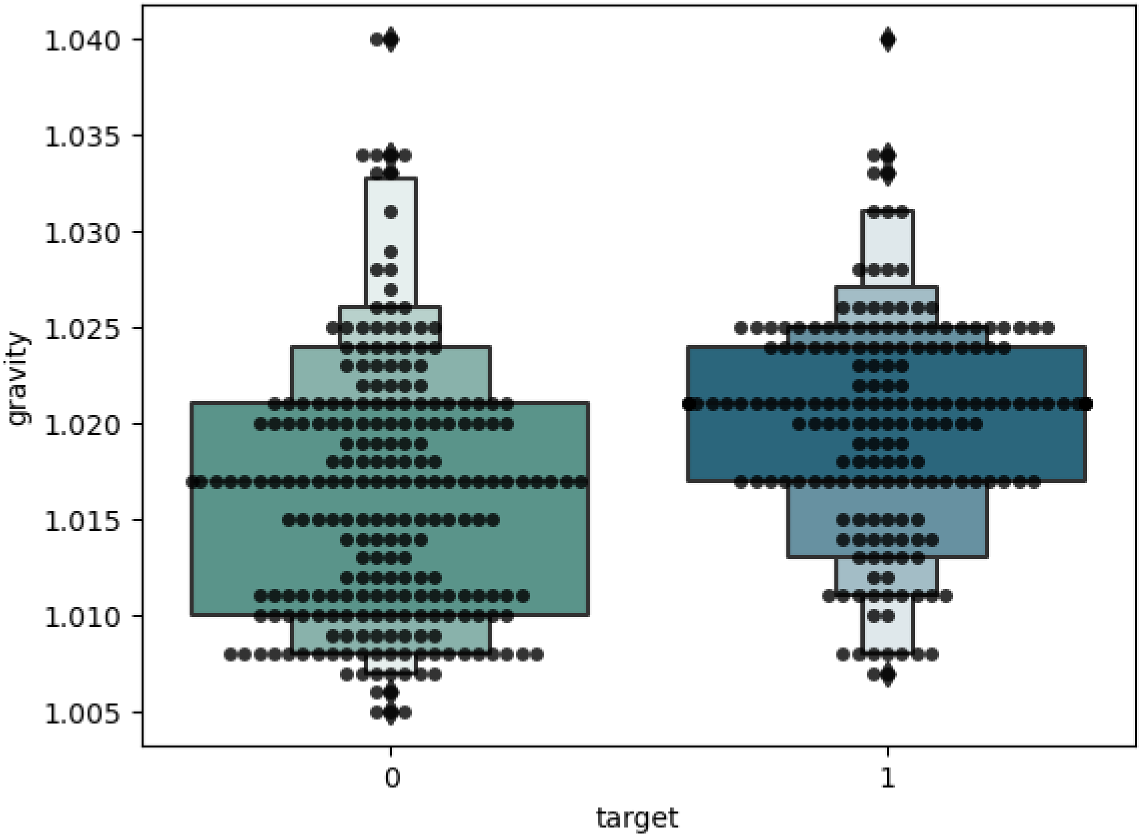
In [9]:
```
sns.pairplot(df,hue='target')
```

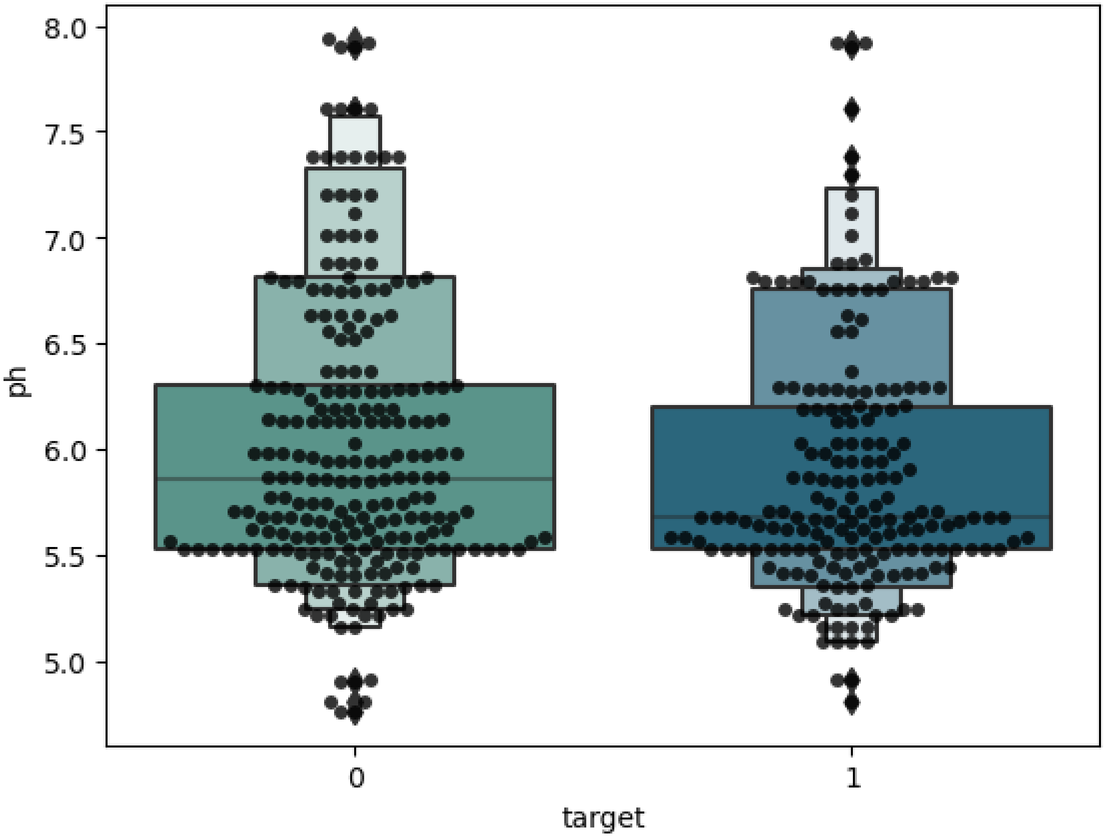
Out[9]: <seaborn.axisgrid.PairGrid at 0x7e908d7ce110>

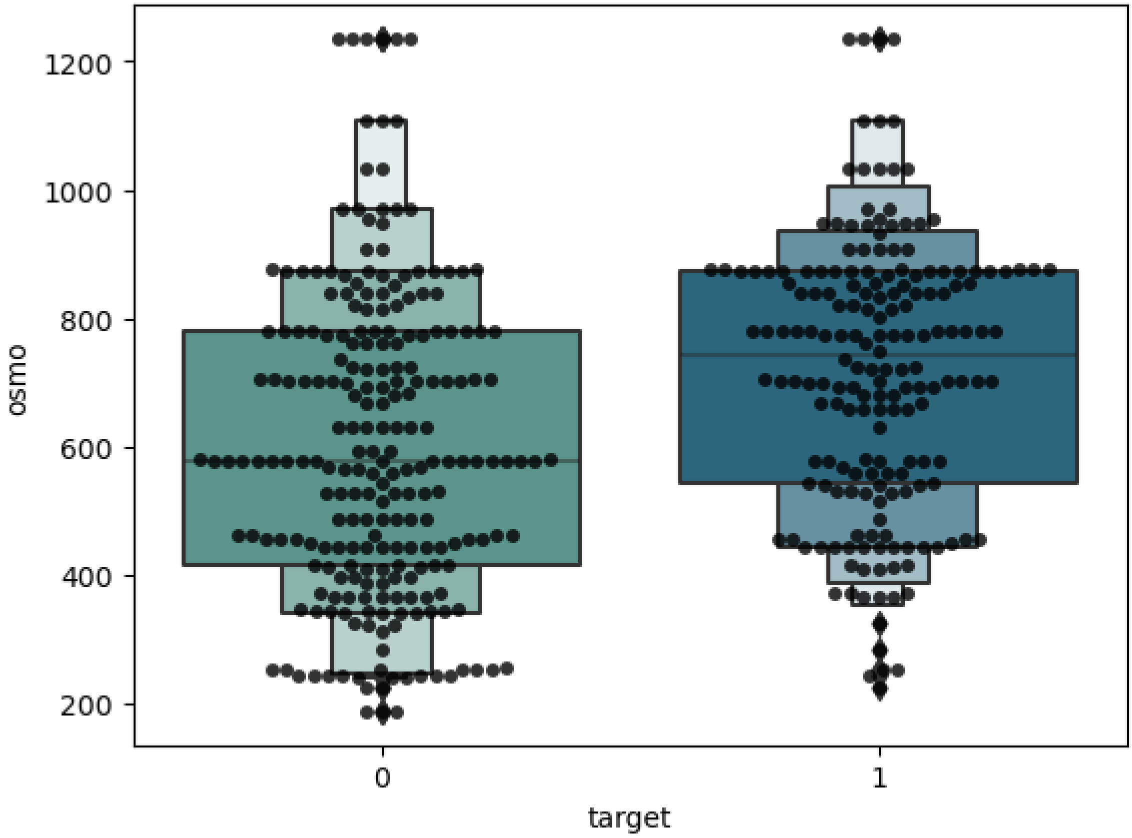


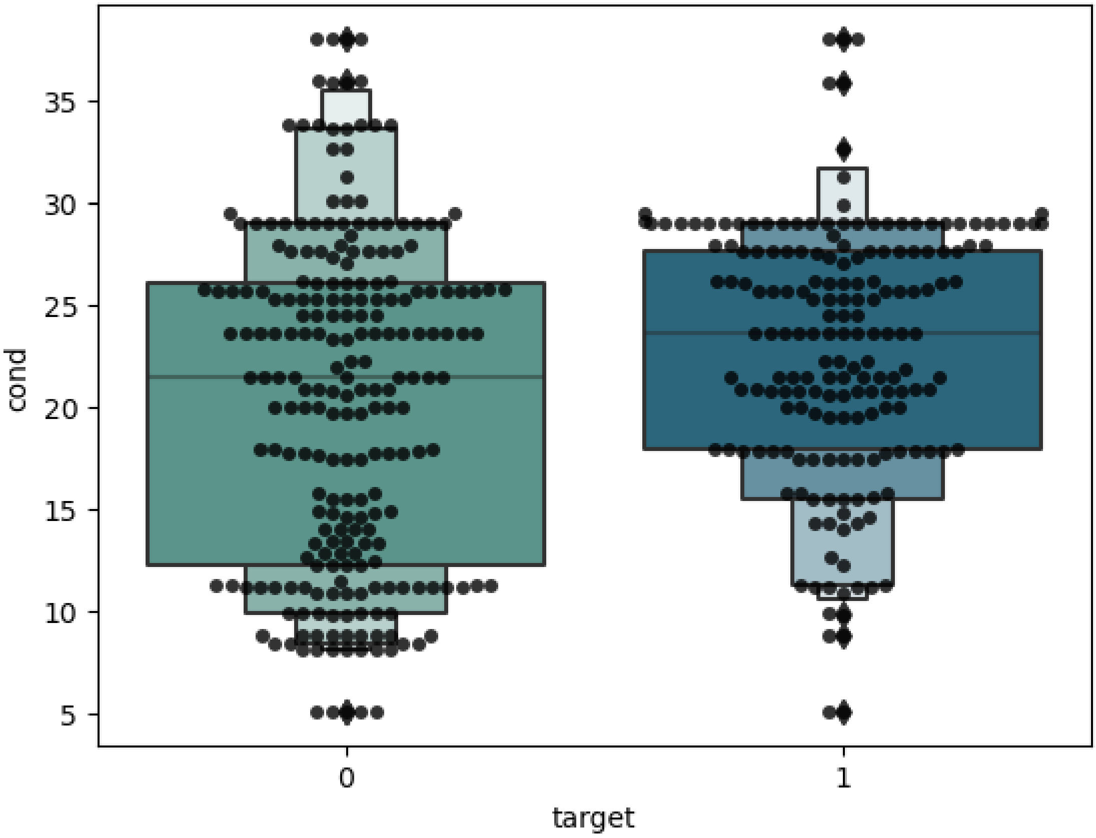
In [10]:
```
for i in df:
    sns.swarmplot(x = df["target"], y = df[i], color = "black", alpha = 0.8)
    sns.boxenplot(x = df["target"], y = df[i], palette="crest")
    plt.show()
```

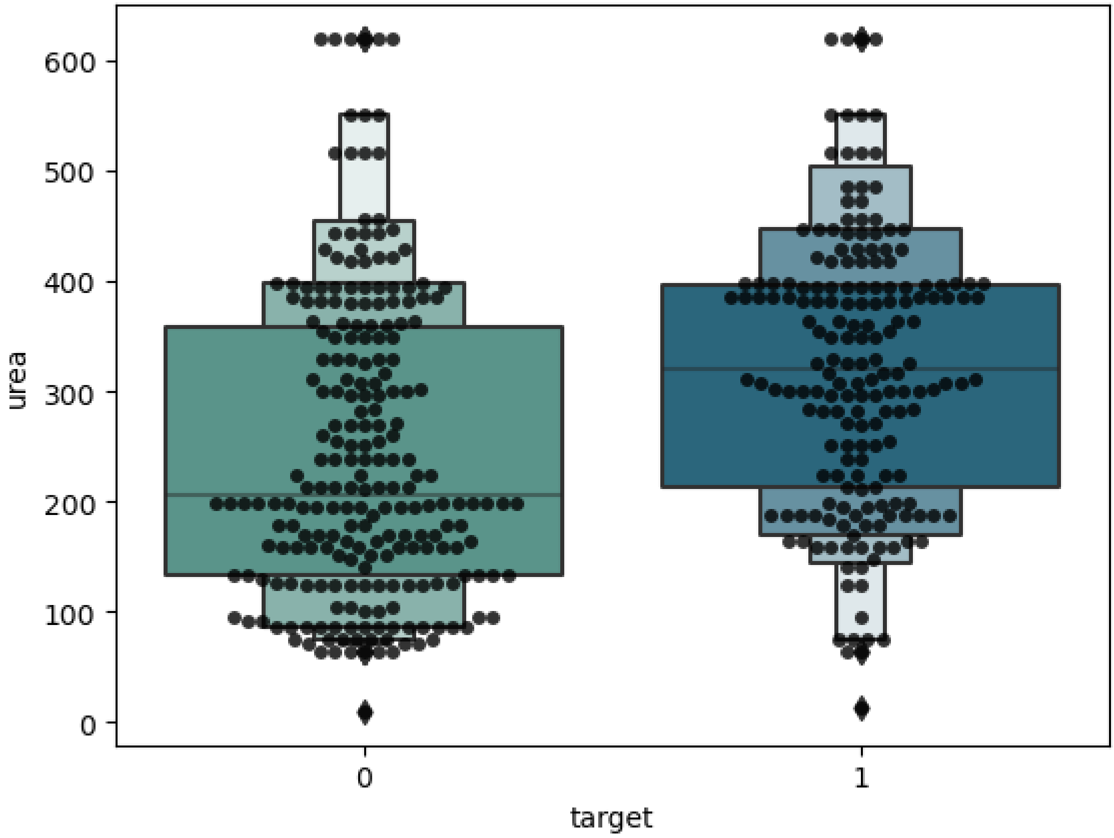

400
300
200
100
0
id
0
1
target

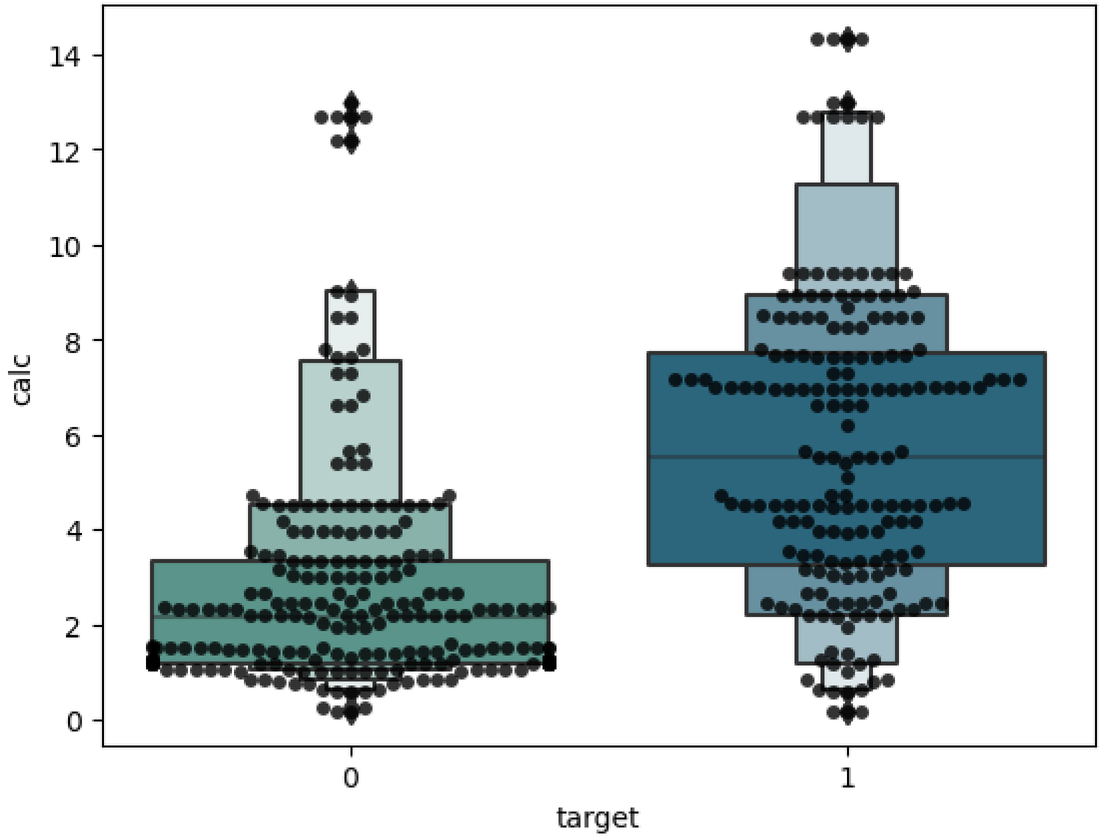

1.040
1.035
1.030
1.025
1.020
1.015
1.010
1.005
gravity
0
1
target

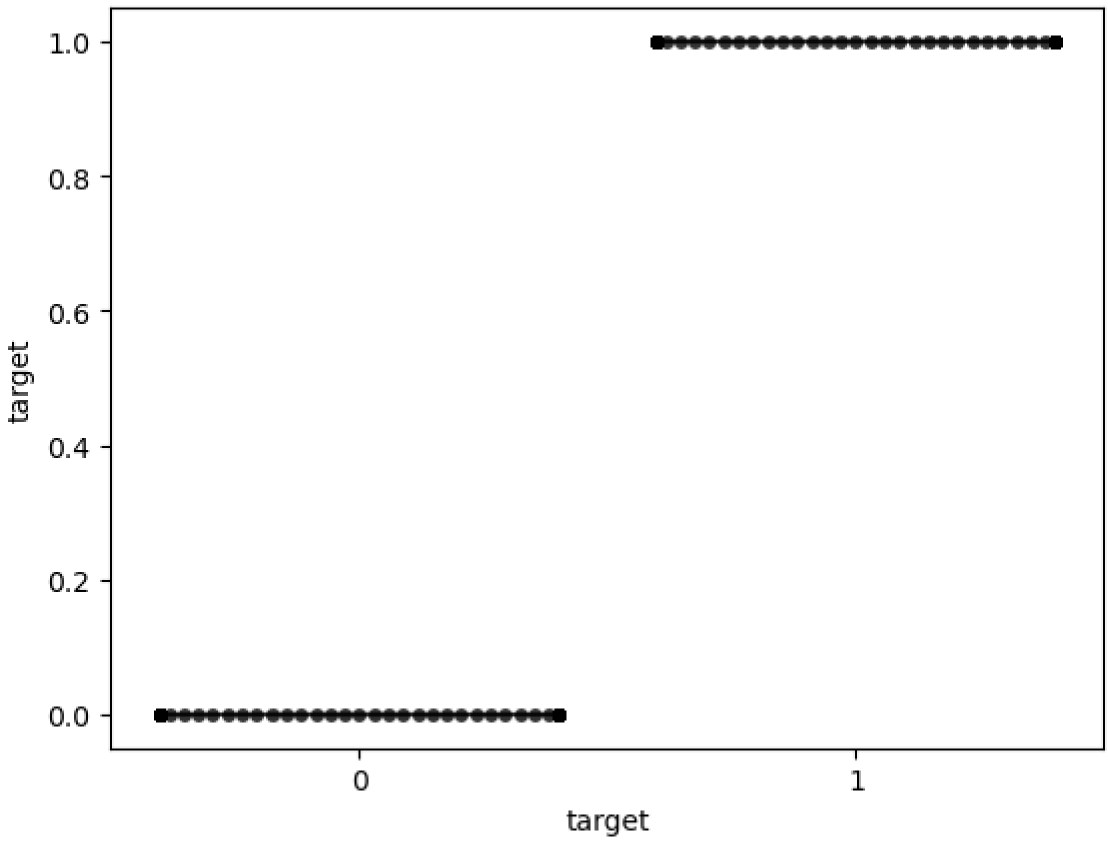

8.0
7.5
7.0
6.5
6.0
5.5
5.0
ph
0
1
target


1200
1000
800
600
400
200
osmo
0
1
target


35
30
25
20
15
10
5
cond
0
1
target


600
500
400
300
200
100
0
urea
0
1
target





In [11]:
```
df.corr()
```

Out[11]:

	id	gravity	ph	osmo	cond	urea	calc	target
id	1.000000	-0.004775	-0.086619	0.008030	0.032843	-0.023822	0.032360	0.018222
gravity	-0.004775	1.000000	-0.290349	0.692317	0.470433	0.631710	0.494304	0.282577
ph	-0.086619	-0.290349	1.000000	-0.309495	-0.190185	-0.279749	-0.214402	-0.094983
osmo	0.008030	0.692317	-0.309495	1.000000	0.708480	0.809880	0.472114	0.244770
cond	0.032843	0.470433	-0.190185	0.708480	1.000000	0.499109	0.330609	0.172224
urea	-0.023822	0.631710	-0.279749	0.809880	0.499109	1.000000	0.489879	0.265211
calc	0.032360	0.494304	-0.214402	0.472114	0.330609	0.489879	1.000000	0.467439
target	0.018222	0.282577	-0.094983	0.244770	0.172224	0.265211	0.467439	1.000000

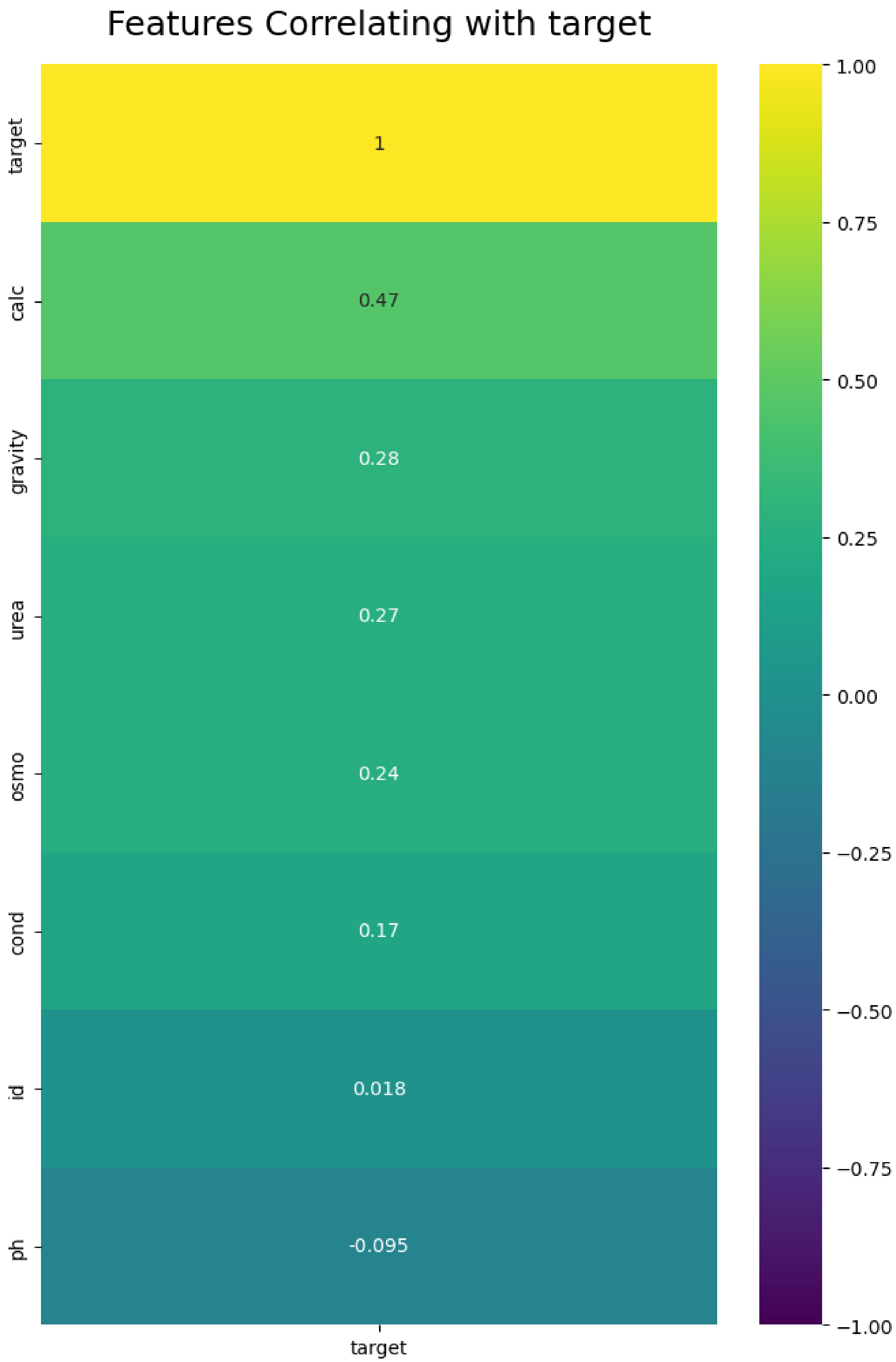
In [12]:

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),annot=True,cbar=False,cmap='Blues',fmt='.1f')
plt.show()
```

In [13]:

```
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(df.corr()[['target']].sort_values(by='target', ascending=Fals
heatmap.set_title('Features Correlating with target', fontdict={'fontsize':18}, pad
```

Out[13]: Text(0.5, 1.0, 'Features Correlating with target')



Splitting the Dataset

In [14]:
```
#Splitting Data
X = df.drop('target', axis = 1)
```

```
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_st
```

Model Preparation

In [15]:
```
kfold = StratifiedKFold(n_splits=10)
random_state = 42

classifiers = []

classifiers.append(SVC(random_state=random_state))
classifiers.append(DecisionTreeClassifier(random_state=random_state))
classifiers.append(AdaBoostClassifier(DecisionTreeClassifier(random_state=random_st
classifiers.append(RandomForestClassifier(random_state=random_state))
classifiers.append(GradientBoostingClassifier(random_state=random_state))
classifiers.append(KNeighborsClassifier())
classifiers.append(ExtraTreesClassifier())
classifiers.append(XGBClassifier())
classifiers.append(GaussianNB())
classifiers.append(LogisticRegression(random_state = random_state,max_iter=1000))

cv_results = []
for classifier in classifiers :
    cv_results.append(cross_val_score(classifier, X_train, y = y_train, scoring = '

cv_means = []
cv_std = []
for cv_result in cv_results:
    cv_means.append(cv_result.mean())
    cv_std.append(cv_result.std())

cv_res = pd.DataFrame({"CrossValMeans":cv_means,"CrossValerrors": cv_std,"Algorithn
"RandomForest","GradientBoosting","KNeighboors","ExtraTreesClassifier","XGBClassifi
```

In [16]:
```
cv_res.sort_values(by = 'CrossValMeans', ascending = False)
```

Out[16]:

	CrossValMeans	CrossValerrors	Algorithm
4	0.736720	0.100146	GradientBoosting
6	0.734046	0.075470	ExtraTreesClassifier
3	0.730927	0.071531	RandomForest
9	0.715954	0.078247	LogisticRegression
7	0.700802	0.089607	XGBClassifier
8	0.655615	0.073035	GaussianNB
2	0.646435	0.070839	AdaBoost
1	0.634492	0.071930	DecisionTree
0	0.604456	0.084832	SVC
5	0.583244	0.077690	KNeighboors

In [17]:
```
test_df=pd.read_csv('/kaggle/input/playground-series-s3e12/test.csv')
```

In [18]:
```
GB = GradientBoostingClassifier()
GB.fit(X,y)
```

Out[18]:
```
GradientBoostingClassifier()
```

In [19]:
```
ET=ExtraTreesClassifier()
ET.fit(X,y)
```

Out[19]:
```
ExtraTreesClassifier()
```

In [20]:
```
XGB=XGBClassifier()
XGB.fit(X,y)
```

Out[20]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [21]:
```
RFC=RandomForestClassifier()
RFC.fit(X,y)
```

Out[21]:
```
RandomForestClassifier()
```

In [22]:
```
ET.score(X_test,y_test)
```

Out[22]:
```
1.0
```

In [23]:
```
GB.score(X_test,y_test)
```

Out[23]:
```
0.9879518072289156
```

In [24]:
```
RFC.score(X_test,y_test)
```

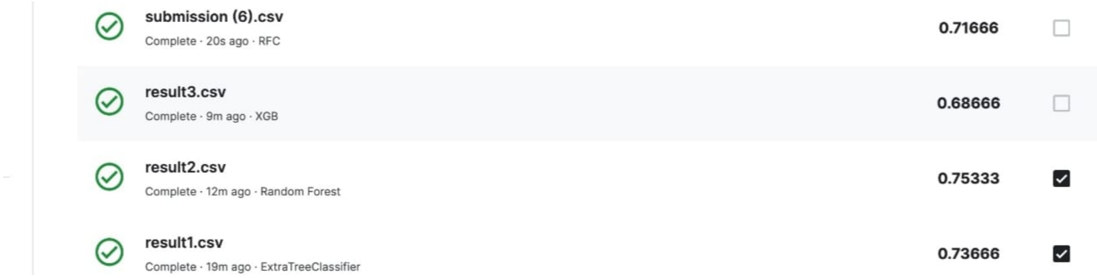
Out[24]:
```
1.0
```

In [25]:
```
XGB.score(X_test,y_test)
```

Out[25]:
```
1.0
```

Submission

In [26]:
```
y_pred = RFC.predict(test_df)
import numpy as np
result=pd.DataFrame({'id':np.array(test_df['id']),'target':y_pred})
result.to_csv('result2.csv',index=False)
```



RandomForest appears to be the best model by default, followed by ExtraTrees, although hyperparameter tweaking can enhance the score of boosting models. In addition, for a more generalizable model, combine the best models in the end.

Thanks for taking the time to read my notebook

In [ ]:

```

```