```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
C:\Users\thero\Anaconda3\lib\site-packages\statsmodels\tools\
_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use
the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

```
df=pd.read_csv(r'D:\Datasets\water_potability.csv')
df.head()
```

```
         ph    Hardness        Solids  Chloramines     Sulfate
Conductivity  \
0       NaN  204.890455  20791.318981     7.300212  368.516441
564.308654
1  3.716080  129.422921  18630.057858     6.635246         NaN
592.885359
2  8.099124  224.236259  19909.541732     9.275884         NaN
418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136
363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738
398.410813

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
```

Exploratory Data Analysis

```
df.shape
```

```
(3276, 10)
```

```
df.isnull().sum()
```

```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB

df.describe()

                ph     Hardness        Solids  Chloramines  
Sulfate  \
count  2785.000000  3276.000000   3276.000000  3276.000000  
2495.000000
mean      7.080795   196.369496  22014.092526     7.122277  
333.775777
std       1.594320    32.879761   8768.570828     1.583085  
41.416840
min       0.000000    47.432000    320.942611     0.352000  
129.000000
25%       6.093092   176.850538  15666.690297     6.127421  
307.699498
50%       7.036752   196.967627  20927.833607     7.130299  
333.073546
75%       8.062066   216.667456  27332.762127     8.114887  
359.950170
max      14.000000   323.124000  61227.196008    13.127000  
481.030642

       Conductivity  Organic_carbon  Trihalomethanes    Turbidity  
Potability
count   3276.000000     3276.000000      3114.000000  3276.000000  
3276.000000
mean     426.205111       14.284970        66.396293     3.966786  
0.390110
std       80.824064        3.308162        16.175008     0.780382  
0.487849
min      181.483754        2.200000         0.738000     1.450000  
0.000000
```

```
25%      365.734414       12.065801        55.844536     3.439711
0.000000
50%      421.884968       14.218338        66.622485     3.955028
0.000000
75%      481.792304       16.557652        77.337473     4.500320
1.000000
max      753.342620       28.300000       124.000000     6.739000
1.000000
```

```
df['Sulfate'].mean()
```

```
333.77577661081335
```

```
df.fillna(df.mean(), inplace=True)
df.head()
```

```
         ph    Hardness        Solids  Chloramines     Sulfate
Conductivity  \
0  7.080795  204.890455  20791.318981     7.300212  368.516441
564.308654
1  3.716080  129.422921  18630.057858     6.635246  333.775777
592.885359
2  8.099124  224.236259  19909.541732     9.275884  333.775777
418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136
363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738
398.410813

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
```

```
df.isnull().sum()
```

```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

```
df.info()
```

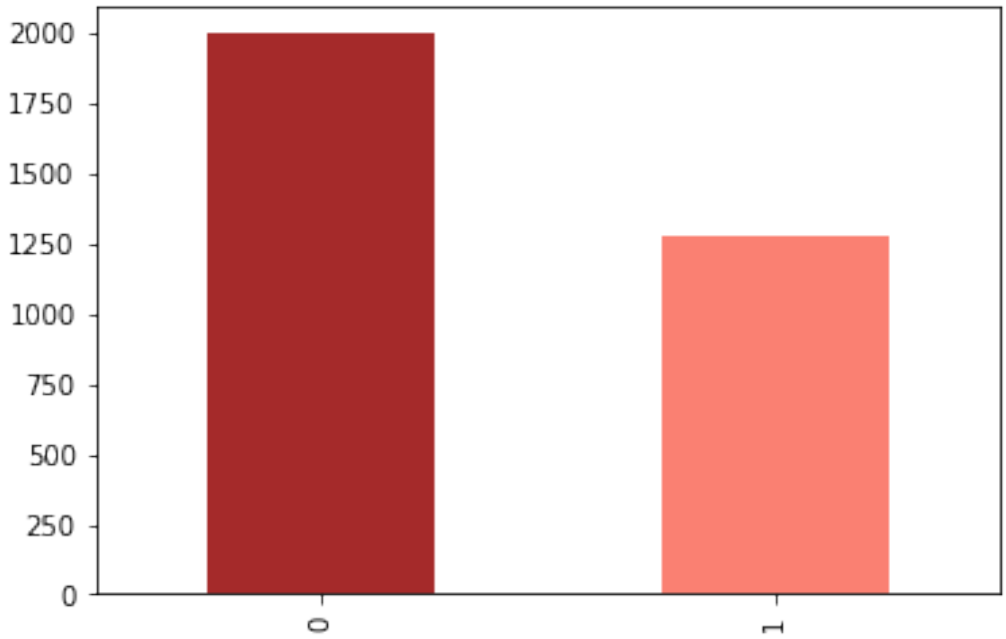
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          3276 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3276 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB

df.describe()

                ph     Hardness        Solids  Chloramines  
Sulfate  \
count  3276.000000  3276.000000   3276.000000  3276.000000  
3276.000000
mean      7.080795   196.369496  22014.092526     7.122277  
333.775777
std       1.469956    32.879761   8768.570828     1.583085  
36.142612
min       0.000000    47.432000    320.942611     0.352000  
129.000000
25%       6.277673   176.850538  15666.690297     6.127421  
317.094638
50%       7.080795   196.967627  20927.833607     7.130299  
333.775777
75%       7.870050   216.667456  27332.762127     8.114887  
350.385756
max      14.000000   323.124000  61227.196008    13.127000  
481.030642

       Conductivity  Organic_carbon  Trihalomethanes    Turbidity  
Potability
count   3276.000000     3276.000000      3276.000000  3276.000000  
3276.000000
mean     426.205111       14.284970        66.396293     3.966786  
0.390110
std       80.824064        3.308162        15.769881     0.780382  
0.487849
min      181.483754        2.200000         0.738000     1.450000  
0.000000
25%      365.734414       12.065801        56.647656     3.439711
```

```
0.000000
50%      421.884968       14.218338        66.396293     3.955028
0.000000
75%      481.792304       16.557652        76.666609     4.500320
1.000000
max      753.342620       28.300000       124.000000     6.739000
1.000000
```

```
df.Potability.value_counts()
```

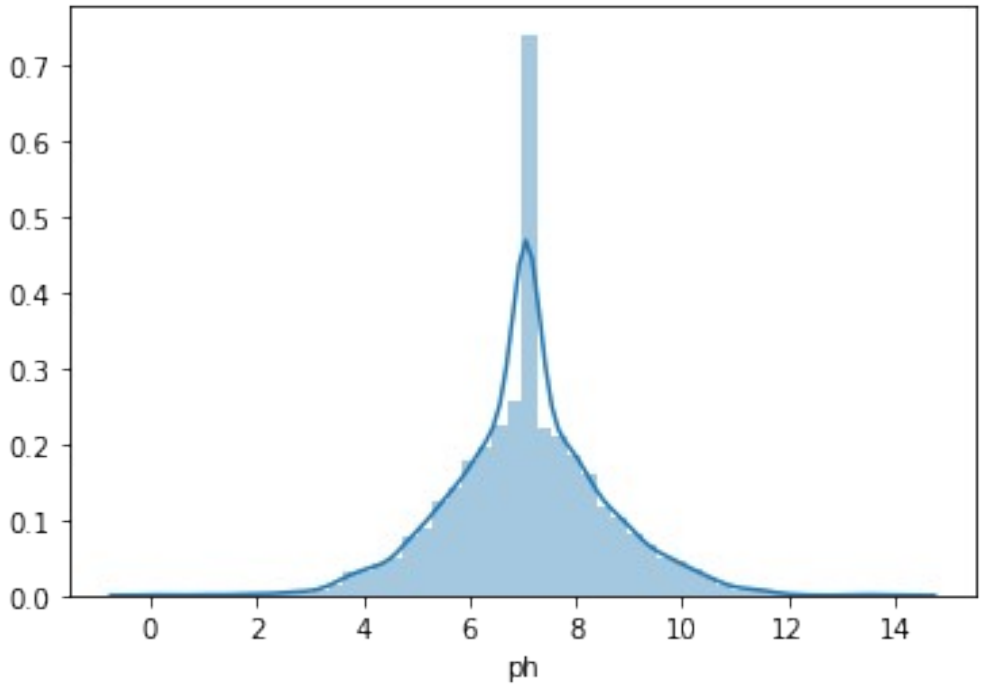
```
0    1998
1    1278
Name: Potability, dtype: int64
```

```
df.Potability.value_counts().plot(kind="bar", color=["brown",
"salmon"])
plt.show()
```

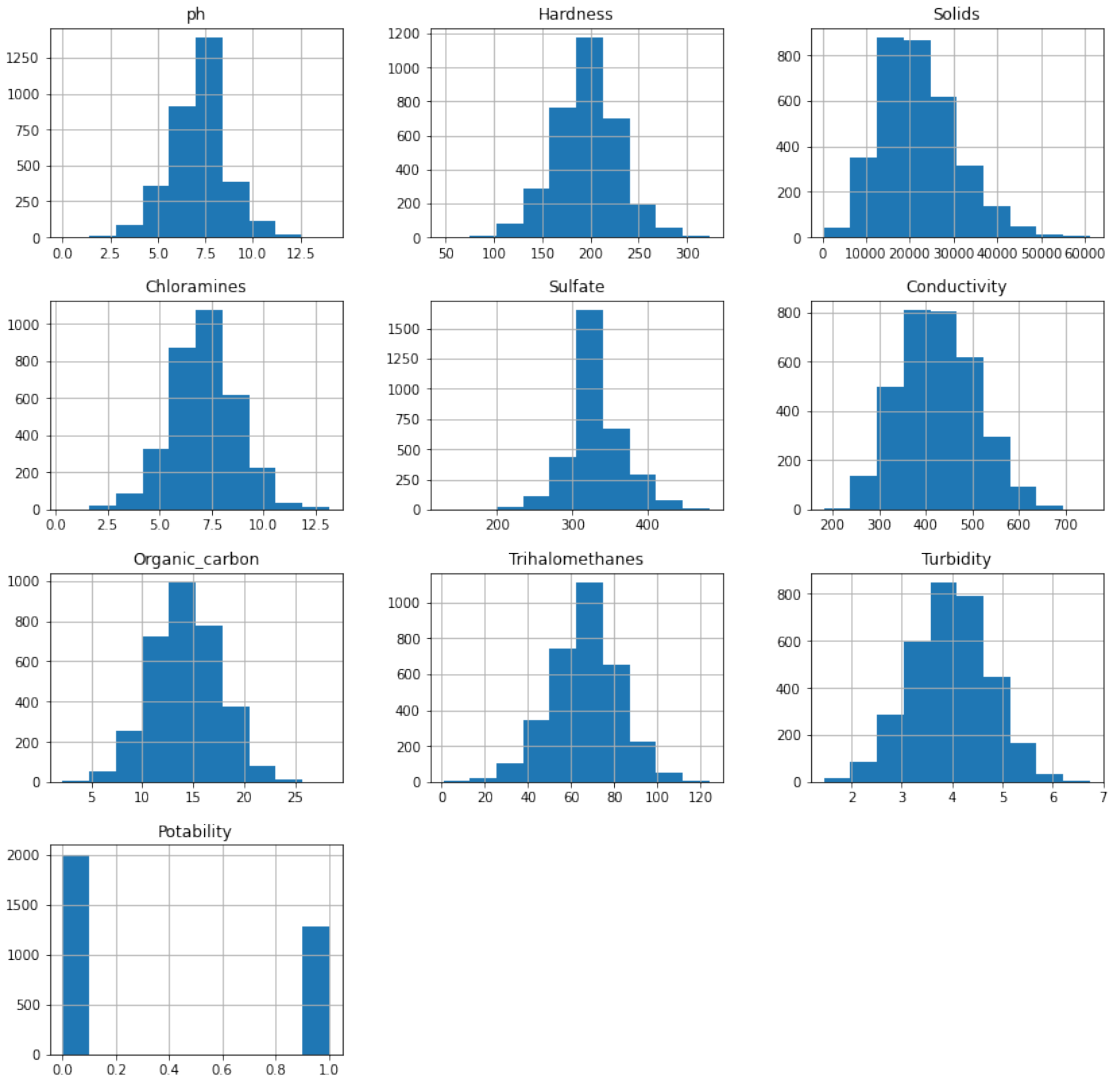



```
sns.distplot(df['ph'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x203e0631d48>
```

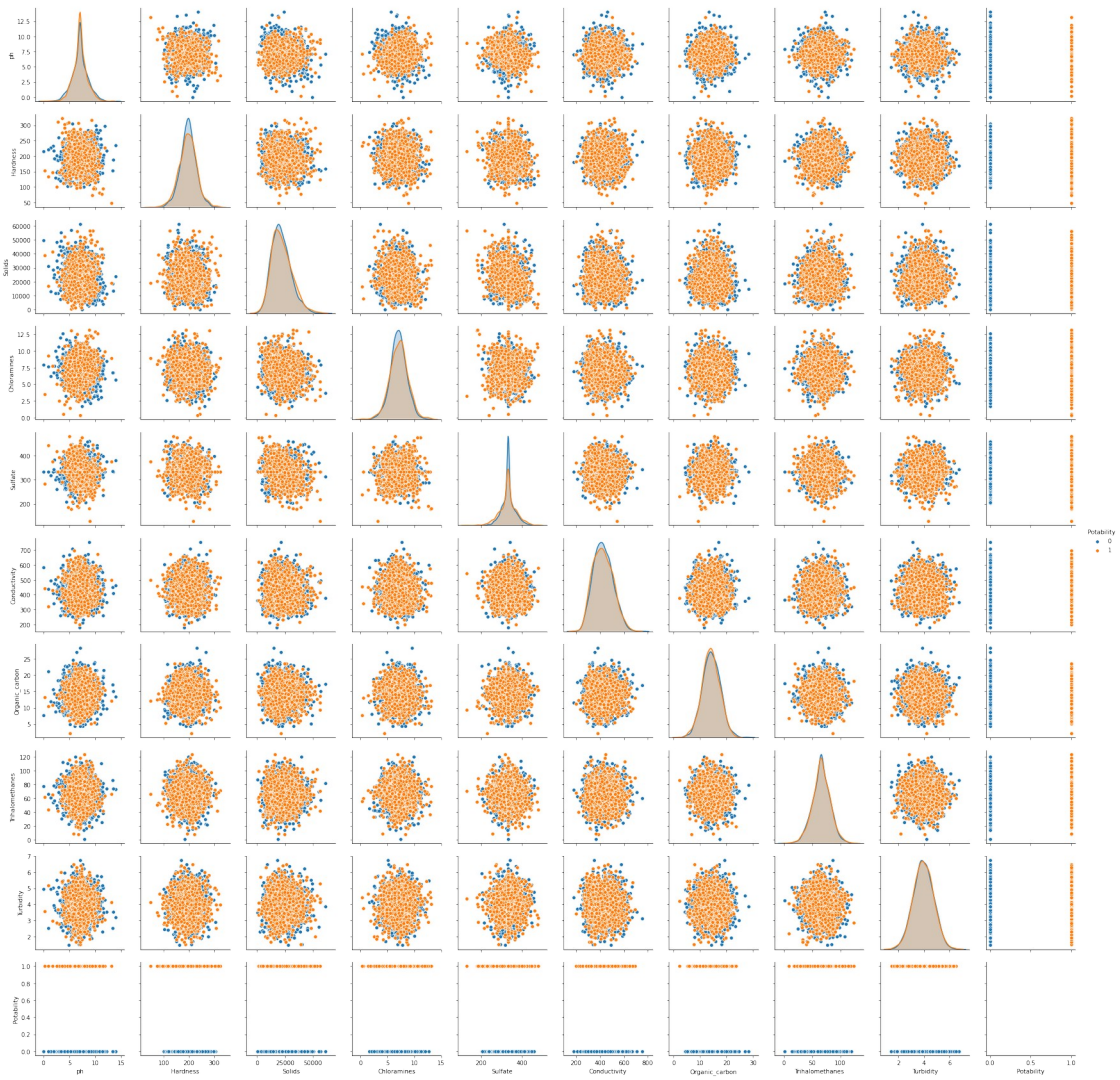



```
df.hist(figsize=(14,14))
plt.show()
```



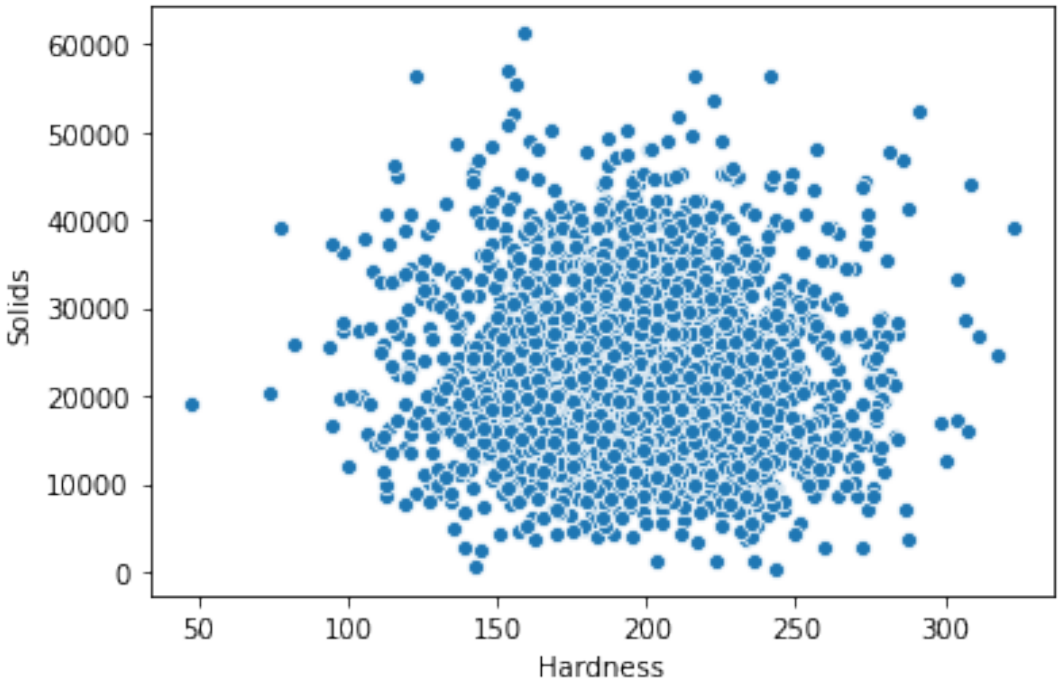

```
sns.pairplot(df,hue='Potability')
```

```
C:\Users\thero\Anaconda3\lib\site-packages\statsmodels\nonparametric\
kde.py:487: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\Users\thero\Anaconda3\lib\site-packages\statsmodels\nonparametric\
kdetools.py:34: RuntimeWarning: invalid value encountered in
double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
C:\Users\thero\Anaconda3\lib\site-packages\statsmodels\nonparametric\
kde.py:487: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\Users\thero\Anaconda3\lib\site-packages\statsmodels\nonparametric\
kdetools.py:34: RuntimeWarning: invalid value encountered in
double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2

<seaborn.axisgrid.PairGrid at 0x203e09dbf48>
```



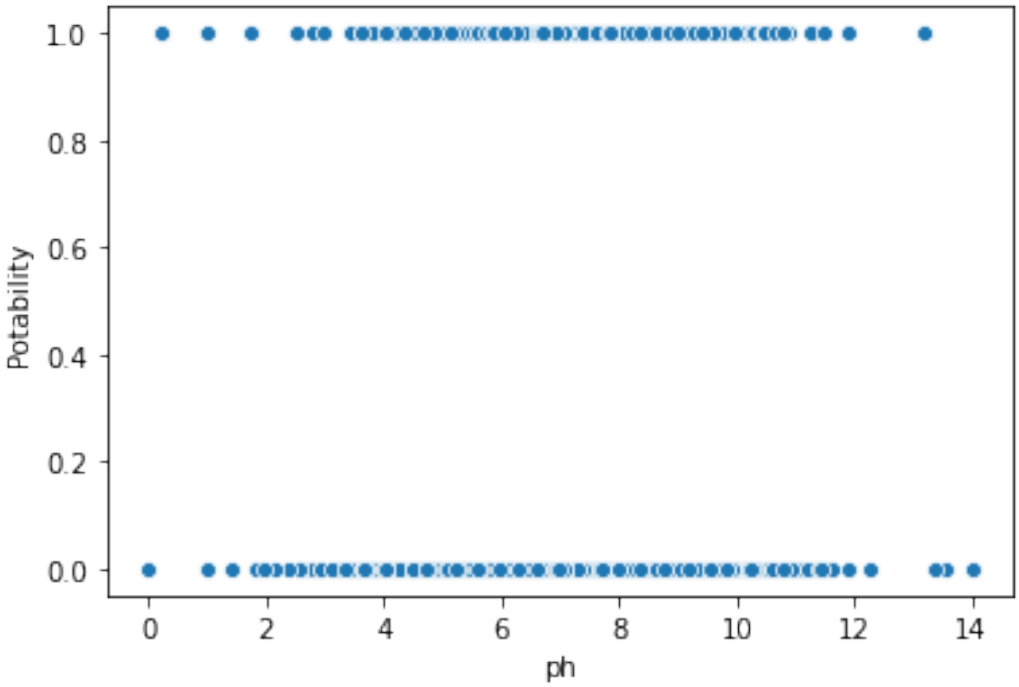

```
sns.scatterplot(df['Hardness'],df['Solids'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x203bf3aa9c8>
```



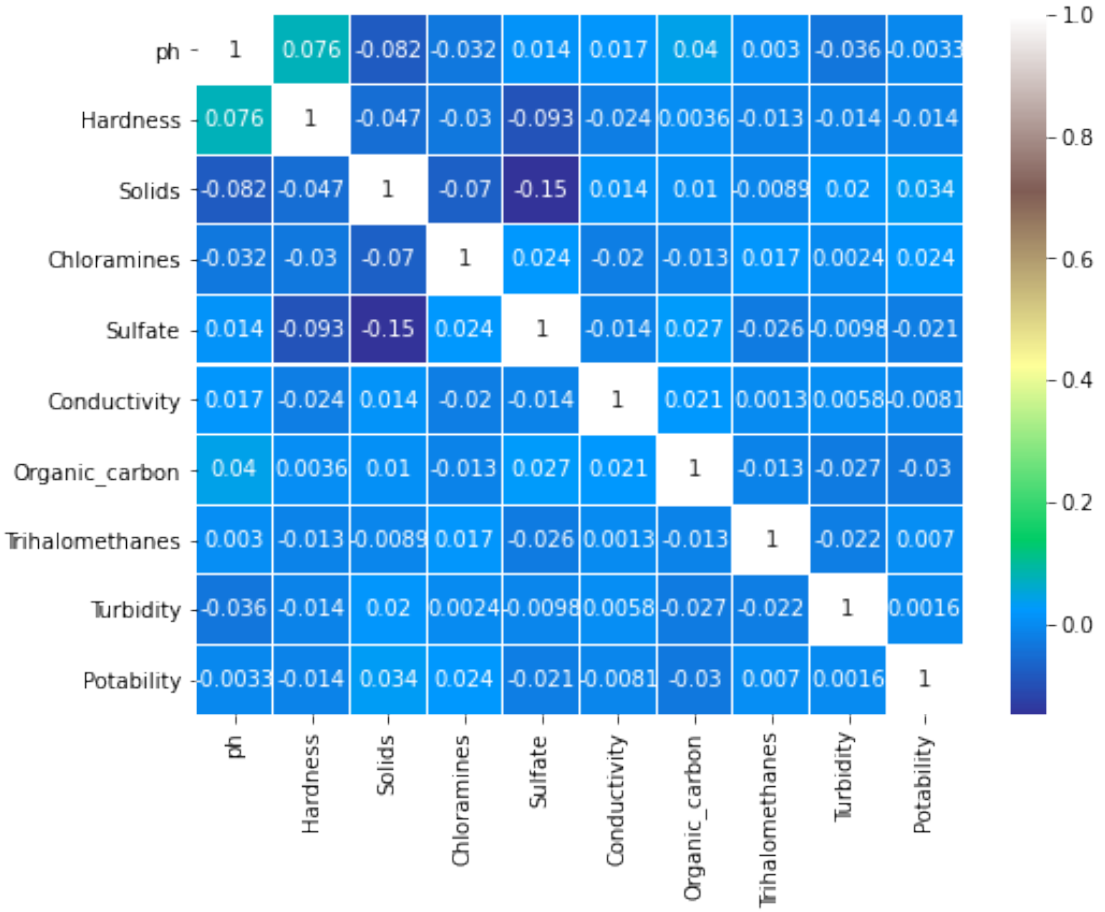

```
sns.scatterplot(df['ph'],df['Potability'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x203bf1f6e48>
```



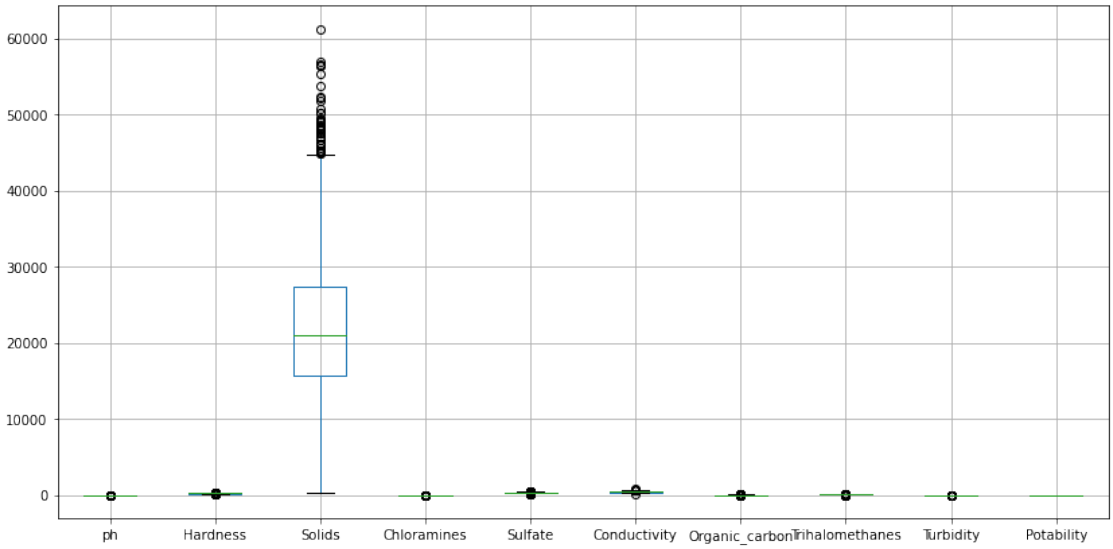

```
# create a correlation heatmap
sns.heatmap(df.corr(),annot=True, cmap='terrain', linewidths=0.1)
fig=plt.gcf()
```

```
fig.set_size_inches(8,6)
plt.show()
```




```
df.boxplot(figsize=(14,7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x203c11b1388>
```



```
df['Solids'].describe()
```

```
count     3276.000000
mean     22014.092526
std       8768.570828
min        320.942611
25%      15666.690297
50%      20927.833607
75%      27332.762127
max      61227.196008
Name: Solids, dtype: float64
```

Partitioning

```
X = df.drop('Potability',axis=1)

Y= df['Potability']

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=
0.2, random_state=101,shuffle=True)

Y_train.value_counts()
```

```
0    1596
1    1024
Name: Potability, dtype: int64
```

```
Y_test.value_counts()
```

```
0    402
1    254
Name: Potability, dtype: int64
```

Normalization

```
#from sklearn.preprocessing import StandardScaler
#sc=StandardScaler()

#X_train = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)
```

Model Building

DT

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import
accuracy_score,confusion_matrix,precision_score
dt=DecisionTreeClassifier(criterion= 'gini', min_samples_split= 10,
splitter= 'best')
dt.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(min_samples_split=10)
```

```
prediction=dt.predict(X_test)
accuracy_dt=accuracy_score(Y_test,prediction)*100
accuracy_dt
```

```
58.84146341463414
```

```
print("Accuracy on training set: {:.3f}".format(dt.score(X_train,
Y_train)))
print("Accuracy on test set: {:.3f}".format(dt.score(X_test, Y_test)))
```

```
Accuracy on training set: 0.923
Accuracy on test set: 0.588
```

```
accuracy_score(prediction,Y_test)
```

```
0.5884146341463414
```

```
print("Feature importances:\n{}".format(dt.feature_importances_))
```

```
Feature importances:
[0.14979308 0.1306299  0.11639812 0.10910309 0.11691893 0.09302001
 0.10460544 0.10201786 0.07751357]
```

```
confusion_matrix(prediction,Y_test)
```

```
array([[272, 140],
       [130, 114]], dtype=int64)
```

Prediction on only one set of data

```
X_DT=dt.predict([[5.735724,
158.318741,25363.016594,7.728601,377.543291,568.304671,13.626624,75.95
2337,4.732954]])
```

```
X_DT
```

```
array([1], dtype=int64)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(metric='manhattan', n_neighbors=22)
knn.fit(X_train,Y_train)

KNeighborsClassifier(metric='manhattan', n_neighbors=22)

prediction_knn=knn.predict(X_test)
accuracy_knn=accuracy_score(Y_test,prediction_knn)*100
print('accuracy_score score     :
',accuracy_score(Y_test,prediction_knn)*100,'%')

accuracy_score score     :  61.737804878048784 %

confusion_matrix(prediction,Y_test)

array([[272, 140],
       [130, 114]], dtype=int64)
```

Hyperparameter Tuning / Model Optimization

DT HPT

```
dt.get_params().keys()

dict_keys(['ccp_alpha', 'class_weight', 'criterion', 'max_depth',
'max_features', 'max_leaf_nodes', 'min_impurity_decrease',
'min_impurity_split', 'min_samples_leaf', 'min_samples_split',
'min_weight_fraction_leaf', 'presort', 'random_state', 'splitter'])

#example of grid searching key hyperparametres for logistic regression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

# define models and parameters
model = DecisionTreeClassifier()
criterion = ["gini", "entropy"]
splitter = ["best", "random"]
```

```
min_samples_split = [2,4,6,8,10]

# define grid search
grid = dict(splitter=splitter, criterion=criterion,
min_samples_split=min_samples_split)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search_dt = GridSearchCV(estimator=model, param_grid=grid,
n_jobs=-1, cv=cv,
                              scoring='accuracy',error_score=0, iid=True)
grid_search_dt.fit(X_train, Y_train)

# summarize results
print(f"Best: {grid_search_dt.best_score_:.3f} using
{grid_search_dt.best_params_}")
means = grid_search_dt.cv_results_['mean_test_score']
stds = grid_search_dt.cv_results_['std_test_score']
params = grid_search_dt.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")

print("Training Score:",grid_search_dt.score(X_train, Y_train)*100)
print("Testing Score:", grid_search_dt.score(X_test, Y_test)*100)
```

```
Best: 0.590 using {'criterion': 'gini', 'min_samples_split': 10,
'splitter': 'best'}
0.584 (0.029) with: {'criterion': 'gini', 'min_samples_split': 2,
'splitter': 'best'}
0.569 (0.030) with: {'criterion': 'gini', 'min_samples_split': 2,
'splitter': 'random'}
0.584 (0.028) with: {'criterion': 'gini', 'min_samples_split': 4,
'splitter': 'best'}
0.571 (0.025) with: {'criterion': 'gini', 'min_samples_split': 4,
'splitter': 'random'}
0.588 (0.029) with: {'criterion': 'gini', 'min_samples_split': 6,
'splitter': 'best'}
0.584 (0.031) with: {'criterion': 'gini', 'min_samples_split': 6,
'splitter': 'random'}
0.584 (0.031) with: {'criterion': 'gini', 'min_samples_split': 8,
'splitter': 'best'}
0.583 (0.036) with: {'criterion': 'gini', 'min_samples_split': 8,
'splitter': 'random'}
0.590 (0.028) with: {'criterion': 'gini', 'min_samples_split': 10,
'splitter': 'best'}
0.589 (0.026) with: {'criterion': 'gini', 'min_samples_split': 10,
'splitter': 'random'}
0.589 (0.027) with: {'criterion': 'entropy', 'min_samples_split': 2,
'splitter': 'best'}
0.571 (0.030) with: {'criterion': 'entropy', 'min_samples_split': 2,
'splitter': 'random'}
```

```
0.586 (0.030) with: {'criterion': 'entropy', 'min_samples_split': 4,
'splitter': 'best'}
0.578 (0.032) with: {'criterion': 'entropy', 'min_samples_split': 4,
'splitter': 'random'}
0.583 (0.029) with: {'criterion': 'entropy', 'min_samples_split': 6,
'splitter': 'best'}
0.582 (0.033) with: {'criterion': 'entropy', 'min_samples_split': 6,
'splitter': 'random'}
0.585 (0.026) with: {'criterion': 'entropy', 'min_samples_split': 8,
'splitter': 'best'}
0.576 (0.027) with: {'criterion': 'entropy', 'min_samples_split': 8,
'splitter': 'random'}
0.586 (0.029) with: {'criterion': 'entropy', 'min_samples_split': 10,
'splitter': 'best'}
0.585 (0.035) with: {'criterion': 'entropy', 'min_samples_split': 10,
'splitter': 'random'}
Training Score: 92.32824427480915
Testing Score: 59.60365853658537

C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning
```

```
from sklearn.metrics import  make_scorer
from sklearn.model_selection import cross_val_score

def classification_report_with_accuracy_score(Y_test, y_pred2):
    print (classification_report(Y_test, y_pred2)) # print
classification report
    return accuracy_score(Y_test, y_pred2) # return accuracy score


nested_score = cross_val_score(grid_search_dt, X=X_train, y=Y_train,
cv=cv,

scoring=make_scorer(classification_report_with_accuracy_score))
print (nested_score)
```

```
C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.62      0.68      0.65       159
           1       0.42      0.36      0.39       103

    accuracy                           0.55       262
   macro avg       0.52      0.52      0.52       262
```

```
weighted avg       0.54      0.55      0.55       262


---------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-217-e8cc3119bf88> in <module>
      8
      9 nested_score = cross_val_score(grid_search_dt, X=X_train,
y=Y_train, cv=cv,
---> 10
scoring=make_scorer(classification_report_with_accuracy_score))
     11 print (nested_score)

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in
zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

~\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py
in cross_val_score(estimator, X, y, groups, scoring, cv, n_jobs,
verbose, fit_params, pre_dispatch, error_score)
    404                                 fit_params=fit_params,
    405                                 pre_dispatch=pre_dispatch,
--> 406                                 error_score=error_score)
    407     return cv_results['test_score']
    408

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in
zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

~\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py
in cross_validate(estimator, X, y, groups, scoring, cv, n_jobs,
verbose, fit_params, pre_dispatch, return_train_score,
return_estimator, error_score)
    246             return_times=True,
return_estimator=return_estimator,
    247             error_score=error_score)
--> 248         for train, test in cv.split(X, y, groups))
```

```
    249
    250     zipped_scores = list(zip(*scores))

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self,
iterable)
    922                 self._iterating = self._original_iterator is
not None
    923
--> 924             while self.dispatch_one_batch(iterator):
    925                 pass
    926

~\Anaconda3\lib\site-packages\joblib\parallel.py in
dispatch_one_batch(self, iterator)
    757                 return False
    758             else:
--> 759                 self._dispatch(tasks)
    760                 return True
    761

~\Anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self,
batch)
    714         with self._lock:
    715             job_idx = len(self._jobs)
--> 716             job = self._backend.apply_async(batch,
callback=cb)
    717             # A job can complete so quickly than its callback
is
    718             # called before we get here, causing self._jobs to

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in
apply_async(self, func, callback)
    180     def apply_async(self, func, callback=None):
    181         """Schedule a func to be run"""
--> 182         result = ImmediateResult(func)
    183         if callback:
    184             callback(result)

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in
__init__(self, batch)
    547         # Don't delay the application, to avoid keeping the
input
    548         # arguments in memory
--> 549         self.results = batch()
    550
    551     def get(self):

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    223         with parallel_backend(self._backend,
n_jobs=self._n_jobs):
```

```
    224             return [func(*args, **kwargs)
--> 225                     for func, args, kwargs in self.items]
    226
    227     def __len__(self):

~\Anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    223         with parallel_backend(self._backend,
n_jobs=self._n_jobs):
    224             return [func(*args, **kwargs)
--> 225                     for func, args, kwargs in self.items]
    226
    227     def __len__(self):

~\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py
in _fit_and_score(estimator, X, y, scorer, train, test, verbose,
parameters, fit_params, return_train_score, return_parameters,
return_n_test_samples, return_times, return_estimator, error_score)
    529             estimator.fit(X_train, **fit_params)
    530         else:
--> 531             estimator.fit(X_train, y_train, **fit_params)
    532
    533     except Exception as e:

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in
zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in
fit(self, X, y, groups, **fit_params)
    734                 return results
    735
--> 736             self._run_search(evaluate_candidates)
    737
    738         # For multi-metric evaluation, store the best_index_,
best_params_ and

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in
_run_search(self, evaluate_candidates)
   1186     def _run_search(self, evaluate_candidates):
   1187         """Search all candidates in param_grid"""
-> 1188         evaluate_candidates(ParameterGrid(self.param_grid))
   1189
   1190

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in
```

```
evaluate_candidates(candidate_params)
    713                                   for parameters, (train, test)
    714                                   in product(candidate_params,
--> 715                                              cv.split(X, y,
groups)))
    716
    717                 if len(out) < 1:

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self,
iterable)
    932
    933             with self._backend.retrieval_context():
--> 934                 self.retrieve()
    935             # Make sure that we get a last message telling us
we are done
    936             elapsed_time = time.time() - self._start_time

~\Anaconda3\lib\site-packages\joblib\parallel.py in retrieve(self)
    831             try:
    832                 if getattr(self._backend, 'supports_timeout',
False):
--> 833
self._output.extend(job.get(timeout=self.timeout))
    834                 else:
    835                     self._output.extend(job.get())

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in
wrap_future_result(future, timeout)
    519         AsyncResults.get from multiprocessing."""
    520         try:
--> 521             return future.result(timeout=timeout)
    522         except LokyTimeoutError:
    523             raise TimeoutError()

~\Anaconda3\lib\concurrent\futures\_base.py in result(self, timeout)
    428                 return self.__get_result()
    429
--> 430             self._condition.wait(timeout)
    431
    432             if self._state in [CANCELLED,
CANCELLED_AND_NOTIFIED]:

~\Anaconda3\lib\threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g.,
KeyboardInterrupt)
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:
```

```
KeyboardInterrupt:
```

```
dt_y_predicted = grid_search_dt.predict(X_test)
dt_y_predicted
```

```
array([1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
1,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
1,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
0,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,
1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
1,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
1,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
1,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0,
```

```
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0,
0,
       0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
dtype=int64)
```

```
grid_search_dt.best_params_
```

```
{'criterion': 'gini', 'min_samples_split': 10, 'splitter': 'best'}
```

```
dt_grid_score=accuracy_score(Y_test, dt_y_predicted)
dt_grid_score
```

```
0.5960365853658537
```

```
confusion_matrix(Y_test, dt_y_predicted)
```

```
array([[276, 126],
       [139, 115]], dtype=int64)
```

KNN HPT

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

# define models and parameters
model = KNeighborsClassifier()
n_neighbors = range(1, 31)
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski']

# define grid search
grid = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
```

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=1, random_state=1)
grid_search_knn = GridSearchCV(estimator=model, param_grid=grid,
n_jobs=-1, cv=cv,
                               scoring='accuracy',error_score=0, iid=True)
grid_search_knn.fit(X_train, Y_train)

# summarize results
print(f"Best: {grid_search_knn.best_score_:.3f} using
{grid_search_knn.best_params_}")
means = grid_search_knn.cv_results_['mean_test_score']
stds = grid_search_knn.cv_results_['std_test_score']
params = grid_search_knn.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")
```

```
Best: 0.603 using {'metric': 'manhattan', 'n_neighbors': 22,
'weights': 'uniform'}
0.536 (0.029) with: {'metric': 'euclidean', 'n_neighbors': 1,
'weights': 'uniform'}
0.536 (0.029) with: {'metric': 'euclidean', 'n_neighbors': 1,
'weights': 'distance'}
0.579 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 2,
'weights': 'uniform'}
0.536 (0.029) with: {'metric': 'euclidean', 'n_neighbors': 2,
'weights': 'distance'}
0.542 (0.023) with: {'metric': 'euclidean', 'n_neighbors': 3,
'weights': 'uniform'}
0.542 (0.024) with: {'metric': 'euclidean', 'n_neighbors': 3,
'weights': 'distance'}
0.574 (0.017) with: {'metric': 'euclidean', 'n_neighbors': 4,
'weights': 'uniform'}
0.542 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 4,
'weights': 'distance'}
0.545 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 5,
'weights': 'uniform'}
0.544 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 5,
'weights': 'distance'}
0.579 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 6,
'weights': 'uniform'}
0.556 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 6,
'weights': 'distance'}
0.561 (0.022) with: {'metric': 'euclidean', 'n_neighbors': 7,
'weights': 'uniform'}
0.564 (0.018) with: {'metric': 'euclidean', 'n_neighbors': 7,
'weights': 'distance'}
0.580 (0.028) with: {'metric': 'euclidean', 'n_neighbors': 8,
'weights': 'uniform'}
0.564 (0.027) with: {'metric': 'euclidean', 'n_neighbors': 8,
'weights': 'distance'}
```

```
0.560 (0.024) with: {'metric': 'euclidean', 'n_neighbors': 9,
'weights': 'uniform'}
0.569 (0.026) with: {'metric': 'euclidean', 'n_neighbors': 9,
'weights': 'distance'}
0.585 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 10,
'weights': 'uniform'}
0.566 (0.027) with: {'metric': 'euclidean', 'n_neighbors': 10,
'weights': 'distance'}
0.557 (0.021) with: {'metric': 'euclidean', 'n_neighbors': 11,
'weights': 'uniform'}
0.566 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 11,
'weights': 'distance'}
0.584 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 12,
'weights': 'uniform'}
0.563 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 12,
'weights': 'distance'}
0.565 (0.019) with: {'metric': 'euclidean', 'n_neighbors': 13,
'weights': 'uniform'}
0.561 (0.027) with: {'metric': 'euclidean', 'n_neighbors': 13,
'weights': 'distance'}
0.588 (0.010) with: {'metric': 'euclidean', 'n_neighbors': 14,
'weights': 'uniform'}
0.569 (0.024) with: {'metric': 'euclidean', 'n_neighbors': 14,
'weights': 'distance'}
0.581 (0.013) with: {'metric': 'euclidean', 'n_neighbors': 15,
'weights': 'uniform'}
0.574 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 15,
'weights': 'distance'}
0.590 (0.011) with: {'metric': 'euclidean', 'n_neighbors': 16,
'weights': 'uniform'}
0.575 (0.024) with: {'metric': 'euclidean', 'n_neighbors': 16,
'weights': 'distance'}
0.578 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 17,
'weights': 'uniform'}
0.572 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 17,
'weights': 'distance'}
0.590 (0.012) with: {'metric': 'euclidean', 'n_neighbors': 18,
'weights': 'uniform'}
0.580 (0.023) with: {'metric': 'euclidean', 'n_neighbors': 18,
'weights': 'distance'}
0.586 (0.012) with: {'metric': 'euclidean', 'n_neighbors': 19,
'weights': 'uniform'}
0.585 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 19,
'weights': 'distance'}
0.595 (0.013) with: {'metric': 'euclidean', 'n_neighbors': 20,
'weights': 'uniform'}
0.581 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 20,
'weights': 'distance'}
0.589 (0.014) with: {'metric': 'euclidean', 'n_neighbors': 21,
'weights': 'uniform'}
```

```
0.585 (0.019) with: {'metric': 'euclidean', 'n_neighbors': 21,
'weights': 'distance'}
0.598 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 22,
'weights': 'uniform'}
0.586 (0.021) with: {'metric': 'euclidean', 'n_neighbors': 22,
'weights': 'distance'}
0.592 (0.014) with: {'metric': 'euclidean', 'n_neighbors': 23,
'weights': 'uniform'}
0.587 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 23,
'weights': 'distance'}
0.598 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 24,
'weights': 'uniform'}
0.587 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 24,
'weights': 'distance'}
0.590 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 25,
'weights': 'uniform'}
0.587 (0.018) with: {'metric': 'euclidean', 'n_neighbors': 25,
'weights': 'distance'}
0.600 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 26,
'weights': 'uniform'}
0.584 (0.018) with: {'metric': 'euclidean', 'n_neighbors': 26,
'weights': 'distance'}
0.590 (0.013) with: {'metric': 'euclidean', 'n_neighbors': 27,
'weights': 'uniform'}
0.588 (0.019) with: {'metric': 'euclidean', 'n_neighbors': 27,
'weights': 'distance'}
0.595 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 28,
'weights': 'uniform'}
0.583 (0.019) with: {'metric': 'euclidean', 'n_neighbors': 28,
'weights': 'distance'}
0.589 (0.018) with: {'metric': 'euclidean', 'n_neighbors': 29,
'weights': 'uniform'}
0.590 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 29,
'weights': 'distance'}
0.590 (0.015) with: {'metric': 'euclidean', 'n_neighbors': 30,
'weights': 'uniform'}
0.588 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 30,
'weights': 'distance'}
0.534 (0.031) with: {'metric': 'manhattan', 'n_neighbors': 1,
'weights': 'uniform'}
0.534 (0.031) with: {'metric': 'manhattan', 'n_neighbors': 1,
'weights': 'distance'}
0.589 (0.017) with: {'metric': 'manhattan', 'n_neighbors': 2,
'weights': 'uniform'}
0.534 (0.031) with: {'metric': 'manhattan', 'n_neighbors': 2,
'weights': 'distance'}
0.550 (0.018) with: {'metric': 'manhattan', 'n_neighbors': 3,
'weights': 'uniform'}
0.544 (0.018) with: {'metric': 'manhattan', 'n_neighbors': 3,
'weights': 'distance'}
```

```
0.577 (0.019) with: {'metric': 'manhattan', 'n_neighbors': 4,
'weights': 'uniform'}
0.555 (0.020) with: {'metric': 'manhattan', 'n_neighbors': 4,
'weights': 'distance'}
0.556 (0.020) with: {'metric': 'manhattan', 'n_neighbors': 5,
'weights': 'uniform'}
0.556 (0.016) with: {'metric': 'manhattan', 'n_neighbors': 5,
'weights': 'distance'}
0.585 (0.020) with: {'metric': 'manhattan', 'n_neighbors': 6,
'weights': 'uniform'}
0.564 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 6,
'weights': 'distance'}
0.571 (0.016) with: {'metric': 'manhattan', 'n_neighbors': 7,
'weights': 'uniform'}
0.571 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 7,
'weights': 'distance'}
0.586 (0.019) with: {'metric': 'manhattan', 'n_neighbors': 8,
'weights': 'uniform'}
0.571 (0.020) with: {'metric': 'manhattan', 'n_neighbors': 8,
'weights': 'distance'}
0.571 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 9,
'weights': 'uniform'}
0.574 (0.025) with: {'metric': 'manhattan', 'n_neighbors': 9,
'weights': 'distance'}
0.581 (0.014) with: {'metric': 'manhattan', 'n_neighbors': 10,
'weights': 'uniform'}
0.571 (0.024) with: {'metric': 'manhattan', 'n_neighbors': 10,
'weights': 'distance'}
0.562 (0.023) with: {'metric': 'manhattan', 'n_neighbors': 11,
'weights': 'uniform'}
0.575 (0.026) with: {'metric': 'manhattan', 'n_neighbors': 11,
'weights': 'distance'}
0.582 (0.019) with: {'metric': 'manhattan', 'n_neighbors': 12,
'weights': 'uniform'}
0.570 (0.031) with: {'metric': 'manhattan', 'n_neighbors': 12,
'weights': 'distance'}
0.572 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 13,
'weights': 'uniform'}
0.571 (0.030) with: {'metric': 'manhattan', 'n_neighbors': 13,
'weights': 'distance'}
0.583 (0.012) with: {'metric': 'manhattan', 'n_neighbors': 14,
'weights': 'uniform'}
0.569 (0.024) with: {'metric': 'manhattan', 'n_neighbors': 14,
'weights': 'distance'}
0.571 (0.017) with: {'metric': 'manhattan', 'n_neighbors': 15,
'weights': 'uniform'}
0.568 (0.023) with: {'metric': 'manhattan', 'n_neighbors': 15,
'weights': 'distance'}
0.585 (0.010) with: {'metric': 'manhattan', 'n_neighbors': 16,
'weights': 'uniform'}
```

```
0.574 (0.030) with: {'metric': 'manhattan', 'n_neighbors': 16,
'weights': 'distance'}
0.579 (0.017) with: {'metric': 'manhattan', 'n_neighbors': 17,
'weights': 'uniform'}
0.585 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 17,
'weights': 'distance'}
0.593 (0.011) with: {'metric': 'manhattan', 'n_neighbors': 18,
'weights': 'uniform'}
0.587 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 18,
'weights': 'distance'}
0.590 (0.016) with: {'metric': 'manhattan', 'n_neighbors': 19,
'weights': 'uniform'}
0.592 (0.022) with: {'metric': 'manhattan', 'n_neighbors': 19,
'weights': 'distance'}
0.597 (0.008) with: {'metric': 'manhattan', 'n_neighbors': 20,
'weights': 'uniform'}
0.589 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 20,
'weights': 'distance'}
0.592 (0.012) with: {'metric': 'manhattan', 'n_neighbors': 21,
'weights': 'uniform'}
0.592 (0.020) with: {'metric': 'manhattan', 'n_neighbors': 21,
'weights': 'distance'}
0.603 (0.013) with: {'metric': 'manhattan', 'n_neighbors': 22,
'weights': 'uniform'}
0.588 (0.022) with: {'metric': 'manhattan', 'n_neighbors': 22,
'weights': 'distance'}
0.595 (0.015) with: {'metric': 'manhattan', 'n_neighbors': 23,
'weights': 'uniform'}
0.592 (0.019) with: {'metric': 'manhattan', 'n_neighbors': 23,
'weights': 'distance'}
0.599 (0.015) with: {'metric': 'manhattan', 'n_neighbors': 24,
'weights': 'uniform'}
0.594 (0.018) with: {'metric': 'manhattan', 'n_neighbors': 24,
'weights': 'distance'}
0.592 (0.013) with: {'metric': 'manhattan', 'n_neighbors': 25,
'weights': 'uniform'}
0.596 (0.019) with: {'metric': 'manhattan', 'n_neighbors': 25,
'weights': 'distance'}
0.595 (0.017) with: {'metric': 'manhattan', 'n_neighbors': 26,
'weights': 'uniform'}
0.593 (0.016) with: {'metric': 'manhattan', 'n_neighbors': 26,
'weights': 'distance'}
0.593 (0.015) with: {'metric': 'manhattan', 'n_neighbors': 27,
'weights': 'uniform'}
0.598 (0.021) with: {'metric': 'manhattan', 'n_neighbors': 27,
'weights': 'distance'}
0.598 (0.016) with: {'metric': 'manhattan', 'n_neighbors': 28,
'weights': 'uniform'}
0.597 (0.017) with: {'metric': 'manhattan', 'n_neighbors': 28,
'weights': 'distance'}
```

```
0.592 (0.013) with: {'metric': 'manhattan', 'n_neighbors': 29,
'weights': 'uniform'}
0.599 (0.017) with: {'metric': 'manhattan', 'n_neighbors': 29,
'weights': 'distance'}
0.597 (0.011) with: {'metric': 'manhattan', 'n_neighbors': 30,
'weights': 'uniform'}
0.595 (0.016) with: {'metric': 'manhattan', 'n_neighbors': 30,
'weights': 'distance'}
0.536 (0.029) with: {'metric': 'minkowski', 'n_neighbors': 1,
'weights': 'uniform'}
0.536 (0.029) with: {'metric': 'minkowski', 'n_neighbors': 1,
'weights': 'distance'}
0.579 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 2,
'weights': 'uniform'}
0.536 (0.029) with: {'metric': 'minkowski', 'n_neighbors': 2,
'weights': 'distance'}
0.542 (0.023) with: {'metric': 'minkowski', 'n_neighbors': 3,
'weights': 'uniform'}
0.542 (0.024) with: {'metric': 'minkowski', 'n_neighbors': 3,
'weights': 'distance'}
0.574 (0.017) with: {'metric': 'minkowski', 'n_neighbors': 4,
'weights': 'uniform'}
0.542 (0.016) with: {'metric': 'minkowski', 'n_neighbors': 4,
'weights': 'distance'}
0.545 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 5,
'weights': 'uniform'}
0.544 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 5,
'weights': 'distance'}
0.579 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 6,
'weights': 'uniform'}
0.556 (0.025) with: {'metric': 'minkowski', 'n_neighbors': 6,
'weights': 'distance'}
0.561 (0.022) with: {'metric': 'minkowski', 'n_neighbors': 7,
'weights': 'uniform'}
0.564 (0.018) with: {'metric': 'minkowski', 'n_neighbors': 7,
'weights': 'distance'}
0.580 (0.028) with: {'metric': 'minkowski', 'n_neighbors': 8,
'weights': 'uniform'}
0.564 (0.027) with: {'metric': 'minkowski', 'n_neighbors': 8,
'weights': 'distance'}
0.560 (0.024) with: {'metric': 'minkowski', 'n_neighbors': 9,
'weights': 'uniform'}
0.569 (0.026) with: {'metric': 'minkowski', 'n_neighbors': 9,
'weights': 'distance'}
0.585 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 10,
'weights': 'uniform'}
0.566 (0.027) with: {'metric': 'minkowski', 'n_neighbors': 10,
'weights': 'distance'}
0.557 (0.021) with: {'metric': 'minkowski', 'n_neighbors': 11,
'weights': 'uniform'}
```

```
0.566 (0.025) with: {'metric': 'minkowski', 'n_neighbors': 11,
'weights': 'distance'}
0.584 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 12,
'weights': 'uniform'}
0.563 (0.025) with: {'metric': 'minkowski', 'n_neighbors': 12,
'weights': 'distance'}
0.565 (0.019) with: {'metric': 'minkowski', 'n_neighbors': 13,
'weights': 'uniform'}
0.561 (0.027) with: {'metric': 'minkowski', 'n_neighbors': 13,
'weights': 'distance'}
0.588 (0.010) with: {'metric': 'minkowski', 'n_neighbors': 14,
'weights': 'uniform'}
0.569 (0.024) with: {'metric': 'minkowski', 'n_neighbors': 14,
'weights': 'distance'}
0.581 (0.013) with: {'metric': 'minkowski', 'n_neighbors': 15,
'weights': 'uniform'}
0.574 (0.025) with: {'metric': 'minkowski', 'n_neighbors': 15,
'weights': 'distance'}
0.590 (0.011) with: {'metric': 'minkowski', 'n_neighbors': 16,
'weights': 'uniform'}
0.575 (0.024) with: {'metric': 'minkowski', 'n_neighbors': 16,
'weights': 'distance'}
0.578 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 17,
'weights': 'uniform'}
0.572 (0.025) with: {'metric': 'minkowski', 'n_neighbors': 17,
'weights': 'distance'}
0.590 (0.012) with: {'metric': 'minkowski', 'n_neighbors': 18,
'weights': 'uniform'}
0.580 (0.023) with: {'metric': 'minkowski', 'n_neighbors': 18,
'weights': 'distance'}
0.586 (0.012) with: {'metric': 'minkowski', 'n_neighbors': 19,
'weights': 'uniform'}
0.585 (0.025) with: {'metric': 'minkowski', 'n_neighbors': 19,
'weights': 'distance'}
0.595 (0.013) with: {'metric': 'minkowski', 'n_neighbors': 20,
'weights': 'uniform'}
0.581 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 20,
'weights': 'distance'}
0.589 (0.014) with: {'metric': 'minkowski', 'n_neighbors': 21,
'weights': 'uniform'}
0.585 (0.019) with: {'metric': 'minkowski', 'n_neighbors': 21,
'weights': 'distance'}
0.598 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 22,
'weights': 'uniform'}
0.586 (0.021) with: {'metric': 'minkowski', 'n_neighbors': 22,
'weights': 'distance'}
0.592 (0.014) with: {'metric': 'minkowski', 'n_neighbors': 23,
'weights': 'uniform'}
0.587 (0.016) with: {'metric': 'minkowski', 'n_neighbors': 23,
'weights': 'distance'}
```

```
0.598 (0.016) with: {'metric': 'minkowski', 'n_neighbors': 24,
'weights': 'uniform'}
0.587 (0.016) with: {'metric': 'minkowski', 'n_neighbors': 24,
'weights': 'distance'}
0.590 (0.016) with: {'metric': 'minkowski', 'n_neighbors': 25,
'weights': 'uniform'}
0.587 (0.018) with: {'metric': 'minkowski', 'n_neighbors': 25,
'weights': 'distance'}
0.600 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 26,
'weights': 'uniform'}
0.584 (0.018) with: {'metric': 'minkowski', 'n_neighbors': 26,
'weights': 'distance'}
0.590 (0.013) with: {'metric': 'minkowski', 'n_neighbors': 27,
'weights': 'uniform'}
0.588 (0.019) with: {'metric': 'minkowski', 'n_neighbors': 27,
'weights': 'distance'}
0.595 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 28,
'weights': 'uniform'}
0.583 (0.019) with: {'metric': 'minkowski', 'n_neighbors': 28,
'weights': 'distance'}
0.589 (0.018) with: {'metric': 'minkowski', 'n_neighbors': 29,
'weights': 'uniform'}
0.590 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 29,
'weights': 'distance'}
0.590 (0.015) with: {'metric': 'minkowski', 'n_neighbors': 30,
'weights': 'uniform'}
0.588 (0.020) with: {'metric': 'minkowski', 'n_neighbors': 30,
'weights': 'distance'}

C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning
```

```
from sklearn.metrics import  make_scorer
from sklearn.model_selection import cross_val_score

def classification_report_with_accuracy_score(Y_test, y_pred2):
    print (classification_report(Y_test, y_pred2)) # print
classification report
    return accuracy_score(Y_test, y_pred2) # return accuracy score


nested_score = cross_val_score(grid_search_knn, X=X_train, y=Y_train,
cv=cv,

scoring=make_scorer(classification_report_with_accuracy_score))
print (nested_score)
```

```
C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.61      0.90      0.73       159
           1       0.43      0.12      0.18       103

    accuracy                           0.59       262
   macro avg       0.52      0.51      0.46       262
weighted avg       0.54      0.59      0.51       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.62      0.94      0.74       159
           1       0.52      0.11      0.18       103

    accuracy                           0.61       262
   macro avg       0.57      0.52      0.46       262
weighted avg       0.58      0.61      0.52       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.62      0.92      0.74       159
           1       0.50      0.13      0.20       103

    accuracy                           0.61       262
   macro avg       0.56      0.52      0.47       262
weighted avg       0.57      0.61      0.53       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning
```

```
              precision    recall  f1-score   support

           0       0.60      0.91      0.72       159
           1       0.32      0.07      0.11       103

    accuracy                           0.58       262
   macro avg       0.46      0.49      0.42       262
weighted avg       0.49      0.58      0.48       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.61      0.88      0.72       160
           1       0.34      0.10      0.15       102

    accuracy                           0.58       262
   macro avg       0.47      0.49      0.44       262
weighted avg       0.50      0.58      0.50       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.61      0.85      0.71       160
           1       0.40      0.16      0.23       102

    accuracy                           0.58       262
   macro avg       0.51      0.50      0.47       262
weighted avg       0.53      0.58      0.52       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.61      0.93      0.74       160
           1       0.39      0.07      0.12       102
```

```
    accuracy                           0.60       262
   macro avg       0.50      0.50      0.43       262
weighted avg       0.52      0.60      0.50       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.60      0.85      0.70       160
           1       0.31      0.11      0.16       102

    accuracy                           0.56       262
   macro avg       0.46      0.48      0.43       262
weighted avg       0.49      0.56      0.49       262


C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning

              precision    recall  f1-score   support

           0       0.62      0.93      0.74       160
           1       0.48      0.11      0.18       102

    accuracy                           0.61       262
   macro avg       0.55      0.52      0.46       262
weighted avg       0.56      0.61      0.52       262

              precision    recall  f1-score   support

           0       0.62      0.89      0.73       160
           1       0.45      0.14      0.21       102

    accuracy                           0.60       262
   macro avg       0.54      0.52      0.47       262
weighted avg       0.55      0.60      0.53       262

[0.59160305 0.61068702 0.60687023 0.57633588 0.57633588 0.58015267
 0.59541985 0.5610687  0.60687023 0.59923664]

C:\Users\thero\Anaconda3\lib\site-packages\sklearn\model_selection\
_search.py:849: FutureWarning: The parameter 'iid' is deprecated in
0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning
```

```
knn_y_predicted = grid_search_knn.predict(X_test)
```

```
knn_y_predicted
```

```
array([0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
dtype=int64)
```

```
knn_grid_score=accuracy_score(Y_test, knn_y_predicted)
```

```
knn_grid_score
```

```
0.6173780487804879
```

```
grid_search_knn.best_params_
```

```
{'metric': 'manhattan', 'n_neighbors': 22, 'weights': 'uniform'}
```

```
confusion_matrix(Y_test, knn_y_predicted)
```

```
array([[376,  26],
       [225,  29]], dtype=int64)
```

Prediction on only one set of data

```
X_KNN=knn.predict([[5.735724,
158.318741,25363.016594,7.728601,377.543291,568.304671,13.626624,75.95
2337,4.732954]])
```

```
X_KNN
```

```
array([0], dtype=int64)
```