In [26]:
```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import VotingClassifier, BaggingClassifier, Rando
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.datasets import load_breast_cancer

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

In [2]:
```
dataset = load_breast_cancer()
df = pd.DataFrame(dataset.data,columns=dataset.feature_names)
df['target'] = dataset.target
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

The dataset has 569 rows and 31 features with 30 features are floats and 1 target feature of integer.

In [4]:
```
df.describe(include="all").T
```

Out[4]:

	count	mean	std	min	25%	50%	75'
mean radius	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.7800C
mean texture	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.8000C
mean perimeter	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.1000C
mean area	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.7000C
mean smoothness	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.1053C
mean compactness	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.1304C
mean concavity	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.1307C
mean concave points	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.0740C
mean symmetry	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.1957C
mean fractal dimension	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.06612
radius error	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.4789C
texture error	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.4740C
perimeter error	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.3570C
area error	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.1900C
smoothness error	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.00814
compactness error	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.03245
concavity error	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.04205
concave points error	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.01471
symmetry error	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.02348
fractal dimension error	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.00455
worst radius	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.7900C
worst texture	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.7200C
worst perimeter	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.4000C
worst area	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.0000C
worst smoothness	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.1460C
worst compactness	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.3391C
worst concavity	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.3829C

	count	mean	std	min	25%	50%	75'
worst concave points	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.1614C
worst symmetry	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.3179C
worst fractal dimension	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.0920ε
target	569.0	0.627417	0.483918	0.000000	0.000000	1.000000	1.0000C

In [5]:

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

Out[5]:

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [6]:
```
df
```

Out[6]:

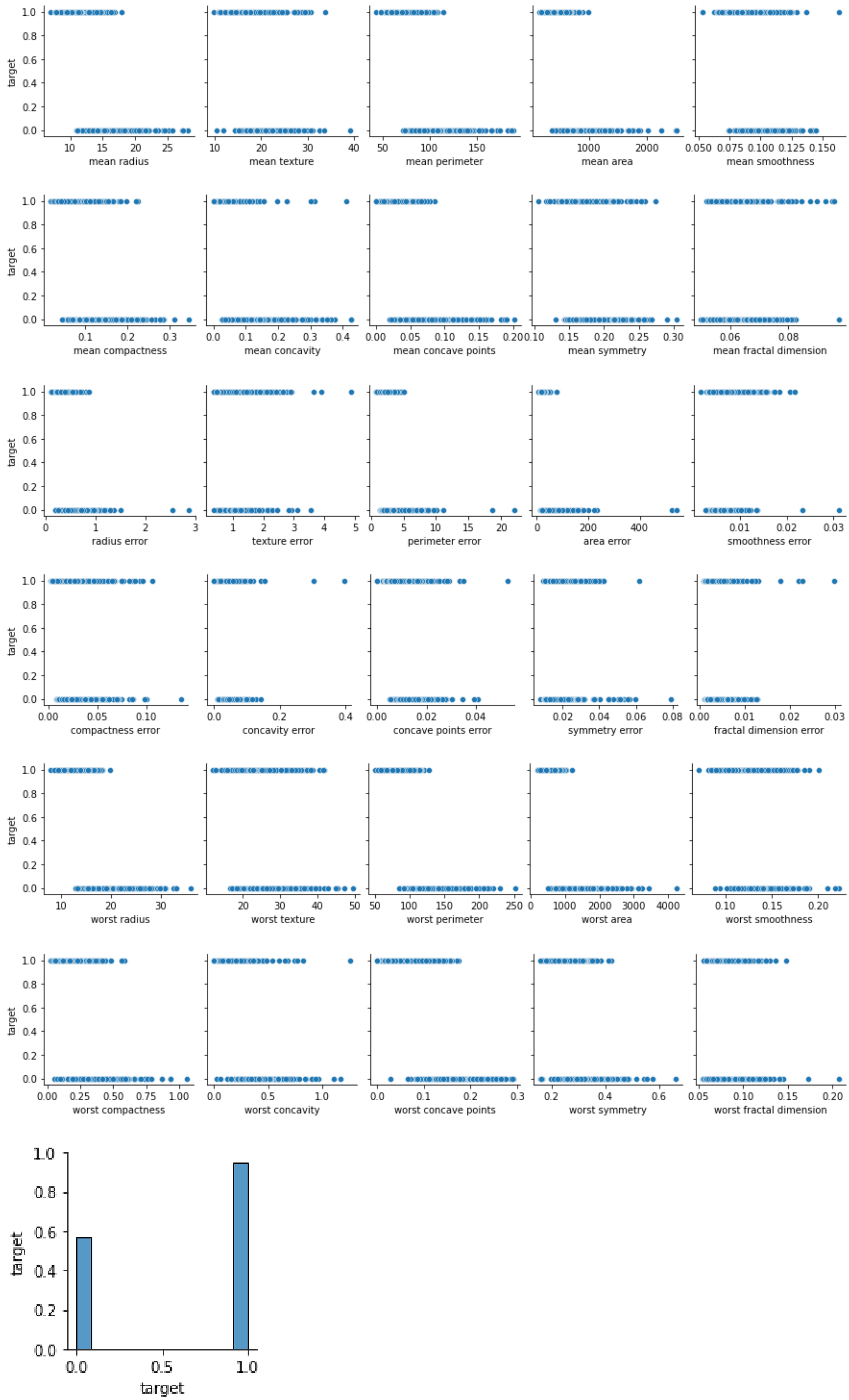
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sym
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0

569 rows × 31 columns

In [7]:
```
s = pd.value_counts(df.target)

# For class 0
num_benign = s[0]
# For class 1
num_malign = s[1]
total_cases = len(df)

percent_b = num_benign / total_cases
percent_m = num_malign / total_cases

print("Distribution between Benign and Malignant\nPercent Benign: {0:.3
```

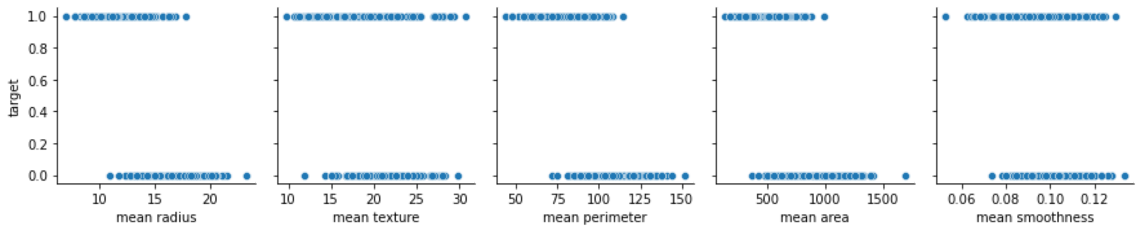
```
Distribution between Benign and Malignant
Percent Benign: 0.373
Percent Malignant: 0.627
```

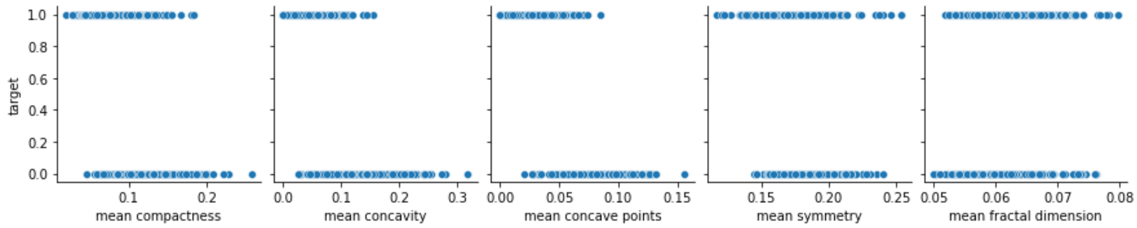
In [8]:
```
for i in range(0, len(df.columns),5):
    sns.pairplot(data=df,x_vars=df.columns[i:i+5],y_vars=['target'])
```

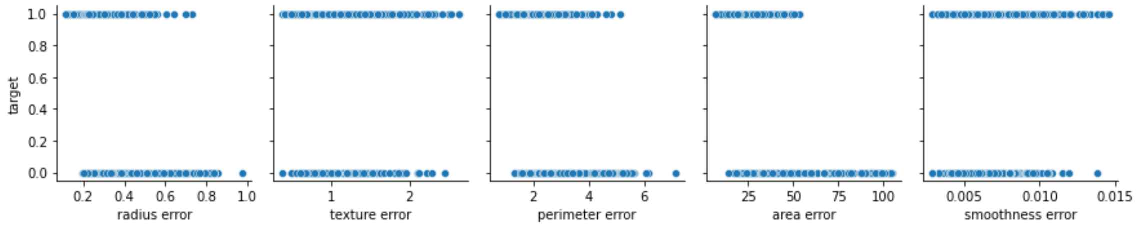


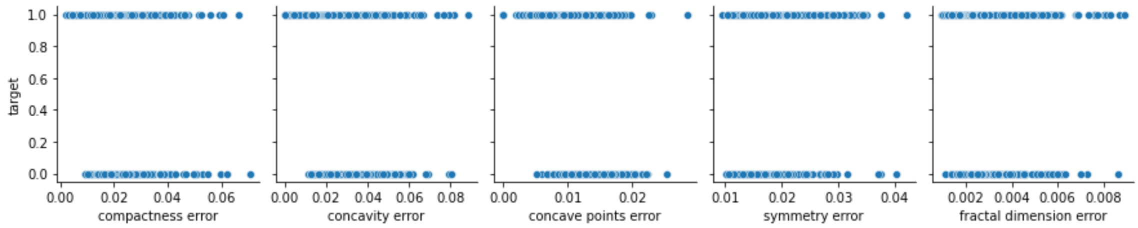
In [9]:
```
q1 = df.quantile(0.2)
q3 = df.quantile(0.8)
iqr = q3-q1

df= df[~((df<(q1 - 1.5 * iqr)) | (df > (q3 + 1.5 * iqr ))).any(axis=1)]
```

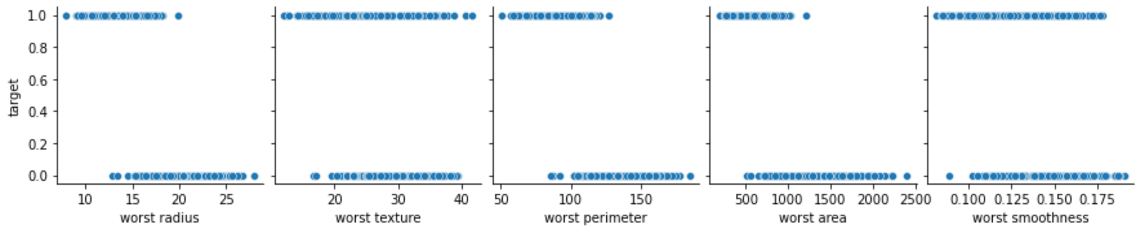
In [10]:
```python
for i in range(0, len(df.columns), 5):
    sns.pairplot(data=df, x_vars=df.columns[i:i+5], y_vars=['target'])
```

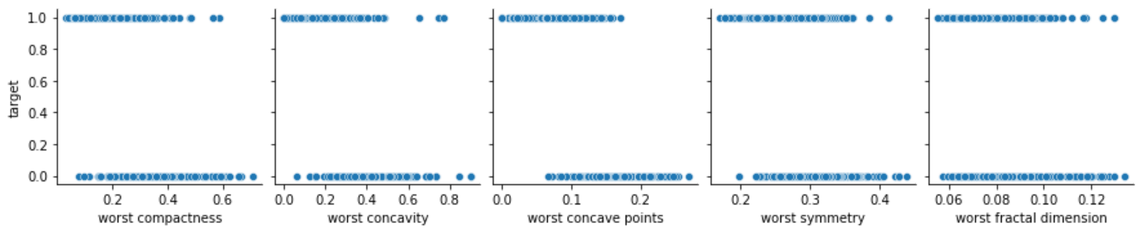


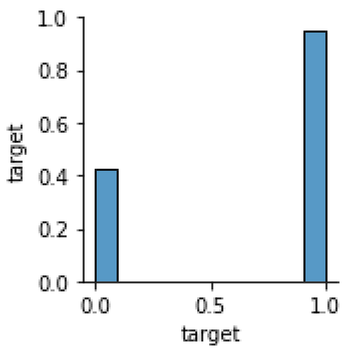






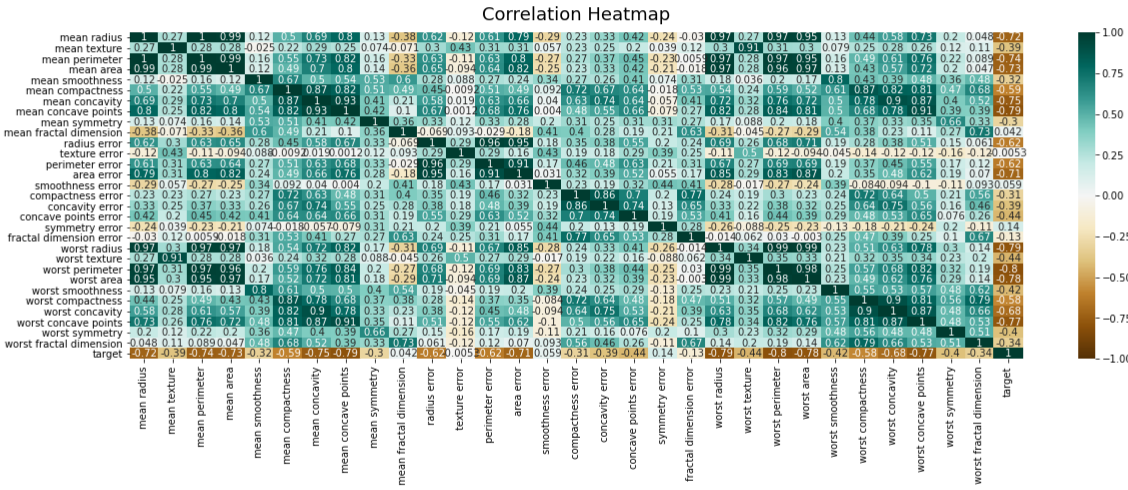






In [11]:

```
plt.figure(figsize=(20,6))
heatmap=sns.heatmap(df.corr(),vmin=-1, vmax=1, annot=True,cmap='BrBG')
heatmap.set_title('Correlation Heatmap', fontdict = {'fontsize':18},pac
```



From the heatmap, mean radius, mean perimeter, mean concave points, radius error, perimeter error, concave points error, worst radius, worst texture, worst perimeter, worst area has a correlation score that exceed the p-value of 0.8. Therefore, the features are excluded.

In [12]:

```
features = list(df.columns)
features = ['mean texture','mean area','mean smoothness','mean compactr
            'mean fractal dimension','texture error','area error','smoo
            'concavity error','concave points error','symmetry error','
            'worst compactness','worst concavity','worst concave points
X = df[features]
```

In [13]:

```
X.head()
```

Out[13]:

	mean texture	mean area	mean smoothness	mean compactness	mean concavity	mean symmetry	mean fractal dimension	texture error	are erro
1	17.77	1326.0	0.08474	0.07864	0.0869	0.1812	0.05667	0.7339	74.0
2	21.25	1203.0	0.10960	0.15990	0.1974	0.2069	0.05999	0.7869	94.0
4	14.34	1297.0	0.10030	0.13280	0.1980	0.1809	0.05883	0.7813	94.4
5	15.70	477.1	0.12780	0.17000	0.1578	0.2087	0.07613	0.8902	27.1
6	19.98	1040.0	0.09463	0.10900	0.1127	0.1794	0.05742	0.7732	53.9

5 rows × 21 columns

In [14]:

```
# Normalization:
X = (X - np.min(X))/ (np.max(X) - np.min(X))
y = df['target']
```

In [15]:
```python
# Prepare training data for building the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0

```

In [16]:
```python
# LogReg Model
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
print(classification_report(y_test, lr_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.83      0.91        30
           1       0.93      1.00      0.96        64

    accuracy                           0.95        94
   macro avg       0.96      0.92      0.94        94
weighted avg       0.95      0.95      0.95        94
```

In [17]:
```python
knn = KNeighborsClassifier()
knn_model = knn.fit(X_train,y_train)
knn_pred = knn.predict(X_test)
print(classification_report(y_test,knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.83      0.89        30
           1       0.93      0.98      0.95        64

    accuracy                           0.94        94
   macro avg       0.94      0.91      0.92        94
weighted avg       0.94      0.94      0.93        94
```

In [20]:
```python
gnb = GaussianNB()
gnb_model = gnb.fit(X_train,y_train)
gnb_pred = gnb.predict(X_test)
print(classification_report(y_test,gnb_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87        30
           1       0.95      0.92      0.94        64

    accuracy                           0.91        94
   macro avg       0.90      0.91      0.90        94
weighted avg       0.92      0.91      0.92        94
```

In [21]:
```
svm = SVC()
svm_model = svm.fit(X_train,y_train)
svm_pred = svm.predict(X_test)
print(classification_report(y_test,svm_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        30
           1       0.95      0.98      0.97        64

    accuracy                           0.96        94
   macro avg       0.96      0.94      0.95        94
weighted avg       0.96      0.96      0.96        94
```

Voting Classifier

In [23]:
```
models = [('svm', svm),('KNN',knn), ('GaussianNB',gnb)]
hv = VotingClassifier(models, voting='hard')

hv_model = hv.fit(X_train, y_train)
hv_pred = hv.predict(X_test)
print(classification_report(y_test,hv_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        30
           1       0.95      0.98      0.97        64

    accuracy                           0.96        94
   macro avg       0.96      0.94      0.95        94
weighted avg       0.96      0.96      0.96        94
```

In [32]:
```
bagging = BaggingClassifier()
bagging.fit(X_train, y_train)
bag_pred = bagging.predict(X_test)
print(classification_report(y_test, bag_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        30
           1       0.98      0.95      0.97        64

    accuracy                           0.96        94
   macro avg       0.95      0.96      0.95        94
weighted avg       0.96      0.96      0.96        94
```

In [34]:

```python
start = time.time()
param_dist = {'max_depth': [2, 3, 4],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2', None],
              'criterion': ['gini', 'entropy']}
fit_rf = RandomForestClassifier()
cv_rf = GridSearchCV(fit_rf, cv=10, param_grid = param_dist, n_jobs = 

cv_rf.fit(X_train, y_train)
print('Best Parameters using grid search: \n', cv_rf.best_params_)
end = time.time()
print('Time taken in grid search: {0: .2f}'.format(end-start))
```

```
Best Parameters using grid search:
 {'bootstrap': False, 'criterion': 'gini', 'max_depth': 4, 'max_features':
'sqrt'}
Time taken in grid search:  25.41
```

In [37]:

```python
rf_pred = cv_rf.predict(X_test)
print(classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        30
           1       0.95      0.95      0.95        64

    accuracy                           0.94        94
   macro avg       0.93      0.93      0.93        94
weighted avg       0.94      0.94      0.94        94
```

In [38]:

```python
xgb = XGBClassifier()
xgb_model = xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
print(classification_report(y_test,xgb_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        30
           1       0.98      0.95      0.97        64

    accuracy                           0.96        94
   macro avg       0.95      0.96      0.95        94
weighted avg       0.96      0.96      0.96        94
```

Summary

Voting Classifier is the best model out of the 3 models. The three models has the same f1-score which is 0.97, so we look to precision and recall values. Out of the three models, voting classifier achieved the best score between precision and recall. Voting classifier scores 0.95 which is the lowest of out the three models, however the recall score of 0.98 which scored highest among all 3 models, improving the quality of the model. Hence, The best model is the Voting Classifier model out of all 3 models. The three models has same weighted avg score, scoring 0.96 on precision, recall, and f1-score, therefore it cant be used for comparison.

In [ ]:

```
1
```