In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn import linear_model, tree, ensemble
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import *
import warnings
warnings.filterwarnings("ignore")
```

First I started by simply importing all of the libraries needed and the data. I just went through by first starting by looking at the head of the data and then seing if there was any duplicated data and how much of the data was missing.

In [2]:

```
train = pd.read_csv("C:/Users/Antonio/Downloads/titanictrain.csv")
test = pd.read_csv("C:/Users/Antonio/Downloads/titanictest.csv")

display(train.head())
display(test.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [3]:

```
p = (train.isna().sum()/len(train)*100).sort_values(ascending=False)
print(f"Train data has {train.duplicated().sum()} duplicated data")
print(f"Test data has {test.duplicated().sum()} duplicated data")
print(p)
```

```
Train data has 0 duplicated data
Test data has 0 duplicated data
Cabin          77.104377
Age            19.865320
Embarked        0.224467
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
dtype: float64
```

In [4]:

```
train.info()
```

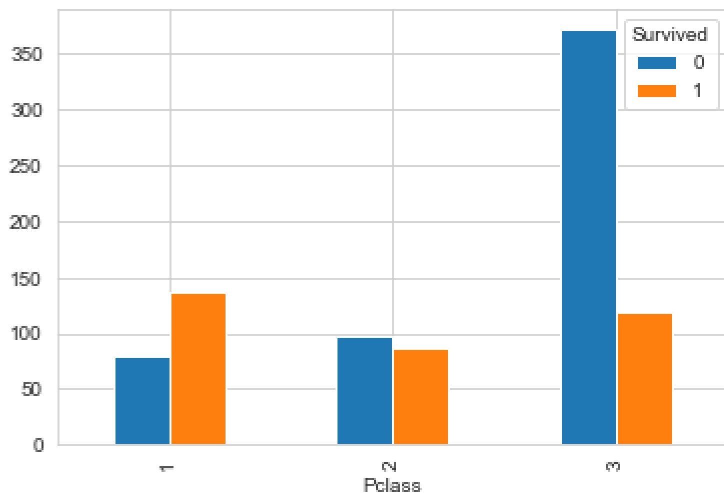
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
```

```
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [5]:
```
crossTabResult = pd.crosstab(index=train['Pclass'],columns=train['Survived'])
print(crossTabResult)
crossTabResult.plot.bar()
```

```
Survived    0    1
Pclass
1          80  136
2          97   87
3         372  119
```

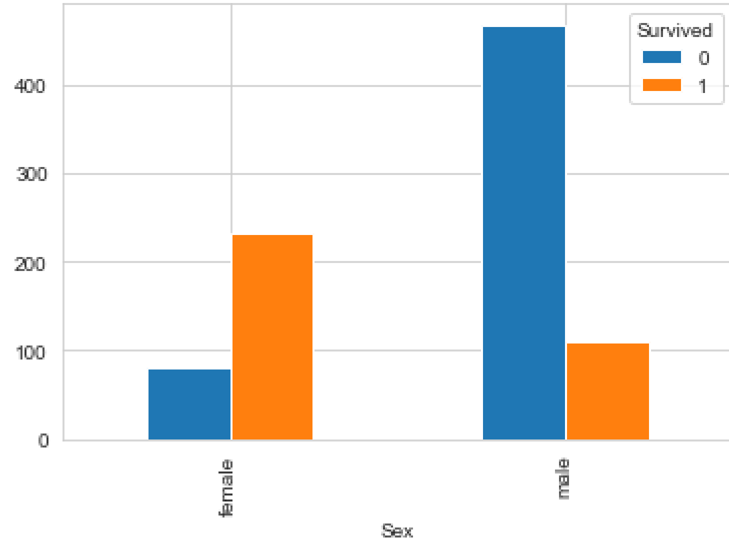
Out[5]: <AxesSubplot:xlabel='Pclass'>



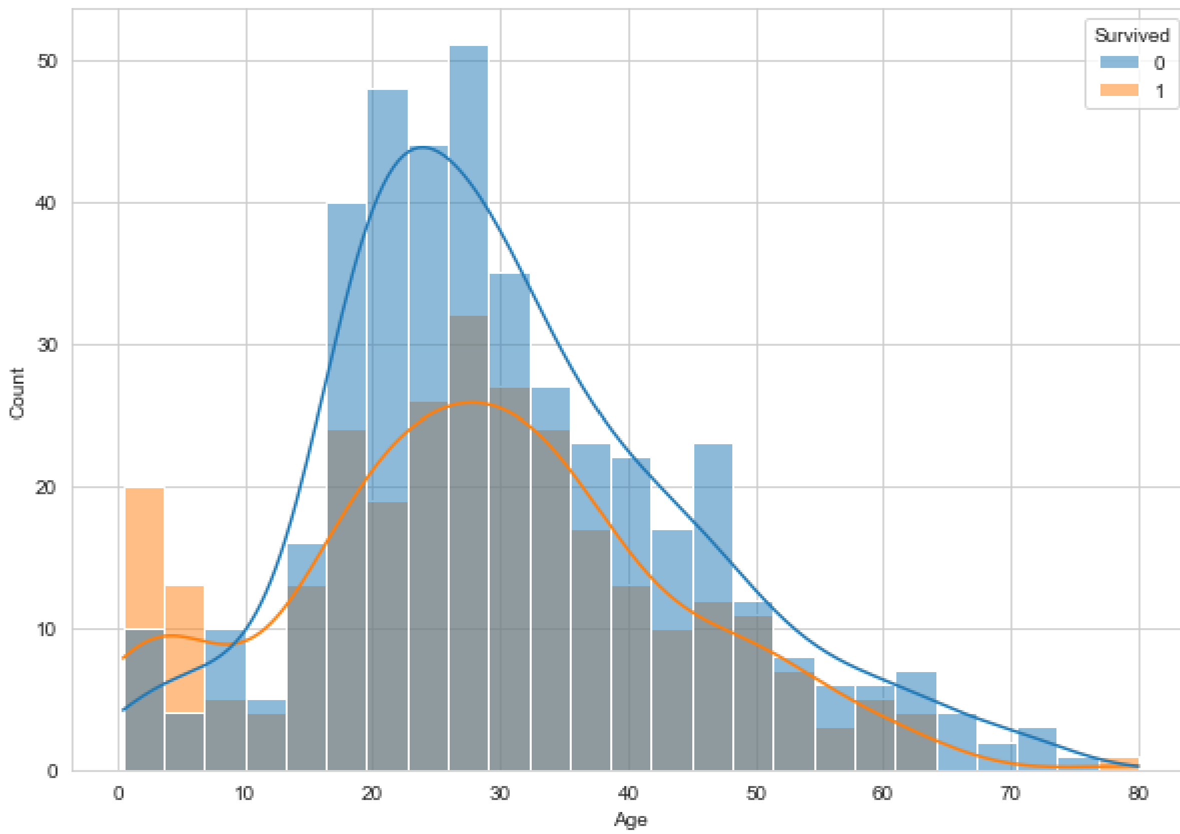
In [6]:
```
crossTabResult = pd.crosstab(index=train['Sex'],columns=train['Survived'])
print(crossTabResult)
crossTabResult.plot.bar()
```

```
Survived    0    1
Sex
female     81  233
male      468  109
```

Out[6]: <AxesSubplot:xlabel='Sex'>



In [7]:
```
fig, ax1 = plt.subplots(figsize=(10, 7))
sns.histplot(data = train, x='Age', ax=ax1, bins=25, hue='Survived', kde=True)
plt.show()
```



Here I'm just dropping uneeeded data types suich as the passengerid, name, cabin and ticket. Instead of using the name we pulled the title off them and used that instead creating a new feature and then using the mean age of that title to fill in any of the missing ages. Same thing with the fare just using the mean of the class to fill in any missing data.

In [8]:
```
train.drop("PassengerId", axis=1, inplace=True)
test.drop("PassengerId", axis=1, inplace=True)
```

In [9]:
```
train["Title"] = train["Name"].str.extract('([A-Za-z]+)\.')
```

```
test["Title"] = test["Name"].str.extract('([A-Za-z]+)\.')

train["Title"].value_counts()
```

Out[9]:
```
Mr          517
Miss        182
Mrs         125
Master       40
Dr            7
Rev           6
Mlle          2
Major         2
Col           2
Countess      1
Capt          1
Ms            1
Sir           1
Lady          1
Mme           1
Don           1
Jonkheer      1
Name: Title, dtype: int64
```

In [10]:
```
def convert_title(title):
    if title in ["Ms", "Mile", "Miss"]:
        return "Miss"
    elif title in ["Mme", "Mrs"]:
        return "Mrs"
    elif title == "Mr":
        return "Mr"
    elif title == "Master":
        return "Master"
    else:
        return "Other"

train["Title"] = train["Title"].map(convert_title)
test["Title"] = test["Title"].map(convert_title)

train["Title"].value_counts()
```

Out[10]:
```
Mr        517
Miss      183
Mrs       126
Master     40
Other      25
Name: Title, dtype: int64
```

In [11]:
```
train.drop("Name", axis=1, inplace=True)
test.drop("Name", axis=1, inplace=True)
```

In [12]:
```
train.drop("Cabin", axis=1, inplace=True)
test.drop("Cabin", axis=1, inplace=True)

train.drop("Ticket", axis=1, inplace=True)
test.drop("Ticket", axis=1, inplace=True)
```

In [13]:
```
train.groupby('Title')['Age'].mean()
```

Out[13]:
```
Title
Master     4.574167
```

```
Miss      21.816327
Mr        32.368090
Mrs       35.788991
Other     43.750000
Name: Age, dtype: float64
```

In [14]:
```
data = [train, test]
for df in data:
    df.loc[(df["Age"].isnull()) & (df["Title"]=='Master'), 'Age'] = 5
    df.loc[(df["Age"].isnull()) & (df["Title"]=='Miss'), 'Age'] = 22
    df.loc[(df["Age"].isnull()) & (df["Title"]=='Mr'), 'Age'] = 32
    df.loc[(df["Age"].isnull()) & (df["Title"]=='Mrs'), 'Age'] = 36
    df.loc[(df["Age"].isnull()) & (df["Title"]=='Other'), 'Age'] = 44
```

In [15]:
```
train = pd.get_dummies(train, prefix=["Sex", "Embarked", "Title"])
test = pd.get_dummies(test, prefix=["Sex", "Embarked", "Title"])
```

In [16]:
```
test.Fare.fillna(train.groupby("Pclass").mean()["Fare"][3], inplace=True)
```

Here I'm just putting the data through the standard scaler and then using k-fold cross valadation to test each of the models. The all produce fairly similar results so I just decided to use the KNN and SVC to see if any parameter hypertuning would help

In [17]:
```
X_train = train.drop("Survived", axis=1)
y_train = train.Survived
```

In [18]:
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(test)
```

In [19]:
```
kf =KFold(n_splits=5, shuffle=True, random_state=42)

cnt = 1
# split()  method generate indices to split data into training and test set.
for train_index, test_index in kf.split(X_train, y_train):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:{len(test_index)}')
    cnt += 1
```

```
Fold:1, Train set: 712, Test set:179
Fold:2, Train set: 713, Test set:178
Fold:3, Train set: 713, Test set:178
Fold:4, Train set: 713, Test set:178
Fold:5, Train set: 713, Test set:178
```

In [20]:
```
score = cross_val_score(LogisticRegression(random_state= 42), X_train, y_train, cv= kf,
print(f'Scores for each fold are: {score}')
print(f'Average score: {"{:.2f}".format(score.mean())}')
```

```
Scores for each fold are: [0.81564246 0.81460674 0.87078652 0.81460674 0.82022472]
Average score: 0.83
```

In [21]:
```
score = cross_val_score(KNeighborsClassifier(n_neighbors= 10), X_train, y_train, cv= kf
print(f'Scores for each fold are: {score}')
print(f'Average score: {"{:.2f}".format(score.mean())}')
```

```
Scores for each fold are: [0.81005587 0.8258427  0.86516854 0.79775281 0.84831461]
Average score: 0.83
```

In [22]:
```
score = cross_val_score(SVC(gamma='auto'), X_train, y_train, cv= kf, scoring="accuracy"
print(f'Scores for each fold are: {score}')
print(f'Average score: {"{:.2f}".format(score.mean())}')
```

```
Scores for each fold are: [0.81005587 0.8258427  0.88764045 0.78651685 0.84269663]
Average score: 0.83
```

In [23]:
```
SVC_model = SVC()
KNN_model = KNeighborsClassifier()
log_model = LogisticRegression()
RFR = RandomForestClassifier()
GBT = GradientBoostingClassifier()
```

In [24]:
```
X= train.drop('Survived', axis=1)
y= train['Survived']
```

In [25]:
```
from sklearn.model_selection import train_test_split
for i in range(4):
    X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_stat
```

In [26]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True)
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

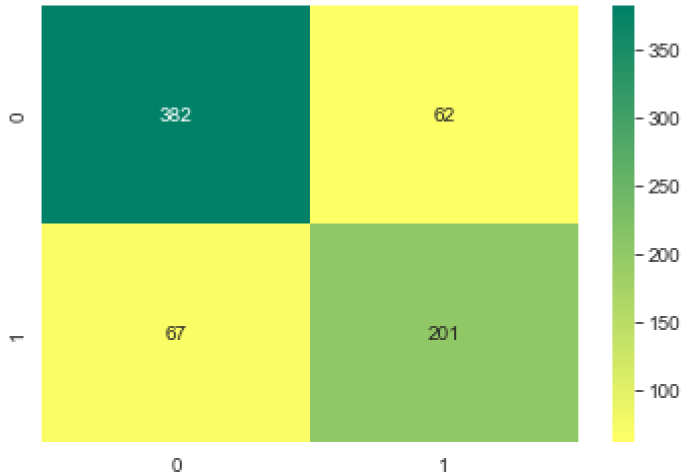
Here I decided to do some hyperparameter tuning and as we can see the KNN model is super accurate with a 98% accuracy it almost correctly guessed all of those that didn't survive. This model has great precision, accuracy and recall and if I had to put a model into production it would be that one

In [27]:
```
SVC_model = SVC()
params = {
    'C': [0.1,1, 10, 100],
    'kernel': ['rbf'],
    'gamma' : ['scale','auto']
}
clf = GridSearchCV(SVC_model, params, cv=10)
clf.fit(X_train, y_train)
print("Best hyperparameter:", clf.best_params_)
```

```
Best hyperparameter: {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
```

In [28]:
```
y_pred = clf.predict(X_train)
print(f"Train Accuracy: {accuracy_score(y_train, y_pred)}")
print(f"Train F1-Score: {f1_score(y_train, y_pred)}")
sns.heatmap(confusion_matrix(y_train, y_pred), fmt='.3g', annot=True, cmap='summer_r')
plt.show()
```

```
Train Accuracy: 0.8188202247191011
Train F1-Score: 0.7570621468926554
```



In [29]:
```
print(classification_report(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.86      0.86       444
           1       0.76      0.75      0.76       268

    accuracy                           0.82       712
   macro avg       0.81      0.81      0.81       712
weighted avg       0.82      0.82      0.82       712
```

In [30]:
```
print_score(clf, X_train, y_train, X_test, y_test, train=True)
print_score(clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 81.88%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.850780    0.764259   0.81882    0.807519      0.818213
recall       0.860360    0.750000   0.81882    0.805180      0.818820
f1-score     0.855543    0.757062   0.81882    0.806303      0.818474
support    444.000000  268.000000   0.81882  712.000000    712.000000
_______________________________________________
Confusion Matrix:
 [[382  62]
 [ 67 201]]

Test Result:
================================================
Accuracy Score: 80.45%
_______________________________________________
```

```
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.836538   0.760000  0.804469    0.798269      0.804897
recall       0.828571   0.770270  0.804469    0.799421      0.804469
f1-score     0.832536   0.765101  0.804469    0.798818      0.804658
support    105.000000  74.000000  0.804469  179.000000    179.000000
_______________________________________________
Confusion Matrix:
 [[87 18]
 [17 57]]
```

In [31]:

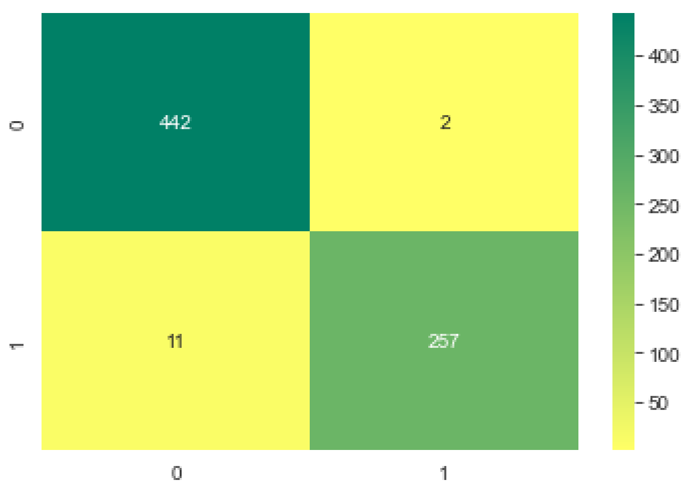
```python
params = {
    'leaf_size' : [1, 5, 10, 15, 20 ,25],
    'n_neighbors' : [1, 5, 10, 15, 20 ,25],
    'p' : [1,2],
    'algorithm' : ['ball_tree', 'kd_tree', 'brute'],
    'weights' : ['uniform', 'distance']
}
clf = GridSearchCV(KNN_model, params, cv=10)
clf.fit(X_train, y_train)
print("Best hyperparameter:", clf.best_params_)
```

```
Best hyperparameter: {'algorithm': 'ball_tree', 'leaf_size': 10, 'n_neighbors': 25, 'p':
1, 'weights': 'distance'}
```

In [32]:

```python
y_pred = clf.predict(X_train)
print(f"Train Accuracy: {accuracy_score(y_train, y_pred)}")
print(f"Train F1-Score: {f1_score(y_train, y_pred)}")
sns.heatmap(confusion_matrix(y_train, y_pred), fmt='.3g', annot=True, cmap='summer_r')
plt.show()
```

```
Train Accuracy: 0.9817415730337079
Train F1-Score: 0.9753320683111955
```



In [33]:

```python
print(classification_report(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       444
           1       0.99      0.96      0.98       268

    accuracy                           0.98       712
   macro avg       0.98      0.98      0.98       712
```

```
weighted avg       0.98      0.98      0.98       712
```

In [34]:
```
print_score(clf, X_train, y_train, X_test, y_test, train=True)
print_score(clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 98.17%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.975717    0.992278  0.981742    0.983998      0.981951
recall       0.995495    0.958955  0.981742    0.977225      0.981742
f1-score     0.985507    0.975332  0.981742    0.980420      0.981677
support    444.000000  268.000000  0.981742  712.000000    712.000000
_______________________________________________
Confusion Matrix:
 [[442   2]
 [ 11 257]]

Test Result:
================================================
Accuracy Score: 80.45%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.807018   0.800000  0.804469    0.803509      0.804116
recall       0.876190   0.702703  0.804469    0.789447      0.804469
f1-score     0.840183   0.748201  0.804469    0.794192      0.802157
support    105.000000  74.000000  0.804469  179.000000    179.000000
_______________________________________________
Confusion Matrix:
 [[92 13]
 [22 52]]
```

Here I created the random forests and decision trees and selected the best hyper parameters, we get some interesting results seeing that the gradient boosted tree is actually the best model and that gradient boosted tree is actually the best model. It has the second best training accuracy, which translated well into the test with about 84% accuracy. Which is interesting since the KNN model had nearly perfect accuracy on training on the training but not the test which probably means that the model is overfitted and perhaps a kfold cross validation or something similar would help solve that.

In [35]:
```
params = {
    "n_estimators" : [100, 150, 200],
    "max_features" : ["auto","sqrt"],
    "max_depth": [3,5,7,9]
}
clf = GridSearchCV(RFR, params, cv=10)
clf.fit(X_train, y_train)
print("Best hyperparameter:", clf.best_params_)
```

```
Best hyperparameter: {'max_depth': 5, 'max_features': 'auto', 'n_estimators': 150}
```

In [36]:
```
print_score(clf, X_train, y_train, X_test, y_test, train=True)
print_score(clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
```

```
================================================
Accuracy Score: 86.10%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.852761    0.878924  0.860955    0.865842      0.862609
recall       0.939189    0.731343  0.860955    0.835266      0.860955
f1-score     0.893891    0.798371  0.860955    0.846131      0.857937
support    444.000000  268.000000  0.860955  712.000000    712.000000
_______________________________________________
Confusion Matrix:
 [[417  27]
 [ 72 196]]

Test Result:
================================================
Accuracy Score: 82.12%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.828829   0.808824  0.821229    0.818826      0.820558
recall       0.876190   0.743243  0.821229    0.809717      0.821229
f1-score     0.851852   0.774648  0.821229    0.813250      0.819935
support    105.000000  74.000000  0.821229  179.000000    179.000000
_______________________________________________
Confusion Matrix:
 [[92 13]
 [19 55]]
```

In [37]:

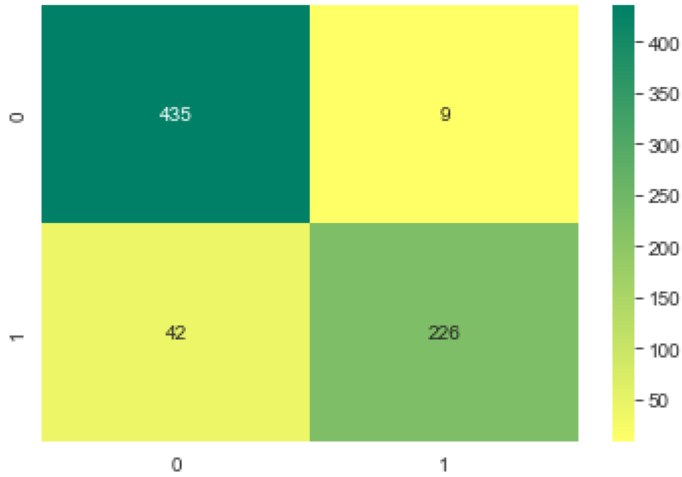
```
params = {
    "n_estimators" : [100, 150, 200, 250],
    "max_features" : ["auto","sqrt"],
    "max_depth": [3,5,7]
}
clf = GridSearchCV(GBT, params, cv=10)
clf.fit(X_train, y_train)
print("Best hyperparameter:", clf.best_params_)
```

```
Best hyperparameter: {'max_depth': 3, 'max_features': 'auto', 'n_estimators': 200}
```

In [38]:

```
y_pred = clf.predict(X_train)
print(f"Train Accuracy: {accuracy_score(y_train, y_pred)}")
print(f"Train F1-Score: {f1_score(y_train, y_pred)}")
sns.heatmap(confusion_matrix(y_train, y_pred), fmt='.3g', annot=True, cmap='summer_r')
plt.show()
```

```
Train Accuracy: 0.9283707865168539
Train F1-Score: 0.8986083499005965
```



In [39]:
```
print(classification_report(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94       444
           1       0.96      0.84      0.90       268

    accuracy                           0.93       712
   macro avg       0.94      0.91      0.92       712
weighted avg       0.93      0.93      0.93       712
```

In [40]:
```
print_score(clf, X_train, y_train, X_test, y_test, train=True)
print_score(clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 92.84%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.911950    0.961702  0.928371    0.936826      0.930677
recall       0.979730    0.843284  0.928371    0.911507      0.928371
f1-score     0.944625    0.898608  0.928371    0.921617      0.927304
support    444.000000  268.000000  0.928371  712.000000    712.000000
_______________________________________________
Confusion Matrix:
 [[435   9]
 [ 42 226]]

Test Result:
================================================
Accuracy Score: 84.36%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.859813   0.819444  0.843575    0.839629      0.843124
recall       0.876190   0.797297  0.843575    0.836744      0.843575
f1-score     0.867925   0.808219  0.843575    0.838072      0.843242
support    105.000000  74.000000  0.843575  179.000000    179.000000
_______________________________________________
Confusion Matrix:
 [[92 13]
```

```
 [15 59]]
```