In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

In [4]:
```
data = pd.read_csv('Social_Network_Ads (2)-Copy1.csv')
data.head()
```

Out[4]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [5]:
```
data.drop(columns=('Gender'),inplace=True)
```

In [6]:
```
data.tail()
```

Out[6]:

	User ID	Age	EstimatedSalary	Purchased
395	15691863	46	41000	1
396	15706071	51	23000	1
397	15654296	50	20000	1
398	15755018	36	33000	0
399	15594041	49	36000	1

In [7]:
```
data.drop(columns=('User ID'),inplace=True)
```

In [8]:
```
data.head()
```

Out[8]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

In [9]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [10]:
```
x = data.drop(columns=('Purchased'))
y = data['Purchased']
```

In [11]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0,train_size=0.75)
```

In [12]:
```
x_train = x_train.astype('float')
x_test  = x_test.astype('float')
```

In [13]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [14]:
```
from sklearn.model_selection import GridSearchCV
```

In [15]:
```
parameter_grid ={'n_neighbors':np.arange(1,50)}
```

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [17]:
```
kNN = KNeighborsClassifier()
```

GridSearchCv

GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

In addition to that, you can specify the number of times for the cross-validation for each set of hyperparameters.

Then all you have to do is create an object of GridSearchCV. Here basically you need to define a few named arguments:

1. **estimator:** estimator object you created
2. **params_grid:** the dictionary object that holds the hyperparameters you want to try
3. **scoring:** evaluation metric that you want to use, you can simply pass a valid string/ object of evaluation metric
4. **cv:** number of cross-validation you have to try for each selected set of hyperparameters
5. **verbose:** you can set it to 1 to get the detailed print out while you fit the data to GridSearchCV
6. **n_jobs:** number of processes you wish to run in parallel for this task if it -1 it will use all available processors.

In [18]:
```
knn_cv=GridSearchCV(kNN,parameter_grid,cv=5)
knn_cv.fit(x_train,y_train)
```

Out[18]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

In [19]:
```
y_pred=knn_cv.predict(x_test)
y_pred
```

Out[19]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], dtype=int64)
```

In [20]:
```
knn_cv.best_params_
```

Out[20]:
```
{'n_neighbors': 5}
```

In [21]:
```
knn_cv.best_score_
```

Out[21]:
```
0.9033333333333333
```

In [22]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [23]:
```
mat = confusion_matrix(y_test,y_pred)
mat
```

Out[23]:
```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

In [24]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        68
           1       0.88      0.91      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100
```

RandomSearchCv

RandomSearchCV has the same purpose of GridSearchCV: they both were designed to find the best parameters to improve your model. However, here not all parameters are tested. Rather, the search is randomized and all the other parameters are held constant while the parameters we are testing is variable. Pratically, the implementation of RandomSearchCV is very similar to that of the GridSearchCV:

In [25]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint
```

In [26]:
```
#setup the parameter and distribution to sample from :- params_dict
```

In [27]:
```
param_dist = {'max_depth':[3,None],'min_samples_leaf':randint(1,9),'criterion':['gini','entropy']}
```

In [28]:
```
#decision Tree
```

In [29]:
```
tree = DecisionTreeClassifier()
```

In [30]:
```
#RandomizedSearchCV
```

In [31]:
```
tree_cv = RandomizedSearchCV(tree,param_dist,cv=5)
```

In [32]:
```
tree_cv.fit(x_train,y_train)
```

Out[32]:
```
RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [3, None],
                                        'min_samples_leaf': <scipy.stats._distn_infrastructure.rv_frozen object
at 0x0000023D64FC6310>})
```

In [33]:
```
tree_cv.predict(x_test)
```

Out[33]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], dtype=int64)
```

In [34]:
```
tree_cv.best_params_
```

Out[34]:
```
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 7}
```

In [35]:
```
tree_cv.best_score_
```

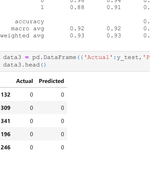
Out[35]:
```
0.9
```

In [36]:
```
mat = confusion_matrix(y_test,y_pred)
mat
```

Out[36]:
```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

In [37]:
```
import seaborn as sns
```

In [38]:
```
sns.heatmap(mat,annot=True,fmt='d')
plt.show()
```



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

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        68
           1       0.88      0.91      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100
```

In [40]:
```
data3 = pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
data3.head()
```

Out[40]:

	Actual	Predicted
132	0	0
309	0	0
341	0	0
196	0	0
246	0	0

RandomSearchCV VS Greadsearchcv

- With GridSearchCV, by calling the method *bestparams* you are guaranteed to get the best model results (according to your scoring) within your test values, since it will test every single one of the values you passed.
- However, with RandomSearchCV, the more samples you test from the value set, the more confident the search will be — but it will never be 100% certain (unless you test every value out of the set of possible values). Statistically speaking, we can be fairly confident that the best parameters found are indeed the best combination of optimal parameters since the search is completely randomized.
- The running times of RandomSearchCV vs. GridSearchCV on the other hand, are widely different. Depending on the n_iter chosen, RandomSearchCV can be two, three, four times faster than GridSearchCV. However, the higher the n_iter chosen, the lower will be the speed of RandomSearchCV and the closer the algorithm will be to GridSearchCV.

GreadsearchCv

In [ ]:
```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max_features = ['auto', 'sqrt']

min_samples_split = [2, 5, 10]

min_samples_leaf = [1, 2, 4]
rf_grid = {'n_estimators': n_estimators,
           'max_features': max_features,
           'max_depth': max_depth,
           'min_samples_split': min_samples_split,
           'min_samples_leaf': min_samples_leaf}
model = GridSearchCV(RandomForestClassifier(), rf_grid, scoring = 'accuracy', cv = 5)
# fit the model
model.fit(x_train, y_train)
```

RandomsearchCv

In [ ]:
```
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max_features = ['auto', 'sqrt']

min_samples_split = [2, 5, 10]

min_samples_leaf = [1, 2, 4]
rf_grid = {'n_estimators': n_estimators,
           'max_features': max_features,
           'max_depth': max_depth,
           'min_samples_split': min_samples_split,
           'min_samples_leaf': min_samples_leaf}
model = RandomizedSearchCV(estimator = RandomForestClassifier(),
                   param_distributions = rf_grid,
                   cv = 5, n_iter = 100)
# fit the model
model.fit(x_train, y_train)
```

In [ ]: