In [47]:

```
import pandas as pd
import numpy as np



# import the ML algorithm
from sklearn.ensemble import RandomForestClassifier


from sklearn.model_selection import train_test_split
```

In [7]:

```
 # Load the training data from titanic data set


df_training = pd.read_csv(r"C:\Users\keert\Desktop\AI ML TILL  CERT\AI and ML\datas
```

In [8]:

```
df_training.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Na
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C8
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Na
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C12
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	Na

In [10]:

```
df_training.describe()
```

Out[10]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [11]:

```
# We will remove ‘Cabin’, ‘Name’ and ‘Ticket’ columns
df_training_dropped = df_training.drop(['Cabin', 'Name', 'Ticket'], axis=1)
df_training_dropped.head()
```

Out[11]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

In [12]:

```
# Examine any missing
df_training_dropped.isnull().sum()
```

Out[12]:

```
PassengerId      0
Survived         0
Pclass           0
Sex              0
Age            177
SibSp            0
Parch            0
Fare             0
Embarked         2
dtype: int64
```

In [14]:

```
# Filling missing Age values with mean
df_training_dropped['Age'] = df_training_dropped['Age'].fillna(df_training['Age'].
```

In [18]:

```
df_training['Embarked'].mode()
```

Out[18]:

```
0    S
Name: Embarked, dtype: object
```

In [21]:

```
# Filling missing Embarked values with mode
df_training_dropped['Embarked'] = df_training_dropped['Embarked'].fillna(df_traini
```

In [22]:

```
# Examine any missing
df_training_dropped.isnull().sum()
```

```
Out[22]: PassengerId    0
         Survived       0
         Pclass         0
         Sex            0
         Age            0
         SibSp          0
         Parch          0
         Fare           0
         Embarked       0
         dtype: int64
```

```
In [23]: df_training_dropped.head()
```

Out[23]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

```
In [37]: df_training_dropped.Embarked.value_counts()
```

```
Out[37]: S    646
         C    168
         Q     77
         Name: Embarked, dtype: int64
```

```
In [24]: # one hot encoding of categorical features
         df_training_dropped.dtypes
```

```
Out[24]: PassengerId      int64
         Survived         int64
         Pclass           int64
         Sex             object
         Age            float64
         SibSp            int64
         Parch            int64
         Fare           float64
         Embarked        object
         dtype: object
```

```
In [25]: df_training_dummied = pd.get_dummies(df_training_dropped, columns=["Pclass", 'Sex'
```

```
In [26]: df_training_dummied.head(3)
```

Out[26]:

	PassengerId	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female
0	1	0	22.0	1	0	7.2500	0	0	1	0
1	2	1	38.0	1	0	71.2833	1	0	0	1
2	3	1	26.0	0	0	7.9250	0	0	1	1

```
In [38]: # X_df = df_training_dummied.iloc[:,:-1]
         X_df = df_training_dummied.drop('Survived', axis=1)
```

```
In [39]: y_df = df_training_dummied['Survived']
```

In [40]:

```
X_df
```

Out[40]:

	PassengerId	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male
0	1	22.0	1	0	7.2500	0	0	1	0	1
1	2	38.0	1	0	71.2833	1	0	0	1	0
2	3	26.0	0	0	7.9250	0	0	1	1	0
3	4	35.0	1	0	53.1000	1	0	0	1	0
4	5	35.0	0	0	8.0500	0	0	1	0	1
...	...	...	...	...	...	...	...	...	...	...
886	887	27.0	0	0	13.0000	0	1	0	0	1
887	888	19.0	0	0	30.0000	1	0	0	1	0
888	889	28.0	1	2	23.4500	0	0	1	1	0
889	890	26.0	0	0	30.0000	1	0	0	0	1
890	891	32.0	0	0	7.7500	0	0	1	0	1

891 rows × 13 columns

In [44]:

```
# Split into train and test sets.

X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size = 0.25,
```

In [48]:

```
model = RandomForestClassifier()

model.get_params()
```

Out[48]:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [51]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[51]:

```
((668, 13), (223, 13), (668,), (223,))
```

In [57]:

```
#Checking accuracy with 1- 150 estimators

n_estimators = np.arange(1, 150, 1)
n_estimators
```

Out[57]:

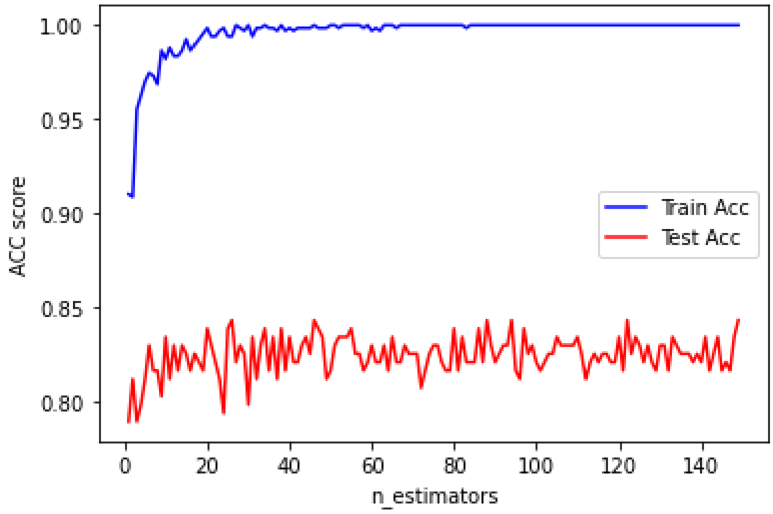
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
        92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104,
       105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
       118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
       131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
       144, 145, 146, 147, 148, 149])
```

In [58]:

```python
from sklearn import metrics

train_results = []
test_results  = []

for estimator in n_estimators:

    rf = RandomForestClassifier(n_estimators=estimator, n_jobs=-1)
    rf.fit(X_train, y_train)

    # we need to predict the training samples
    # calculate the accuracy of the training samples
    train_pred = rf.predict(X_train)
    train_acc  = metrics.accuracy_score(y_train, train_pred)

    train_results.append(train_acc)

    # we need to predict the test samples
    # calculate the accuracy of the test samples
    test_pred  = rf.predict(X_test)
    test_acc   = metrics.accuracy_score(y_test, test_pred)

    test_results.append(test_acc)
```

In [59]:

```python
import matplotlib.pyplot as plt
plt.plot(n_estimators, train_results, 'b', label= 'Train Acc')
plt.plot(n_estimators, test_results,  'r', label= 'Test Acc')

plt.ylabel('ACC score')
plt.xlabel('n_estimators')
plt.legend();
```



In [64]:

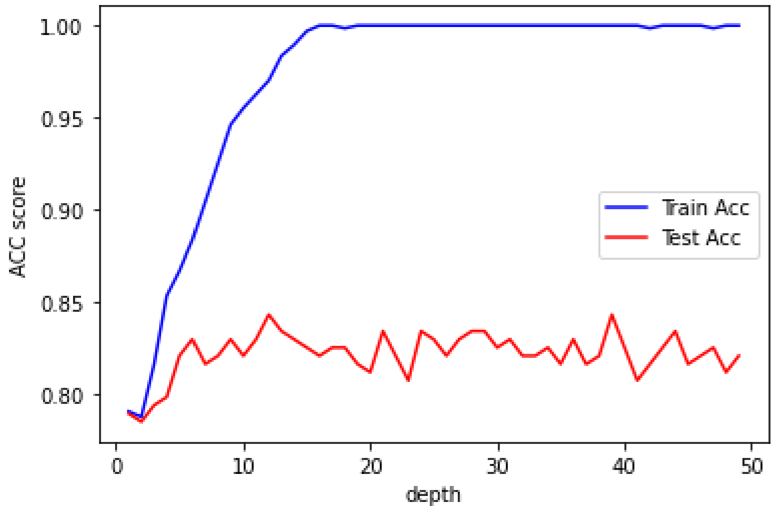
```python
max_depths = np.arange(1, 50, 1)
```

```
max_depths
```

```
Out[64]: 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 [65]: #Checking accuracy with 1- 50 max depth


train_results = []
test_results  = []

for depth in max_depths:

    # Instantiate the RF model with the depth setting
    rf = RandomForestClassifier(max_depth=depth, n_jobs=-1)

    rf.fit(X_train, y_train)

    # we need to predict the training samples
    # calculate the accuracy of the training samples
    train_pred = rf.predict(X_train)
    train_acc  = metrics.accuracy_score(y_train, train_pred)

    train_results.append(train_acc)

    # we need to predict the test samples
    # calculate the accuracy of the test samples
    test_pred  = rf.predict(X_test)
    test_acc   = metrics.accuracy_score(y_test, test_pred)

    test_results.append(test_acc)
```

```
In [67]: plt.plot(max_depths, train_results, 'b', label= 'Train Acc')
plt.plot(max_depths, test_results,  'r', label= 'Test Acc')

plt.ylabel('ACC score')
plt.xlabel('depth')
plt.legend();
```



Observation : the depth of 3 to 8 would be optimal, as this gap between the training and test acc is acceptable

```
In [ ]:
```

In [ ]:

GridSearch

In [69]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.model_selection import KFold
```

In [70]:

```
kf = KFold(n_splits=10, shuffle= True, random_state=20)
```

In [71]:

```
%%time

from sklearn.metrics import make_scorer

scoring = {'accuracy': 'accuracy', 'roc_auc': 'roc_auc'}
#scoring = {'AUC': 'roc_auc', 'Accuracy': make_scorer(accuracy_score)}

# params = {'n_estimators': [5, 50, 100, 200, 400, 600],
#           'max_depth':    [2, 3, 4, 5],
#           'max_features': ['sqrt', 'log2', None],
#           'bootstrap': ['True'],
#           'max_samples': [0.1, 0.3, 0.9, 1.0]}

params = {'n_estimators': [3, 5, 10, 15, 20],
          'max_depth':    [2, 3, 4, 5, 6, 7, 8, 9],
          'max_features': ['sqrt', 'log2'],
          'min_samples_split': np.linspace(0.05, .4, 5)
          }

# gs     = RandomizedSearchCV(estimator = RandomForestClassifier(),
#                             param_distributions= params,
#                             scoring=scoring,
#                             n_jobs=4,
#                             cv=kf,
#                             return_train_score=True,
#                             refit= 'roc_auc'
#                             )
# gs.fit(X_train, y_train)
# gs.best_score_, gs.best_params_

gs     = GridSearchCV(estimator = RandomForestClassifier(),
                                param_grid= params,
                                scoring='accuracy',
                                n_jobs=4,
                                cv=kf,

                            )
gs.fit(X_train, y_train)
gs.best_params_
```

```
CPU times: total: 1.78 s
Wall time: 20.8 s
```

Out[71]:

```
{'max_depth': 9,
 'max_features': 'log2',
 'min_samples_split': 0.05,
 'n_estimators': 10}
```

In [73]:

```
# Using best params

rf = RandomForestClassifier(n_estimators=10,max_depth= 9,max_features='log2', min_
```

In [74]:

```
rf.fit(X_train, y_train)

    # we need to predict the training samples
    # calculate the accuracy of the training samples
train_pred = rf.predict(X_train)
train_acc  = metrics.accuracy_score(y_train, train_pred)

train_results.append(train_acc)

    # we need to predict the test samples
    # calculate the accuracy of the test samples
test_pred  = rf.predict(X_test)
test_acc   = metrics.accuracy_score(y_test, test_pred)

test_results.append(test_acc)
```

In [78]:

```
test_acc,train_acc
```

Out[78]:

```
(0.8116591928251121, 0.8547904191616766)
```

In [ ]:

```

```

In [ ]:

```

```