Kyle Anthony F. Niosco
III – BSCS STDS

July 28, 2022
Prof. Rodrigo C. Belleza, Jr.

Final Exam
(Applied Data Science with Python)

1. Decision Tree

```
In [1]: # Load libraries

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

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, f1_score, classification_report, accuracy_score
from sklearn.tree import plot_tree
from sklearn import tree

%matplotlib inline
```

```
In [2]: #Open/Read Dataset

data = pd.read_csv('cardio_train.csv')

# Printing the dataset shape
print ("Dataset Length: ", len(data))
print ("\nDataset Shape: ", data.shape)

Dataset Length:  70000

Dataset Shape:  (70000, 1)
```

```
In [3]: #Display first 5 values

data.head()
```

Out[3]:

	id;age;gender;height;weight;ap_hi;ap_lo;cholesterol;gluc;smoke;alco;active;cardio
0	0;18393;2;168;62.0;110;80;1;1;0;0;1;0
1	1;20228;1;156;85.0;140;90;3;1;0;0;1;1
2	2;18857;1;165;64.0;130;70;3;1;0;0;0;1
3	3;17623;2;169;82.0;150;100;1;1;0;0;1;1
4	4;17474;1;156;56.0;100;60;1;1;0;0;0;0

```
In [4]: #Data Cleaning/Wrangling

newdata = data['id;age;gender;height;weight;ap_hi;ap_lo;cholesterol;gluc;smoke;alco;active;cardio'].str.split(';', expand=True)

newdata
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	99993	19240	2	168	76.0	120	80	1	1	1	0	1	0
69996	99995	22601	1	158	126.0	140	90	2	2	0	0	1	1
69997	99996	19066	2	183	105.0	180	90	3	1	0	1	0	1
69998	99998	22431	1	163	72.0	135	80	1	2	0	0	0	1
69999	99999	20540	1	170	72.0	120	80	2	1	0	0	1	0

70000 rows × 13 columns

In [5]:
```python
# Assigning the column names

col_names = ['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc',\
             'smoke', 'alco', 'active', 'cardio']

newdata.columns = col_names

newdata.head()
```

Out[5]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

In [6]:
```python
newdata.dtypes
```

Out[6]:
```
id             object
age            object
gender         object
height         object
weight         object
ap_hi          object
ap_lo          object
cholesterol    object
gluc           object
smoke          object
alco           object
active         object
cardio         object
dtype: object
```

In [7]:
```python
# Converting the variable 'age' from object to into integer

newdata['age'] = newdata['age'].astype(int)

newdata.head()
```

Out[7]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

In [8]:
```python
newdata.dtypes
```

Out[8]:
```
id             object
age             int32
gender         object
height         object
weight         object
ap_hi          object
ap_lo          object
cholesterol    object
gluc           object
smoke          object
alco           object
active         object
cardio         object
dtype: object
```

In [9]:
```python
# Dividing the values in 'age' column by 365 to convert the age from days to years

newdata['age'] = (newdata['age'].div(365).round(2))

newdata['age'] = newdata['age'].astype(int)

newdata.head()
```

Out[9]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	50	2	168	62.0	110	80	1	1	0	0	1	0
1	1	55	1	156	85.0	140	90	3	1	0	0	1	1
2	2	51	1	165	64.0	130	70	3	1	0	0	0	1
3	3	48	2	169	82.0	150	100	1	1	0	0	1	1
4	4	47	1	156	56.0	100	60	1	1	0	0	0	0

In [10]:
```
# Printing the dataset shape
print ("Dataset Length: ", len(newdata))
print ("\nDataset Shape: ", newdata.shape)
```
```
Dataset Length:  70000

Dataset Shape:  (70000, 13)
```

In [11]:
```
#Summary of Data

newdata.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   id           70000 non-null  object
 1   age          70000 non-null  int32 
 2   gender       70000 non-null  object
 3   height       70000 non-null  object
 4   weight       70000 non-null  object
 5   ap_hi        70000 non-null  object
 6   ap_lo        70000 non-null  object
 7   cholesterol  70000 non-null  object
 8   gluc         70000 non-null  object
 9   smoke        70000 non-null  object
 10  alco         70000 non-null  object
 11  active       70000 non-null  object
 12  cardio       70000 non-null  object
dtypes: int32(1), object(12)
memory usage: 6.7+ MB
```

In [12]:
```
# Checking the counts of each class label

newdata['cardio'].value_counts()
```
Out[12]:
```
0    35021
1    34979
Name: cardio, dtype: int64
```

In [13]:
```
#Check for missing values

newdata.isnull().sum()
```
Out[13]:
```
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

In [14]:
```
#Splitting feature and target variables

feature = newdata.drop(['cardio'], axis = 1)

target = newdata['cardio']
```

In [15]:
```
#Display first 5 values from the feature variables

feature.head()
```
Out[15]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	0	50	2	168	62.0	110	80	1	1	0	0	1
1	1	55	1	156	85.0	140	90	3	1	0	0	1
2	2	51	1	165	64.0	130	70	3	1	0	0	0
3	3	48	2	169	82.0	150	100	1	1	0	0	1
4	4	47	1	156	56.0	100	60	1	1	0	0	0

In [16]:
```python
#Display first 5 values from the target variable

target.head()
```

Out[16]:
```
0    0
1    1
2    1
3    1
4    0
Name: cardio, dtype: object
```

In [17]:
```python
#Splitting the data into training and testing sets

feature_train, feature_test, target_train, target_test = train_test_split(feature, target, test_size = 0.30, random_state = 42)
```

In [18]:
```python
#Test Size and Training set Info

print("SUMMARY OF SPLITTED TRAINING AND TEST SETS\n")

print('--Number of Training Set:',len(feature_train))

print('--Training Size:', len(feature_train)/len(data) * 100, '%')

print('\n--Number of Test Set:',len(feature_test))

print('--Test Size:', len(feature_test)/len(data) * 100, '%')
```

```
SUMMARY OF SPLITTED TRAINING AND TEST SETS

--Number of Training Set: 49000
--Training Size: 70.0 %

--Number of Test Set: 21000
--Test Size: 30.0 %
```

In [19]:
```python
# Checking the shape of feature variables' train and test sets

feature_train.shape, feature_test.shape
```

Out[19]:
```
((49000, 12), (21000, 12))
```

In [20]:
```python
#Feature Engineering

feature_train.dtypes
```

Out[20]:
```
id             object
age             int32
gender         object
height         object
weight         object
ap_hi          object
ap_lo          object
cholesterol    object
gluc           object
smoke          object
alco           object
active         object
dtype: object
```

In [21]:
```python
feature_train.head()
```

Out[21]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
68681	98125	55	1	160	64.0	120	90	3	1	0	0	1
19961	28510	62	2	167	65.0	120	80	3	3	0	0	0
11040	15795	62	1	160	66.0	120	90	1	1	0	0	1
27673	39560	62	1	163	55.0	125	90	3	1	0	0	1
22876	32677	59	1	158	85.0	150	80	3	1	0	0	1

In [22]:
```
feature_test.head()
```
Out[22]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
46730	66728	59	1	156	64.0	140	80	2	1	0	0	1
48393	69098	59	1	170	85.0	160	90	1	1	0	0	1
41416	59185	63	1	151	90.0	130	80	1	1	0	0	1
34506	49288	54	1	159	97.0	120	80	1	1	0	0	1
43725	62481	50	1	164	68.0	120	80	1	1	0	0	1

In [23]:
```
#Creating Decision Tree Classifier Object using Entropy
clf_ent = DecisionTreeClassifier(criterion='entropy', max_depth = 3, random_state = 42)
#Fitting the Model
clf_ent.fit(feature_train, target_train)
```
Out[23]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
```

In [24]:
```
#Predicting the test set
target_pred_ent = clf_ent.predict(feature_test)
target_pred_ent
```
Out[24]:
```
array(['1', '1', '1', ..., '1', '1', '1'], dtype=object)
```

In [25]:
```
#Predicting using train set
target_pred_train_ent = clf_ent.predict(feature_train)
target_pred_train_ent
```
Out[25]:
```
array(['1', '1', '0', ..., '0', '0', '0'], dtype=object)
```

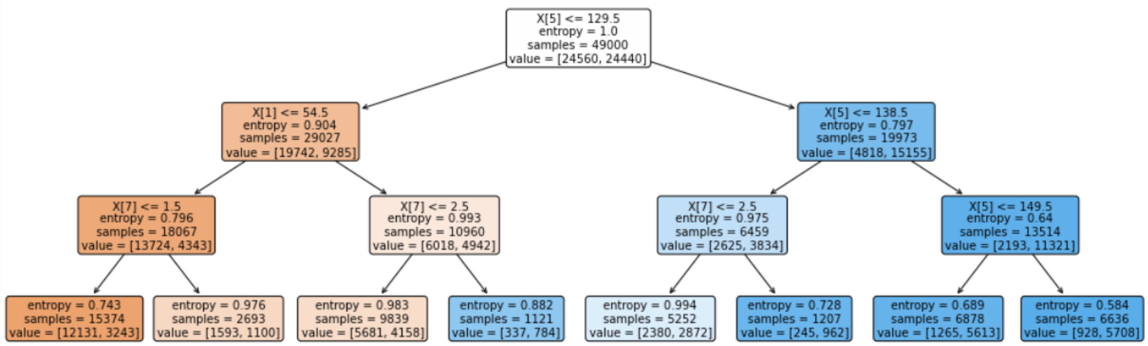
In [26]:
```
#Checking accuracy of Training Set and Testing Set to check Overfitting
print('Model Accuracy for Testing Set: {0:0.4f}'. format(accuracy_score(target_test, target_pred_ent)))
print('\nModel Accuracy for Training Set: {0:0.4f}'. format(accuracy_score(target_train, target_pred_train_ent)))
```
```
Model Accuracy for Testing Set: 0.7283

Model Accuracy for Training Set: 0.7213
```

In [27]:
```
#Comparing Null Accuracy and Model Accuracy
target_test.describe()
```
Out[27]:
```
count     21000
unique        2
top           1
freq      10539
Name: cardio, dtype: object
```

In [28]:
```
#Comparing Null Accuracy and Model Accuracy
null_acc = target_test.describe()[3] / target_test.describe()[0]
print('Model Accuracy: {0:0.4f}'. format(accuracy_score(target_test, target_pred_ent)))
print('\nNull Accuracy: {0:0.4f}'. format(null_acc))
```
```
Model Accuracy: 0.7283

Null Accuracy: 0.5019
```

In [29]:
```python
#Visualization

plt.figure(figsize = (18, 6))

tree.plot_tree(clf_ent.fit(feature_train, target_train), filled = True, impurity = True, rounded = True)
```

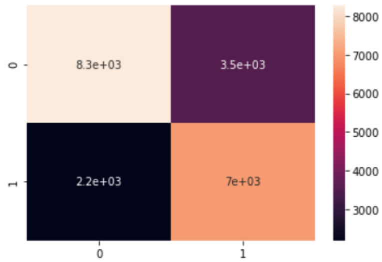
Out[29]:
```
[Text(502.20000000000005, 285.39000000000004, 'X[5] <= 129.5\nentropy = 1.0\nsamples = 49000\nvalue = [24560, 24440]'),
 Text(251.10000000000002, 203.85000000000002, 'X[1] <= 54.5\nentropy = 0.904\nsamples = 29027\nvalue = [19742, 9285]'),
 Text(125.55000000000001, 122.31, 'X[7] <= 1.5\nentropy = 0.796\nsamples = 18067\nvalue = [13724, 4343]'),
 Text(62.775000000000006, 40.76999999999998, 'entropy = 0.743\nsamples = 15374\nvalue = [12131, 3243]'),
 Text(188.32500000000002, 40.76999999999998, 'entropy = 0.976\nsamples = 2693\nvalue = [1593, 1100]'),
 Text(376.65000000000003, 122.31, 'X[7] <= 2.5\nentropy = 0.993\nsamples = 10960\nvalue = [6018, 4942]'),
 Text(313.875, 40.76999999999998, 'entropy = 0.983\nsamples = 9839\nvalue = [5681, 4158]'),
 Text(439.42500000000007, 40.76999999999998, 'entropy = 0.882\nsamples = 1121\nvalue = [337, 784]'),
 Text(753.3000000000001, 203.85000000000002, 'X[5] <= 138.5\nentropy = 0.797\nsamples = 19973\nvalue = [4818, 15155]'),
 Text(627.75, 122.31, 'X[7] <= 2.5\nentropy = 0.975\nsamples = 6459\nvalue = [2625, 3834]'),
 Text(564.975, 40.76999999999998, 'entropy = 0.994\nsamples = 5252\nvalue = [2380, 2872]'),
 Text(690.5250000000001, 40.76999999999998, 'entropy = 0.728\nsamples = 1207\nvalue = [245, 962]'),
 Text(878.8500000000001, 122.31, 'X[5] <= 149.5\nentropy = 0.64\nsamples = 13514\nvalue = [2193, 11321]'),
 Text(816.075, 40.76999999999998, 'entropy = 0.689\nsamples = 6878\nvalue = [1265, 5613]'),
 Text(941.6250000000001, 40.76999999999998, 'entropy = 0.584\nsamples = 6636\nvalue = [928, 5708]')]
```



In [30]:
```python
#Confusion Matrix

cm = confusion_matrix(target_pred_ent, target_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[8272 3516]
 [2189 7023]]

True Positives(TP) =  8272

True Negatives(TN) =  7023

False Positives(FP) =  3516

False Negatives(FN) =  2189
```

```
In [31]: #Visualization of Confusion Matrix

sns.heatmap(cm, annot = True)
plt.savefig('cm.png')
```




```
In [32]: #Evaluation Metrics Report

print(classification_report(target_test, target_pred_ent, zero_division = 0))
```

```
              precision    recall  f1-score   support

           0       0.70      0.79      0.74     10461
           1       0.76      0.67      0.71     10539

    accuracy                           0.73     21000
   macro avg       0.73      0.73      0.73     21000
weighted avg       0.73      0.73      0.73     21000
```

Summary of Insights

1. A Decision Tree algorithm is implemented in this project to predict the presence of a cardiovascular disease in a person. With an accuracy percentage of 72.83%, the model can be said to be performing well for being able to produce that kind of result.
2. There were no signs of overfitting since the values of model accuracy of the train and test sets are comparable, having only a small difference with each other. The model accuracy of the train and test sets yield 72.13% and 72.83%, respectively.
3. It can be concluded that the Decision tree model did a good job in predicting the class labels as it was able to produce a null accuracy of only 50.19%, which is lower than the model accuracy of 72.83%.

2. Naïve Bayes

```
In [1]: # Load Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import confusion_matrix, f1_score, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB

%matplotlib inline
```

```
In [2]: #Open/Read Dataset

data = pd.read_csv('gender_classification_v7.csv')

# Printing the dataset shape
print ("Dataset Length: ", len(data))
print ("\nDataset Shape: ", data.shape)
```

```
Dataset Length:  5001

Dataset Shape:  (5001, 8)
```

In [3]:
```
#Display first 5 values
data.head()
```

Out[3]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female

In [4]:
```
#Summary of Data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   long_hair                  5001 non-null   int64  
 1   forehead_width_cm          5001 non-null   float64
 2   forehead_height_cm         5001 non-null   float64
 3   nose_wide                  5001 non-null   int64  
 4   nose_long                  5001 non-null   int64  
 5   lips_thin                  5001 non-null   int64  
 6   distance_nose_to_lip_long  5001 non-null   int64  
 7   gender                     5001 non-null   object 
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

In [5]:
```
# Checking the counts of each class label
data['gender'].value_counts()
```

Out[5]:
```
Female    2501
Male      2500
Name: gender, dtype: int64
```

In [6]:
```
#Check for missing values
data.isnull().sum()
```

Out[6]:
```
long_hair                    0
forehead_width_cm            0
forehead_height_cm           0
nose_wide                    0
nose_long                    0
lips_thin                    0
distance_nose_to_lip_long    0
gender                       0
dtype: int64
```

In [7]:
```
#Splitting feature and target variables
feature = data.drop(['gender'], axis = 1)
target = data['gender']
```

In [8]:
```
#Display first 5 values from the feature variables
feature.head()
```

Out[8]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
0	1	11.8	6.1	1	0	1	1
1	0	14.0	5.4	0	0	1	0
2	0	11.8	6.3	1	1	1	1
3	0	14.4	6.1	0	1	1	1
4	1	13.5	5.9	0	0	0	0

```
In [9]: #Display first 5 values from the target variable

        target.head()
```

```
Out[9]: 0      Male
        1    Female
        2      Male
        3      Male
        4    Female
        Name: gender, dtype: object
```

```
In [10]: #Splitting the data into training and testing sets

         feature_train, feature_test, target_train, target_test = train_test_split(feature, target, test_size = 0.30, random_state = 0)
```

```
In [11]: #Test Size and Training set Info

         print("SUMMARY OF SPLITTED TRAINING AND TEST SETS\n")

         print('--Number of Training Set:',len(feature_train))

         print('--Training Size:', len(feature_train)/len(data) * 100, '%')

         print('\n--Number of Test Set:',len(feature_test))

         print('--Test Size:', len(feature_test)/len(data) * 100, '%')
```

```
SUMMARY OF SPLITTED TRAINING AND TEST SETS

--Number of Training Set: 3500
--Training Size: 69.98600279944012 %

--Number of Test Set: 1501
--Test Size: 30.01399720055989 %
```

```
In [12]: feature_train.shape, feature_test.shape
```

```
Out[12]: ((3500, 7), (1501, 7))
```

```
In [13]: #Feature Engineering

         feature_train.dtypes
```

```
Out[13]: long_hair                      int64
         forehead_width_cm            float64
         forehead_height_cm           float64
         nose_wide                      int64
         nose_long                      int64
         lips_thin                      int64
         distance_nose_to_lip_long      int64
         dtype: object
```

```
In [14]: feature_train.head()
```

Out[14]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
4160	1	12.3	5.8	0	1	1	0
1073	1	14.7	6.3	1	1	1	0
3583	1	13.1	6.1	1	0	1	1
1357	1	15.2	5.9	0	1	1	1
4645	0	15.4	5.7	1	1	1	1

```
In [15]: feature_test.head()
```

Out[15]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
2373	1	12.4	5.1	0	0	0	0
2755	1	12.4	5.7	1	1	1	1
2265	1	13.4	5.5	0	1	1	1
3901	1	12.0	6.1	0	0	0	0
3175	1	12.5	5.7	0	0	0	0

Using Bernoulli Naive Bayes Classifier

In [16]:
```
#Creating Naive Bayes Classifier Object using BernoulliNB()
bnb = BernoulliNB()

#Fitting the Model
bnb.fit(feature_train, target_train)
```

Out[16]:
```
BernoulliNB()
```

In [17]:
```
#Predicting using test set

target_pred = bnb.predict(feature_test)

target_pred
```

Out[17]:
```
array(['Female', 'Male', 'Male', ..., 'Female', 'Female', 'Female'],
      dtype='<U6')
```

In [18]:
```
#Predicting using train set

target_pred_train = bnb.predict(feature_train)

target_pred_train
```

Out[18]:
```
array(['Female', 'Male', 'Male', ..., 'Female', 'Male', 'Female'],
      dtype='<U6')
```

In [19]:
```
#Checking accuracy of Training Set and Testing Set to check for Overfitting/Underfitting

print('Model Accuracy for Testing Set: {0:0.4f}'. format(accuracy_score(target_test, target_pred)))

print('\nModel Accuracy for Training Set: {0:0.4f}'. format(accuracy_score(target_train, target_pred_train)))
```

```
Model Accuracy for Testing Set: 0.9580

Model Accuracy for Training Set: 0.9611
```

In [20]:
```
#Comparing Null Accuracy and Model Accuracy

target_test.describe()
```

Out[20]:
```
count       1501
unique         2
top       Female
freq         764
Name: gender, dtype: object
```

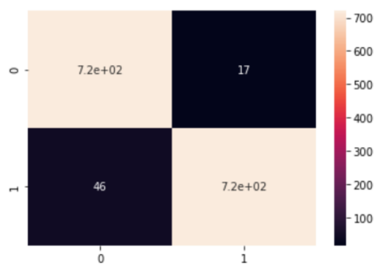
In [21]:
```
#Comparing Null Accuracy and Model Accuracy

null_acc = target_test.describe()[3] / target_test.describe()[0]

print('Model Accuracy: {0:0.4f}'. format(accuracy_score(target_test, target_pred)))

print('\nNull Accuracy: {0:0.4f}'. format(null_acc))
```

```
Model Accuracy: 0.9580

Null Accuracy: 0.5090
```

In [22]:
```python
#Confusion Matrix

cm = confusion_matrix(target_pred, target_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[718  17]
 [ 46 720]]

True Positives(TP) =  718

True Negatives(TN) =  720

False Positives(FP) =  17

False Negatives(FN) =  46
```

In [25]:
```python
#Visualization of Confusion Matrix

sns.heatmap(cm, annot = True)
plt.savefig('confusion.png')
```



In [26]:
```python
#Evaluation Metrics Report

print(classification_report(target_test, target_pred, zero_division = 0))
```

```
              precision    recall  f1-score   support

      Female       0.98      0.94      0.96       764
        Male       0.94      0.98      0.96       737

    accuracy                           0.96      1501
   macro avg       0.96      0.96      0.96      1501
weighted avg       0.96      0.96      0.96      1501
```

Using Gaussian Naive Bayes Classifier

In [27]:
```
#Splitting feature and target variables

feature1 = data.drop(['gender'], axis = 1)

target1 = data['gender']

#Splitting the data into training and testing sets

feature_train1, feature_test1, target_train1, target_test1 = train_test_split(feature1, target1, test_size = 0.30, random_state =
```

In [28]:
```
#Test Size and Training set Info

print("SUMMARY OF SPLITTED TRAINING AND TEST SETS\n")

print('--Number of Training Set:',len(feature_train1))

print('--Training Size:', len(feature_train1)/len(data) * 100, '%')

print('\n--Number of Test Set:',len(feature_test1))

print('--Test Size:', len(feature_test1)/len(data) * 100, '%')
```

```
SUMMARY OF SPLITTED TRAINING AND TEST SETS

--Number of Training Set: 3500
--Training Size: 69.98600279944012 %

--Number of Test Set: 1501
--Test Size: 30.01399720055989 %
```

In [29]:
```
#Creating Naive Bayes Classifier Object using GaussianNB()
gnb = GaussianNB()

#Fitting the Model
gnb.fit(feature_train1, target_train1)
```

Out[29]:
```
GaussianNB()
```

In [30]:
```
#Predicting using test set

target_pred1 = gnb.predict(feature_test1)

target_pred1
```

Out[30]:
```
array(['Female', 'Male', 'Male', ..., 'Female', 'Female', 'Female'],
      dtype='<U6')
```

In [31]:
```
#Predicting using train set

target_pred_train1 = bnb.predict(feature_train1)

target_pred_train1
```

Out[31]:
```
array(['Female', 'Male', 'Male', ..., 'Female', 'Male', 'Female'],
      dtype='<U6')
```

In [32]:
```
#Checking accuracy of Training Set and Testing Set to check for Overfitting/Underfitting

print('Model Accuracy for Testing Set: {0:0.4f}'. format(accuracy_score(target_test1, target_pred1)))

print('\nModel Accuracy for Training Set: {0:0.4f}'. format(accuracy_score(target_train1, target_pred_train1)))
```

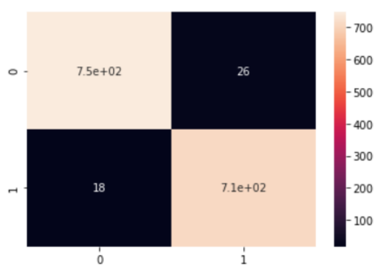
```
Model Accuracy for Testing Set: 0.9707

Model Accuracy for Training Set: 0.9611
```

```
In [33]: #Comparing Null Accuracy and Model Accuracy

         target_test1.describe()

Out[33]: count       1501
         unique         2
         top       Female
         freq         764
         Name: gender, dtype: object
```

```
In [34]: #Comparing Null Accuracy and Model Accuracy

         null_acc1 = target_test1.describe()[3] / target_test1.describe()[0]

         print('Model Accuracy: {0:0.4f}'. format(accuracy_score(target_test1, target_pred1)))

         print('\nNull Accuracy: {0:0.4f}'. format(null_acc1))

         Model Accuracy: 0.9707

         Null Accuracy: 0.5090
```

```
In [35]: #Confusion Matrix

         cm1 = confusion_matrix(target_pred1, target_test1)

         print('Confusion matrix\n\n', cm1)

         print('\nTrue Positives(TP) = ', cm1[0,0])

         print('\nTrue Negatives(TN) = ', cm1[1,1])

         print('\nFalse Positives(FP) = ', cm1[0,1])

         print('\nFalse Negatives(FN) = ', cm1[1,0])

         Confusion matrix

          [[746  26]
          [ 18 711]]

         True Positives(TP) =  746

         True Negatives(TN) =  711

         False Positives(FP) =  26

         False Negatives(FN) =  18
```

```
In [36]: #Visualization of Confusion Matrix

         sns.heatmap(cm1, annot = True)
         plt.savefig('confusion1.png')
```



Using Multinomial Naive Bayes Classifier

In [37]:
```
#Splitting feature and target variables

feature2 = data.drop(['gender'], axis = 1)

target2 = data['gender']

#Splitting the data into training and testing sets

feature_train2, feature_test2, target_train2, target_test2 = train_test_split(feature2, target2, test_size = 0.30, random_state =
```

In [38]:
```
#Test Size and Training set Info

print("SUMMARY OF SPLITTED TRAINING AND TEST SETS\n")

print('--Number of Training Set:',len(feature_train2))

print('--Training Size:', len(feature_train2)/len(data) * 100, '%')

print('\n--Number of Test Set:',len(feature_test2))

print('--Test Size:', len(feature_test2)/len(data) * 100, '%')
```

```
SUMMARY OF SPLITTED TRAINING AND TEST SETS

--Number of Training Set: 3500
--Training Size: 69.98600279944012 %

--Number of Test Set: 1501
--Test Size: 30.01399720055989 %
```

In [39]:
```
#Creating Naive Bayes Classifier Object using MultinomialNB()
mnb = MultinomialNB()

#Fitting the Model
mnb.fit(feature_train2, target_train2)
```

Out[39]:
```
MultinomialNB()
```

In [40]:
```
#Predicting using test set

target_pred2 = mnb.predict(feature_test2)

target_pred2
```

Out[40]:
```
array(['Female', 'Male', 'Male', ..., 'Female', 'Female', 'Female'],
      dtype='<U6')
```

In [41]:
```
#Predicting using train set

target_pred_train2 = mnb.predict(feature_train2)

target_pred_train2
```

Out[41]:
```
array(['Male', 'Male', 'Male', ..., 'Female', 'Male', 'Female'],
      dtype='<U6')
```

In [42]:
```
#Checking accuracy of Training Set and Testing Set to check for Overfitting/Underfitting

print('Model Accuracy for Testing Set: {0:0.4f}'. format(accuracy_score(target_test2, target_pred2)))

print('\nModel Accuracy for Training Set: {0:0.4f}'. format(accuracy_score(target_train2, target_pred_train2)))
```

```
Model Accuracy for Testing Set: 0.9600

Model Accuracy for Training Set: 0.9569
```

In [43]:
```
#Comparing Null Accuracy and Model Accuracy

target_test2.describe()
```

Out[43]:
```
count       1501
unique         2
top       Female
freq         764
Name: gender, dtype: object
```

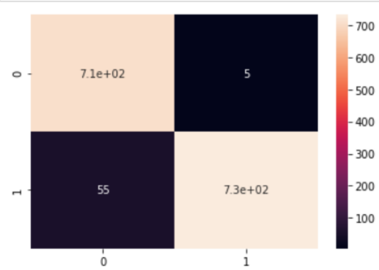
In [44]:
```
#Comparing Null Accuracy and Model Accuracy

null_acc2 = target_test2.describe()[3] / target_test2.describe()[0]

print('Model Accuracy: {0:0.4f}'. format(accuracy_score(target_test2, target_pred2)))

print('\nNull Accuracy: {0:0.4f}'. format(null_acc2))
```

```
Model Accuracy: 0.9600

Null Accuracy: 0.5090
```

In [45]:
```
#Confusion Matrix

cm2 = confusion_matrix(target_pred2, target_test2)

print('Confusion matrix\n\n', cm2)

print('\nTrue Positives(TP) = ', cm2[0,0])

print('\nTrue Negatives(TN) = ', cm2[1,1])

print('\nFalse Positives(FP) = ', cm2[0,1])

print('\nFalse Negatives(FN) = ', cm2[1,0])
```

```
Confusion matrix

 [[709   5]
 [ 55 732]]

True Positives(TP) =  709

True Negatives(TN) =  732

False Positives(FP) =  5

False Negatives(FN) =  55
```

In [46]:
```
#Visualization of Confusion Matrix

sns.heatmap(cm2, annot = True)
plt.savefig('confusion2.png')
```



Summary of Insights ¶

1. A Naive Bayes algorithm is implemented in this project to classify a person's gender based on the given features. Three types of Naive Bayes classifiers were utilized in order to compare which one yields the highest accuracy result. The Gaussian Naive Bayes classifier had the highest among the three, with an accuracy of 97.07%, followed by Multinomial Naive Bayes with 96.00%, and coming at last with a close gap among the two is the Bernoulli Naive Bayes with a 95.80% accuracy.
2. There were no signs of overfitting in all the Naive Bayes Classifiers used, since the values of model accuracies of the train and test sets are comparable, having only a small difference with each other. The model accuracy of the train and test sets in Bernoulli NB is 96.11% and 95.80%, respectively; the model accuracy of the train and test sets in Gaussian NB is 96.11% and 97.07%, respectively and; the model accuracy of the train and test sets in Multinomial NB is 95.69% and 96.00%.
3. It can be concluded that all the Naive Bayes model, using all the its types of classifiers did a good job in predicting the class labels as they were able to produce a lower percentage of null accuracy, when compared to its model accuracy. The null accuracy in all these models is only 50.90%, which is very much lower in all the model accuracies in all the used classifiers as the model accuracy in Bernoulli NB is 95.80%, in Gaussian NB is 97.07%, while in Multinomial NB is 96.00%.