In [1]:
```python
## Import different libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import pickle
from sklearn.utils import resample
```

In [2]:
```python
## Reading the data
data = pd.read_csv('thyroid_data.csv')
```

In [3]:
```python
## Print the first five rows of the data
data
```

Out[3]:

| | S.no | Age | Sex | On Thyroxine | Query on Thyroxine | On Antithyroid Medication | Sick | Pregnant | Thyroid Surgery | I131 Treatment | ... | TSH | Meas |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 41 | F | f | f | f | f | f | f | f | ... | 1.3 | |
| **1** | 1 | 23 | F | f | f | f | f | f | f | f | ... | 4.1 | |
| **2** | 2 | 46 | M | f | f | f | f | f | f | f | ... | 0.98 | |
| **3** | 3 | 70 | F | t | f | f | f | f | f | f | ... | 0.16 | |
| **4** | 4 | 70 | F | f | f | f | f | f | f | f | ... | 0.72 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **3216** | 2774 | 82 | M | f | f | f | f | f | f | f | ... | 2.2 | |
| **3217** | 2776 | 79 | M | f | f | f | f | f | f | f | ... | 1.1 | |
| **3218** | 2782 | 50 | F | f | f | f | f | f | f | f | ... | 4.8 | |
| **3219** | 2786 | 73 | ? | f | f | f | f | f | f | f | ... | 0.015 | |
| **3220** | 2796 | 73 | M | f | t | f | f | f | f | f | ... | ? | |

3221 rows × 28 columns

In [4]:
```python
## Shape of the data
data.shape
```

Out[4]:
```
(3221, 28)
```

In [5]:
```
n = len(data[data['Category'] == 'hyperthyroid'])
print("No of hyperthyroid in Dataset:",n)

n1 = len(data[data['Category'] == 'hypothyroid'])
print("No of hypothyroid in Dataset:",n1)

n2 = len(data[data['Category'] == 'sick'])
print("No of sick in Dataset:",n2)

n3 = len(data[data['Category'] == 'negative'])
print("No of negative in Dataset:",n3)
```

```
No of hyperthyroid in Dataset: 77
No of hypothyroid in Dataset: 220
No of sick in Dataset: 171
No of negative in Dataset: 2753
```

***The data has 3772 rows and 30 columns.***

In [6]:
```
## Columns
data.columns
```

Out[6]:
```
Index(['S.no', 'Age', 'Sex', 'On Thyroxine', 'Query on Thyroxine',
       'On Antithyroid Medication', 'Sick', 'Pregnant', 'Thyroid Surgery',
       'I131 Treatment', 'Query Hypothyroid', 'Query Hyperthyroid', 'Lithium',
       'Goitre', 'Tumor', 'Hypopituitary', 'Psych', 'TSH Measured', 'TSH',
       'T3 Measured', 'T3', 'TT4 Measured', 'TT4', 'T4U Measured', 'T4U',
       'FTI Measured', 'FTI', 'Category'],
      dtype='object')
```

In [7]:
```
## Checking the missing values
data.isnull().sum()
```

Out[7]:
```
S.no                          0
Age                           0
Sex                           0
On Thyroxine                  0
Query on Thyroxine            0
On Antithyroid Medication     0
Sick                          0
Pregnant                      0
Thyroid Surgery               0
I131 Treatment                0
Query Hypothyroid             0
Query Hyperthyroid            0
Lithium                       0
Goitre                        0
Tumor                         0
Hypopituitary                 0
Psych                         0
TSH Measured                  0
TSH                           0
T3 Measured                   0
T3                            0
TT4 Measured                  0
TT4                           0
T4U Measured                  0
T4U                           0
FTI Measured                  0
FTI                           0
Category                      0
dtype: int64
```

***We can see that there are no missing values. But if we see the dataset the missing values are replaced with the invalid values like '?'. Let's replace such values with 'nan' and check for the missing values again.***

In [8]:
```
for column in data.columns:
    count=data[column][data[column]=='?'].count()
    if count != 0:
        print(column,data[column][data[column]=='?'].count())
```

```
Age 1
Sex 127
TSH 247
T3 589
TT4 142
T4U 276
FTI 274

C:\Users\DELL\anaconda3\aaaaaaaaaaaaa\lib\site-packages\pandas\core\ops\array_ops.py:253:
FutureWarning: elementwise comparison failed; returning scalar instead, but in the future
will perform elementwise comparison
  res_values = method(rvalues)
```

In [9]:
```python
## Let's drop some unnecessary columns
data=data.drop([ 'S.no','On Thyroxine', 'Query on Thyroxine',
       'On Antithyroid Medication',
       'I131 Treatment', 'Query Hypothyroid', 'Query Hyperthyroid', 'Lithium',
        'TSH Measured','Hypopituitary','Psych',
       'T3 Measured',  'TT4 Measured', 'T4U Measured',
       'FTI Measured'],axis=1)
```

In [10]:
```python
## Now replace the '?' values with numpy nan
for column in data.columns:
    count=data[column][data[column]== '?'].count()
    if count!=0:
        data[column]=data[column].replace('?',np.nan)
```

In [11]:
```python
for column in data.columns:
    count=data[column][data[column]=='?'].count()
    if count == 0:
        print(column,data[column][data[column]=='?'].count())
```

```
Age 0
Sex 0
Sick 0
Pregnant 0
Thyroid Surgery 0
Goitre 0
Tumor 0
TSH 0
T3 0
TT4 0
T4U 0
FTI 0
Category 0
```

***So, we have replaced all such values with 'nan' values.***

```
In [12]: ##  Unique Values
         for column in data.columns:
             print(column,(data[column].unique()))
```

```
Age ['41' '23' '46' '70' '18' '59' '80' '66' '68' '84' '67' '71' '28' '65'
 '42' '63' '51' '81' '54' '55' '60' '25' '73' '34' '78' '37' '85' '26'
 '58' '64' '44' '48' '61' '35' '83' '21' '87' '53' '77' '27' '69' '74'
 '38' '76' '45' '36' '22' '43' '72' '82' '31' '39' '49' '62' '57' '1' '50'
 '30' '29' '75' '19' '7' '79' '17' '24' '15' '32' '47' '16' '52' '33' '13'
 '10' '89' '56' '20' '90' '40' '88' '14' '86' '94' '12' '4' '11' '8' '5'
 '455' '2' '91' '6' nan '93' '92']
Sex ['F' 'M' nan]
Sick ['f' 't']
Pregnant ['f' 't']
Thyroid Surgery ['f' 't']
Goitre ['f' 't']
Tumor ['f' 't']
TSH ['1.3' '4.1' '0.98' '0.16' '0.72' '0.03' nan '2.2' '0.6' '2.4' '1.1' '2.8'
 '3.3' '12' '1.2' '1.5' '6' '2.1' '0.1' '0.8' '1.9' '3.1' '0.2' '13' '0.3'
 '0.035' '2.5' '0.5' '1.7' '7.3' '1.8' '0.26' '45' '5.4' '0.99' '0.25'
 '0.92' '0.15' '0.64' '1' '0.4' '2' '2.6' '14.8' '15' '19' '0.02' '3'
 '2.9' '3.2' '9' '1.6' '4.3' '0.005' '0.31' '0.61' '0.05' '7.8' '160'
 '0.025' '1.4' '0.01' '8.8' '151' '0.04' '3.9' '9.4' '2.7' '2.3' '0.94'
 '0.045' '3.5' '0.88' '0.08' '4.5' '0.68' '0.7' '0.67' '27' '6.1' '0.75'
 '0.55' '26' '5.2' '0.77' '0.07' '0.9' '11.4' '143' '0.45' '0.57' '0.65'
 '0.015' '16' '108' '0.83' '9.2' '86' '0.62' '0.59' '9.1' '5.9' '52'
 '0.33' '31' '5.8' '0.28' '51' '6.3' '4.4' '9.6' '3.4' '0.09' '24' '0.76'
 '42' '25' '10' '4.6' '8.6' '0.66' '6.2' '0.79' '28' '0.86' '9.7' '0.84'
 '17' '18' '55' '14' '3.7' '0.87' '6.7' '0.74' '7.6' '0.065' '0.29' '0.37'
 '8' '11' '0.48' '44' '7.9' '5' '7.2' '0.89' '0.93' '0.97' '0.12' '6.4'
 '33' '0.85' '7.1' '0.73' '199' '8.2' '188' '0.22' '98' '22' '6.6' '5.1'
 '0.06' '0.42' '3.8' '35' '4' '0.78' '0.63' '0.52' '60' '0.43' '5.6' '6.9'
 '3.6' '29' '0.38' '4.9' '0.41' '9.9' '7.5' '34' '6.5' '4.7' '103' '0.95'
 '0.14' '0.35' '4.2' '0.81' '0.54' '0.58' '8.9' '5.5' '0.34' '9.3' '0.13'
 '54' '0.39' '8.3' '478' '21' '6.8' '0.32' '0.23' '0.24' '8.1' '0.91'
 '5.3' '100' '0.27' '1.01' '58' '41' '183' '18.4' '0.47' '0.17' '12.1'
 '0.19' '0.82' '43' '0.44' '70' '7.7' '8.4' '0.69' '8.5' '0.21' '82'
 '0.055' '0.96' '0.71' '38' '0.36' '9.8' '7' '0.46' '11.1' '39' '76' '5.7'
 '32' '126' '26.4' '0.53' '0.49' '36' '178' '145' '47' '4.8' '10.3' '89'
 '7.4' '472' '0.51' '116' '61' '99' '46' '78' '468']
T3 ['2.5' '2' nan '1.9' '1.2' '0.6' '2.2' '1.6' '3.8' '1.7' '1.8' '2.6' '2.1'
 '0.3' '5.5' '1.4' '3.1' '1.5' '2.3' '2.4' '2.7' '0.9' '1' '2.8' '2.9'
 '0.8' '1.3' '0.4' '3.3' '3.5' '3.4' '1.1' '4.2' '3.7' '3' '0.7' '4.8'
 '4.3' '0.05' '3.2' '5.4' '4' '0.5' '0.2' '3.6' '5.2' '5' '6' '5.3' '3.9'
 '4.6' '4.5' '7.3' '4.7' '6.7' '4.1' '6.1' '0.1' '4.9' '10.6' '5.1' '7'
 '6.2' '4.4' '7.1']
TT4 ['125' '102' '109' '175' '61' '183' '72' '80' '123' '83' '115' '152' '171'
 '97' '99' '70' '117' '121' '130' '108' '104' '134' '199' '57' '129' '113'
 '119' '84' '81' '95' '66' '101' '147' '120' '69' nan '39' '87' '63' '133'
 '86' '163' '162' '103' '96' '151' '112' '82' '138' '71' '77' '93' '107'
 '237' '110' '67' '88' '160' '118' '136' '114' '116' '94' '161' '11' '32'
 '124' '137' '92' '135' '105' '150' '126' '146' '91' '217' '141' '159'
 '122' '100' '111' '140' '205' '225' '85' '90' '74' '219' '127' '132'
 '128' '106' '144' '131' '56' '79' '142' '98' '177' '139' '78' '189' '180'
 '73' '145' '184' '38' '156' '75' '148' '14' '76' '54' '58' '27' '65'
 '193' '13' '143' '12' '64' '257' '164' '59' '167' '18' '41' '176' '37'
 '33' '44' '45' '154' '174' '203' '244' '62' '158' '60' '187' '250' '181'
 '157' '223' '272' '166' '213' '235' '10' '68' '231' '191' '48' '5.8'
 '169' '149' '210' '40' '155' '232' '42' '204' '430' '198' '230' '15'
 '170' '165' '47' '168' '194' '89' '52' '179' '192' '172' '4.8' '50' '182'
 '197' '214' '246' '196' '207' '19' '153' '22' '46' '200' '35' '226' '201'
 '233' '206' '31' '255' '178' '239' '195' '6' '36' '2' '3' '289' '240'
 '209' '43' '34' '252' '29' '263' '301' '23' '188' '211' '253' '21' '173']
T4U ['1.14' nan '0.91' '0.87' '1.3' '0.92' '0.7' '0.93' '0.89' '0.95' '0.99'
 '1.13' '0.86' '0.96' '0.94' '0.9' '1.02' '1.05' '0.62' '1.06' '1.55'
 '0.83' '1.09' '1.07' '1.27' '0.76' '1.16' '1' '0.56' '0.81' '0.68' '0.78'
 '0.85' '1.35' '1.15' '0.82' '1.03' '1.58' '0.79' '1.17' '0.71' '0.72'
 '0.88' '1.11' '1.2' '1.1' '1.33' '0.77' '1.24' '0.53' '1.44' '1.63'
 '1.51' '1.42' '1.23' '1.01' '0.98' '0.61' '1.12' '1.43' '1.25' '1.41'
 '1.68' '0.97' '0.84' '0.8' '1.04' '0.73' '1.08' '1.26' '1.46' '1.29'
 '1.34' '1.66' '1.21' '1.19' '0.75' '0.52' '1.83' '1.39' '1.5' '1.93'
 '1.18' '0.74' '0.58' '1.82' '0.6' '1.67' '1.22' '0.66' '0.67' '1.31'
 '0.54' '1.77' '1.59' '1.97' '1.69' '1.38' '1.28' '1.4' '0.69' '0.65'
 '1.74' '2.03' '1.73' '1.65' '1.36' '1.52' '0.57' '1.53' '1.84' '1.57'
 '1.75' '1.32' '1.37' '0.64' '1.79' '1.8' '0.48' '1.71' '1.62' '1.76'
 '1.56' '1.48' '0.59' '0.31' '1.94' '2.12' '1.47' '0.63' '0.944' '0.49'
 '1.88' '0.5' '0.38' '1.49' '0.41' '1.61' '1.7']
FTI ['109' nan '120' '70' '141' '78' '115' '132' '93' '121' '153' '151' '107'
 '119' '87' '81' '104' '130' '106' '116' '131' '190' '92' '102' '76' '98'
 '90' '61' '94' '129' '95' '91' '33' '113' '148' '140' '171' '155' '186'
 '122' '136' '110' '111' '97' '72' '100' '88' '67' '84' '103' '135' '203'
 '112' '117' '180' '142' '145' '156' '96' '134' '8.9' '60' '139' '41' '99'
 '89' '146' '124' '105' '85' '157' '143' '71' '221' '28' '108' '137' '83'
 '74' '170' '65' '101' '127' '274' '154' '114' '62' '86' '126' '125' '64'
 '172' '162' '79' '118' '73' '152' '163' '149' '14' '51' '165' '77' '32'
 '69' '80' '11' '54' '164' '123' '144' '10' '214' '200' '160' '53' '16'
 '138' '169' '56' '47' '133' '43' '68' '179' '224' '220' '82' '362' '182'
 '75' '66' '161' '57' '58' '312' '63' '128' '147' '158' '281' '207' '216'
 '251' '194' '46' '7' '42' '174' '395' '185' '13' '201' '48' '173' '167'
 '188' '150' '235' '175' '159' '5.4' '189' '59' '166' '34' '228' '232'
 '217' '177' '176' '195' '219' '17' '210' '168' '205' '39' '187' '50'
 '349' '52' '206' '253' '242' '244' '213' '178' '247' '215' '198' '19'
 '237' '37' '7.6' '24' '2' '3' '191' '223' '9' '29' '222' '204' '26' '218'
 '197' '49' '209' '183']
Category ['negative' 'hyperthyroid' 'hypothyroid' 'sick']
```

In [13]:
```
data.dtypes
```

Out[13]:
```
Age                 object
Sex                 object
Sick                object
Pregnant            object
Thyroid Surgery     object
Goitre              object
Tumor               object
TSH                 object
T3                  object
TT4                 object
T4U                 object
FTI                 object
Category            object
dtype: object
```

## Handling Missing Values

In [14]:
```
data['Age'].fillna((data['Age'].median()), inplace = True)
data['TSH'].fillna((data['TSH'].median()), inplace = True)
data['T3'].fillna((data['T3'].median()), inplace = True)
data['TT4'].fillna((data['TT4'].median()), inplace = True)
data['T4U'].fillna((data['T4U'].median()), inplace = True)
data['FTI'].fillna((data['FTI'].median()), inplace = True)
```

### Handling nominal categorical variables

In [15]:
```
## We will perform one hot encoding for nominal categorical variable.
sex = data[["Sex"]]
sex = pd.get_dummies(sex, drop_first= True)
```

In [16]:
```
sick = data[["Sick"]]
sick = pd.get_dummies(sick, drop_first= True)
```

In [17]:
```
pregnant = data[["Pregnant"]]
pregnant = pd.get_dummies(pregnant, drop_first= True)
```

In [18]:
```
thyroid_surgery = data[["Thyroid Surgery"]]
thyroid_surgery = pd.get_dummies(thyroid_surgery, drop_first= True)
```

In [19]:
```
goitre = data[["Goitre"]]
goitre = pd.get_dummies(goitre, drop_first= True)
```

In [20]:
```
tumor = data[["Tumor"]]
tumor = pd.get_dummies(tumor, drop_first= True)
```

In [21]:
```
data.columns
```

Out[21]:
```
Index(['Age', 'Sex', 'Sick', 'Pregnant', 'Thyroid Surgery', 'Goitre', 'Tumor',
       'TSH', 'T3', 'TT4', 'T4U', 'FTI', 'Category'],
      dtype='object')
```

In [22]:
```
final_df = pd.concat([data,sex, sick, pregnant, thyroid_surgery,
       goitre, tumor,], axis = 1)
```

In [23]:
```
final_df.columns
```

Out[23]:
```
Index(['Age', 'Sex', 'Sick', 'Pregnant', 'Thyroid Surgery', 'Goitre', 'Tumor',
       'TSH', 'T3', 'TT4', 'T4U', 'FTI', 'Category', 'Sex_M', 'Sick_t',
       'Pregnant_t', 'Thyroid Surgery_t', 'Goitre_t', 'Tumor_t'],
      dtype='object')
```

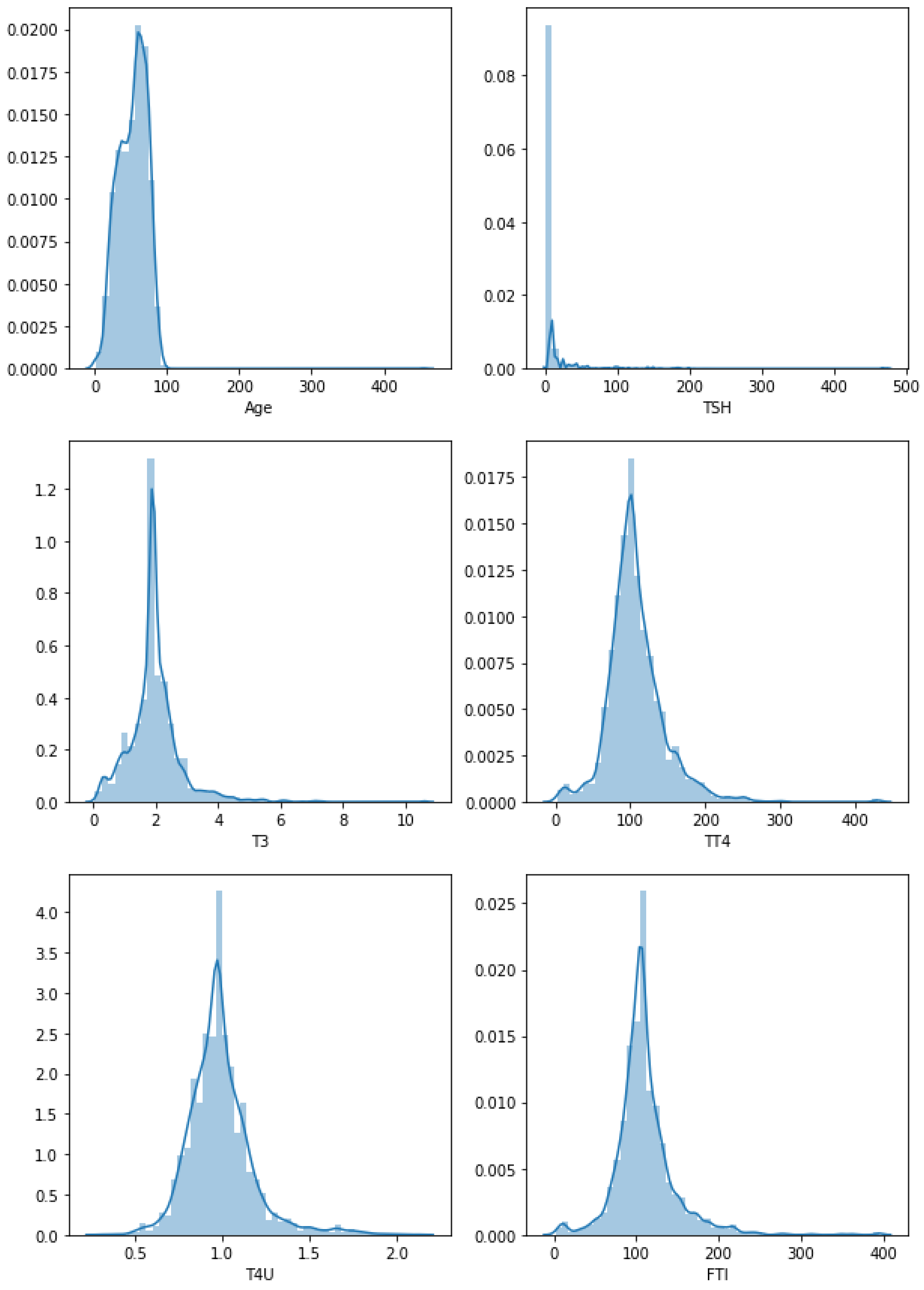
In [24]:
```
final_df.drop(['Sex', 'Sick', 'Pregnant', 'Thyroid Surgery',
       'Goitre', 'Tumor'], axis = 1, inplace = True)
```

```
In [25]: ## Checking the missing values
         final_df.isnull().sum()
```

```
Out[25]: Age                   0
         TSH                   0
         T3                    0
         TT4                   0
         T4U                   0
         FTI                   0
         Category              0
         Sex_M                 0
         Sick_t                0
         Pregnant_t            0
         Thyroid Surgery_t     0
         Goitre_t              0
         Tumor_t               0
         dtype: int64
```

***Great! Now the data has no missing values.***

In [26]:
```python
## Let's check the distribution for our continuous data in the dataset.
columns = ['Age','TSH','T3','TT4','T4U','FTI']

plt.figure(figsize=(10,15),facecolor='white')
plotnumber = 1

for column in columns:
    ax = plt.subplot(3,2,plotnumber)
    sns.distplot(final_df[column])
    plt.xlabel(column,fontsize=10)
    plotnumber+=1
plt.show()
```



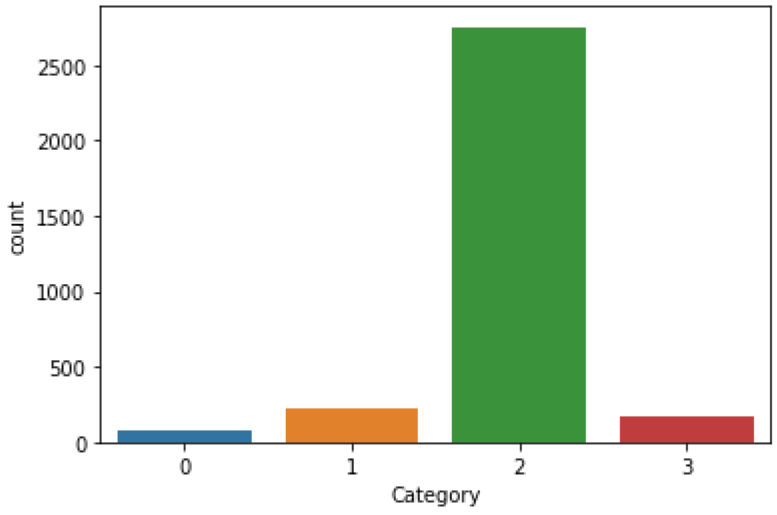
***The graph TSH heavyly skewed towards left. We drop it.***

```
In [27]: final_df = final_df.drop(['TSH'], axis = 1)
```

In [28]:
```
from sklearn.preprocessing import LabelEncoder
lblEn=LabelEncoder()
final_df['Category']=lblEn.fit_transform(final_df['Category'])
```

In [29]:
```
sns.countplot(final_df['Category'])
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2cf60e77748>
```

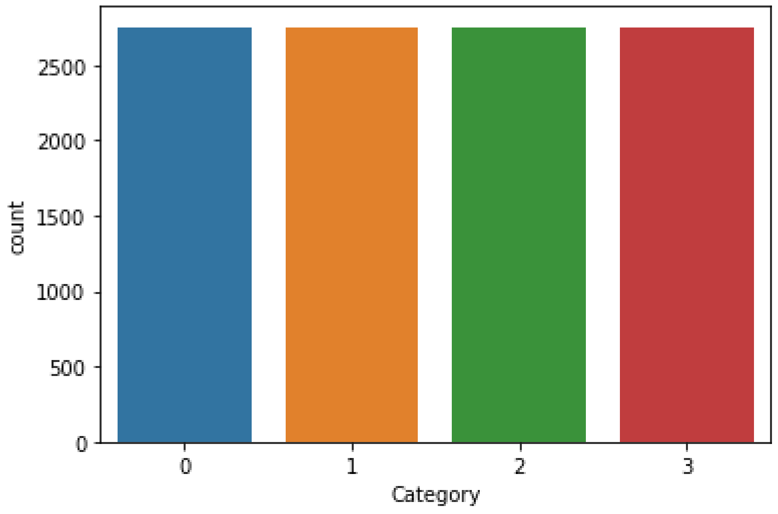


***We can see that the dataset is highly imbalanced.***

In [30]:
```
X=final_df.drop(['Category'],axis=1)
y=final_df.Category
```

In [31]:
```
# 42 input

from imblearn.over_sampling import SMOTENC,RandomOverSampler,KMeansSMOTE


rdsmple=RandomOverSampler()
X_sampled,y_sampled=rdsmple.fit_sample(X,y)

X_sampled.shape


X_sampled=pd.DataFrame(data=X_sampled,columns=X.columns)


X_sampled


sns.countplot(y_sampled)


X_sampled.columns
```

Out[31]:
```
Index(['Age', 'T3', 'TT4', 'T4U', 'FTI', 'Sex_M', 'Sick_t', 'Pregnant_t',
       'Thyroid Surgery_t', 'Goitre_t', 'Tumor_t'],
      dtype='object')
```



In [ ]:

In [ ]:

In [ ]:

***Great! Our dataset is balanced now.***

In [ ]:

In [32]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_sampled,y_sampled,test_size=0.2,random_sta
```

## Fitting data in various models

In [33]:
```
def svm_classifier(X_train,X_test,y_train,y_test):
    classifier_svm=SVC(kernel='rbf',random_state=0)
    classifier_svm.fit(X_train,y_train)
    y_pred=classifier_svm.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)
    return (f'Train Score:{classifier_svm.score(X_train,y_train)}\n Test Score:{classifier_
```

In [34]:
```
def knn_classifier(X_train,X_test,y_train,y_test):
    classifier_knn=KNeighborsClassifier(metric='minkowski',p=2)
    classifier_knn.fit(X_train,y_train)
    y_pred=classifier_knn.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)
    return (f'Train Score:{classifier_knn.score(X_train,y_train)}\n Test Score:{classifier_
```

In [35]:
```
def tree_classifier(X_train,X_test,y_train,y_test):
    classifier_tree=DecisionTreeClassifier(criterion='entropy',random_state=0)
    classifier_tree.fit(X_train,y_train)
    y_pred=classifier_tree.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)
    return (f'Train Score:{classifier_tree.score(X_train,y_train)}\n Test Score:{classifier
```

In [36]:
```
def forest_classifier(X_train,X_test,y_train,y_test):
    classifier_forest=RandomForestClassifier(criterion='entropy',random_state=0)
    classifier_forest.fit(X_train,y_train)
    y_pred=classifier_forest.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)
    return (f'Train Score:{classifier_forest.score(X_train,y_train)}\n Test Score:{classifi
```

In [37]:
```python
def print_score(X_train, X_test, y_train, y_test):
    print("SVM:\n")
    result1=svm_classifier(X_train, X_test, y_train, y_test)
    print(result1)
    print("-"*100)
    print()

    print("KNN:\n")
    result2=knn_classifier(X_train, X_test, y_train, y_test)
    print(result2)

    print("-"*100)
    print()

    print("Decision Tree:\n")
    result4=tree_classifier(X_train, X_test, y_train, y_test)
    print(result4)

    print("-"*100)
    print()

    print("Random Forest:\n")
    result5=forest_classifier(X_train, X_test, y_train, y_test)
    print(result5)
```

In [38]:
```python
print_score(X_train, X_test, y_train, y_test)
```

```
SVM:

Train Score:0.6128959019184924
 Test Score:0.6173399909214707
-----------------------------------------------------------------------------------------
-----------

KNN:

Train Score:0.8712680213418095
 Test Score:0.8551974580118021
-----------------------------------------------------------------------------------------
-----------

Decision Tree:

Train Score:0.9159950051084118
 Test Score:0.8915115751248298
-----------------------------------------------------------------------------------------
-----------

Random Forest:

Train Score:0.9159950051084118
 Test Score:0.902859736722651
```

In [39]:
```
## Performance Metrics
classifier_forest = RandomForestClassifier(criterion = 'entropy')
classifier_forest.fit(X_train,y_train)
y_pred = classifier_forest.predict(X_test)
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[39]:
```
array([[563,   0,   0,   0],
       [  0, 513,   0,  40],
       [ 21,  48, 424,  39],
       [ 50,  15,   0, 490]], dtype=int64)
```

In [40]:
```
## Classification Report (Accuracy, Precision, Recall and F1 Score)
from sklearn.metrics import roc_auc_score,roc_curve,classification_report
```

In [41]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       563
           1       0.89      0.93      0.91       553
           2       1.00      0.80      0.89       532
           3       0.86      0.88      0.87       555

    accuracy                           0.90      2203
   macro avg       0.91      0.90      0.90      2203
weighted avg       0.91      0.90      0.90      2203
```

**Hyperparameter Tuning**

In [42]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score (estimator = classifier_forest, X=X_train,y=y_train,cv=10)
print(accuracies.mean())
```

```
0.9014641161902794
```

In [43]:
```
import pickle
filename = 'thyroid_model.pkl'
pickle.dump(classifier_forest,open(filename,'wb'))
```

In [44]:
```
model = open('thyroid_model.pkl','rb')
forest = pickle.load(model)
```

In [45]:
```
y_pred = forest.predict(X_test)
```

In [46]:
```
confusion_matrix(y_test, y_pred)
```

Out[46]:
```
array([[563,   0,   0,   0],
       [  0, 513,   0,  40],
       [ 21,  48, 424,  39],
       [ 50,  15,   0, 490]], dtype=int64)
```

```
In [47]: X.columns

Out[47]: Index(['Age', 'T3', 'TT4', 'T4U', 'FTI', 'Sex_M', 'Sick_t', 'Pregnant_t',
                'Thyroid Surgery_t', 'Goitre_t', 'Tumor_t'],
               dtype='object')

In [48]: print(forest.predict([[41,2.5,125,1.14,109,0,0,0,0,0,0]]))

         [2]

In [49]: print(forest.predict([[63,5.5,199,1.05,190,0,0,0,0,0,0]]))

         [0]

In [50]: print(forest.predict([[44,1.4,39,1.16,33,1,0,0,0,0,0]]))

         [1]

In [51]: print(forest.predict([[61,1,96,0.93,109,1,1,0,0,0,0]]))

         [3]

In [ ]:
```