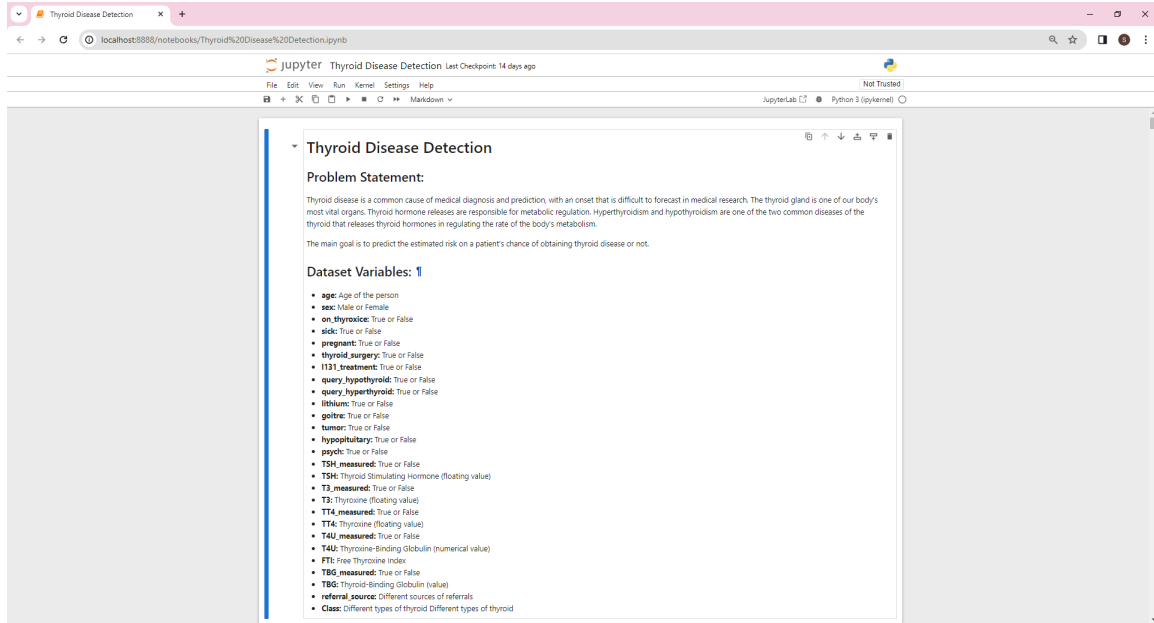


# Thyroid Disease Detection

## Wireframe Documentation



Jupyter Thyroid Disease Detection Last Checkpoint: 15 days ago

File Edit View Run Kernel Settings Help

Not Trusted

Python 3 (ipykernel)

• referral\_source: Different sources of referrals  
• Class: Different types of thyroid

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.utils import resample
from imblearn.over_sampling import SMOTENC, RandomOverSampler, KMeansSMOTE
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
sns.set()
```

```
[2]: data = pd.read_csv('hypothyroid.csv')
```

```
[3]: data.shape
```

```
[3]: (3772, 30)
```

```
[4]: data.head()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	h131_treatment	query_hypothyroid	TT4_measured	TT4
0	41	F	f	f	f	f	f	f	f	f	...	t 125
1	23	F	f	f	f	f	f	f	f	f	...	t 102
2	46	M	f	f	f	f	f	f	f	f	...	t 109
3	70	F	t	f	f	f	f	f	f	f	...	t 175
4	70	F	f	f	f	f	f	f	f	f	...	t 61

5 rows × 30 columns

```
[5]: data.describe()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	h131_treatment	query_hypothyroid	TT4_measured	TT4
count	3772	3772	3772	3772	3772	3772	3772	3772	3772	3772	...	37
unique	94	3	2	2	2	2	2	2	2	2	...	2
top	59	F	f	f	f	f	f	f	f	f	...	...
freq	95	2480	3308	3722	3729	3625	3719	3719	3713	3538	...	35

4 rows × 30 columns

Thyroid Disease Detection

localhost:8888/notebooks/Thyroid%20Disease%20Detection.ipynb

Jupyter Thyroid Disease Detection Last Checkpoint: 15 days ago

File Edit View Run Kernel Settings Help

Not Trusted

Python 3 (ipykernel)

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

```
age 1
sex 150
TSH 369
T3 769
TT4 231
T4U 387
FTI 385
TBG 3772
```

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

So these are the columns which have missing values but missing values are replaced with '?'. We will replace these values with 'nan' and then do imputation of these missing values.

Also, we can see that for column 'TBG' all the values are missing. So we will drop this column as it is of no use to us.

```
[7]: data = data.drop(['TBG'],axis=1)
```

```
[8]: data[['T4U_measured', 'T4U']]
```

	T4U_measured	T4U
0	t 1.14	
1	f ?	
2	t 0.91	
3	f ?	
4	t 0.87	
...	...	...
3767	f ?	
3768	t 1.08	
3769	t 1.07	
3770	t 0.94	

Thyroid Disease Detection

localhost:8888/notebooks/Thyroid%20Disease%20Detection.ipynb

Jupyter Thyroid Disease Detection Last Checkpoint: 15 days ago

File Edit View Run Kernel Settings Help

3772 rows x 2 columns

Since, we are any ways going to handle the missing values, there is no point of having such columns in our dataset. Let's drop such columns as well.

```
[9]: data = data.drop(['TSH_measured', 'T3_measured', 'T4_measured', 'FTI_measured', 'T86_measured'], axis=1)
```

```
[10]: # Now let's 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)
```

```
[11]: 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  
on\_thyroxine 0  
query\_on\_thyroxine 0  
on\_antithyroid\_medication 0  
sick 0  
pregnant 0  
thyroid\_surgery 0  
I131\_treatment 0  
query\_hyperthyroid 0  
query\_hypothyroid 0  
lithium 0  
goitre 0  
tumor 0  
hypopituitary 0  
psych 0  
TSH 0  
T3 0  
T4 0  
FTI 0  
referral\_source 0  
Class 0

Great!! Now that we have replaced all such values with 'nan'. Let's deal with these missing values now. Now that we have replaced all such values with 'nan'. Let's deal with these missing values now.

```
[12]: data.isna().sum()
```

```
[12]: age      1
sex      150
on_thyroxine      0
query_on_thyroxine      0
on_antithyroid_medication      0
```

Thyroid Disease Detection

localhost:8888/notebooks/Thyroid%20Disease%20Detection.ipynb

Jupyter Thyroid Disease Detection Last Checkpoint: 15 days ago

File Edit View Run Kernel Settings Help

Not Trusted

referral\_source 0  
Class 0  
dtype: int64

Since the values are categorical, we have to change them to numerical before we use any imputation techniques.

We can use get dummies but since most of the columns have only two distinct categories we will use mapping for them. Why? Because since there are only two categories then the two columns formed after get dummies will both have very high correlation since they both explain the same thing. So in anyway we will have to drop one of the columns. That's why let's use mapping for such columns. For columns with more than two categories we will use get dummies.

```
[13]: # for column in data.columns:
#     print(column, (data[column].unique()))
```

```
[14]: # We can map the categorical values like below:
data['sex'] = data['sex'].map({'F': 0, 'M': 1})

# except for 'Sex' column all the other columns with two categorical data have same value 'f' and 't'.
# so instead of mapping individually, let's do a smarter work
for column in data.columns:
    if len(data[column].unique())==2:
        data[column] = data[column].map({'f': 0, 't': 1})

# this will map all the rest of the columns as we require. Now there are handful of column left with more than 2 categories.
```

```
[15]: data['referral_source'].unique()
```

```
[15]: array(['SVHC', 'other', 'SVI', 'STMW', 'SVHD'], dtype=object)
```

```
[16]: # we will use get_dummies with that.
data = pd.get_dummies(data, columns=['referral_source'])
```

Now our output class also has 4 distinct categories. There is no sense of using get dummies with our Output class, so we will just map them. Let's use LabelEncoder function for this.

```
[17]: data['Class'].unique()
```

Thyroid Disease Detection

localhost:8888/notebooks/Thyroid%20Disease%20Detection.ipynb

Jupyter Thyroid Disease Detection Last Checkpoint: 15 days ago

File Edit View Run Kernel Settings Help

Not Trusted

JupyterLab Python 3 (ipykernel)

```
[18]: lblEn = LabelEncoder()
data['Class'] = lblEn.fit_transform(data['Class'])

[19]: data.head()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	I131_treatment	query_hypothyroid	...	T3	TT4	T4U	F1
0	41	0.0	0	0	0	0	0	0	0	0	...	2.5	125	1.14	10
1	23	0.0	0	0	0	0	0	0	0	0	...	2	102	NaN	NaN
2	46	1.0	0	0	0	0	0	0	0	0	...	NaN	109	0.91	12
3	70	0.0	1	0	0	0	0	0	0	0	...	1.9	175	NaN	NaN
4	70	0.0	0	0	0	0	0	0	0	0	...	1.2	61	0.87	7

5 rows x 27 columns

```
[20]: data.describe(include='all')
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	I131_treatment	query_hypothyroid
count	3771	3622.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000
unique	93	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	59	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	95	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	0.315295	0.123012	0.013256	0.011400	0.038971	0.014051	0.014051	0.015642	0.062036
std	NaN	0.464698	0.328494	0.114382	0.106174	0.193552	0.117716	0.117716	0.124101	0.241253
min	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Thyroid Disease Detection

localhost:8888/notebooks/Thyroid%20Disease%20Detection.ipynb

Jupyter Thyroid Disease Detection Last Checkpoint: 15 days ago

File Edit View Run Kernel Settings Help

Not Trusted

JupyterLab Python 3 (ipykernel)

```
[21]: # for column in data.columns:
#     print(column, (data[column].unique()))
```

Great! Now that we have encoded all our Categorical values. Let's start with imputing the missing values.

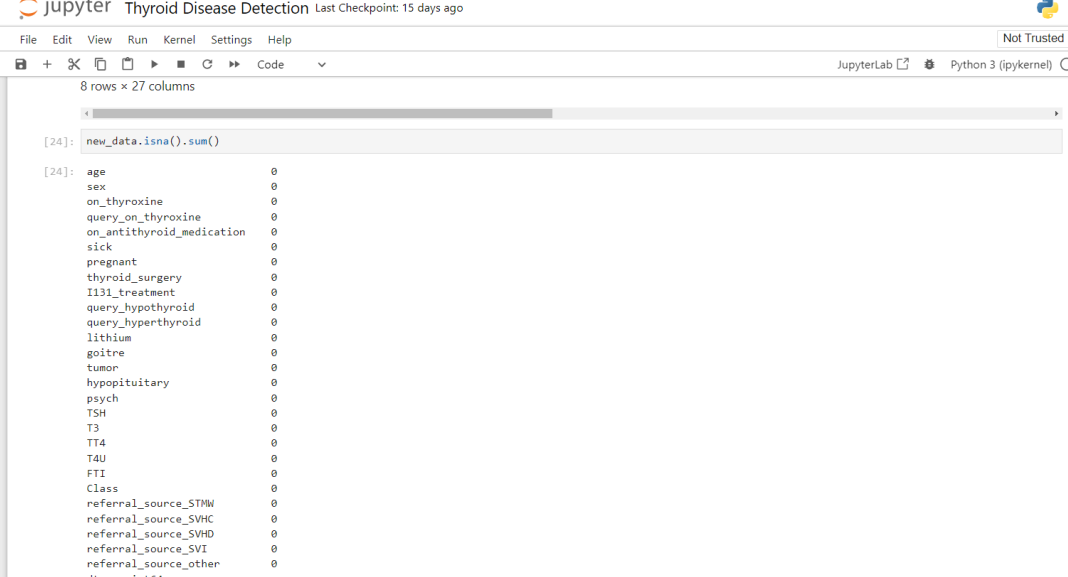
```
[22]: imputer=KNNImputer(n_neighbors=3, weights='uniform', missing_values=np.nan)
new_array=imputer.fit_transform(data) # impute the missing values
# convert the nd-array returned in the step above to a Dataframe
new_data=spd.DataFrame(data=np.round(new_array), columns=data.columns)
```

```
[23]: new_data.describe()
```

	age	sex	on_thyroxine	query_on_thyroxine	on_antithyroid_medication	sick	pregnant	thyroid_surgery	I131_treatment	query_hypothyroid
count	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.000000	3772.00
mean	51.737275	0.307529	0.123012	0.013256	0.011400	0.038971	0.014051	0.014051	0.015642	0.06
std	20.082478	0.461532	0.328494	0.114382	0.106174	0.193552	0.117716	0.117716	0.124101	0.24
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	36.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	54.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	67.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
max	455.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

8 rows x 27 columns

```
[24]: new_data.isna().sum()
```



The screenshot shows a JupyterLab environment with a file browser on the left, a code editor in the center, and a terminal on the right. The file browser displays a directory structure with files like 'Thyroid Disease Detection.ipynb' and 'Thyroid Disease Detection.py'. The code editor contains a Python script that filters out missing values from a dataset. The terminal shows the output of the script, which is a list of 27 variables, all with a count of 0, indicating that all missing values have been removed. The interface includes a file browser, a code editor, and a terminal.

```

[24]: new_data.isna().sum()

[24]: 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 0
T3 0
TT4 0
T4U 0
FTI 0
Class 0
referral_source_STMW 0
referral_source_SVMC 0
referral_source_SVHC 0
referral_source_SVI 0
referral_source_other 0
dtype: int64

```

Great! Now there are no missing values in our new dataset.

Let's check the distribution for our continuous data in the dataset:

```
[24]: age                                0
sex                                    0
on_thyroxine                         0
query_on_thyroxine                   0
on_antithyroid_medication            0
sick                                  0
pregnant                             0
thyroid_surgery                      0
t13t4_treatment                     0
query_hypothyroid                    0
query_hyperthyroid                   0
lithium                              0
goitre                               0
tumor                                0
hypopituitary                       0
psych                                0
TSH                                  0
T3                                    0
TT4                                   0
T4U                                   0
FTI                                   0
Class                                0
referral_source_STMW                 0
referral_source_SVHC                 0
referral_source_SVHD                 0
referral_source_SVI                  0
referral_source_other                0
dtype: object
```

Great! Now there are no missing values in our new dataset.

Let's check the distribution for our continous data in the dataset:

Thyroid Disease Detection

localhost:8888/notebooks/Thyroid%20Disease%20Detection.ipynb

Thyroid Disease Detection Last Checkpoint: 15 days ago

Not Trusted

JupyterLab Python 3 (pykernel)

```
[64]: best_model.predict([[49.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 64.0, 2.0]])
[64]: array([1.])

[65]: import os

# Get the current working directory
current_directory = os.getcwd()
print(f"Current working directory: {current_directory}")

Current working directory: E:\thyroid detection

[66]: import pickle

your_object = ...

# Set the file path for saving
file_path = 'E:\thyroid detection\enc.pickle'

try:
    # Save the object to the file
    with open(file_path, 'wb') as file:
        pickle.dump(your_object, file)
    print("Object saved successfully.")
except Exception as e:
    print(f"An error occurred: {e}")

Object saved successfully.

[ ]:
```

```
[64]: array([1.])
```

```
[65]: import os

# Get the current working directory
current_directory = os.getcwd()
print(f"Current working directory: {current_directory}")
```

Current working directory: E:\thyroid detection

```
[66]: import pickle

your_object = ...

# Set the file path for saving
file_path = 'E:/thyroid detection/enc.pickle'
```

```
try:
    # Save the object to the file
    with open(file_path, 'wb') as file:
        pickle.dump(your_object, file)
    print("Object saved successfully.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Object saved successfully.