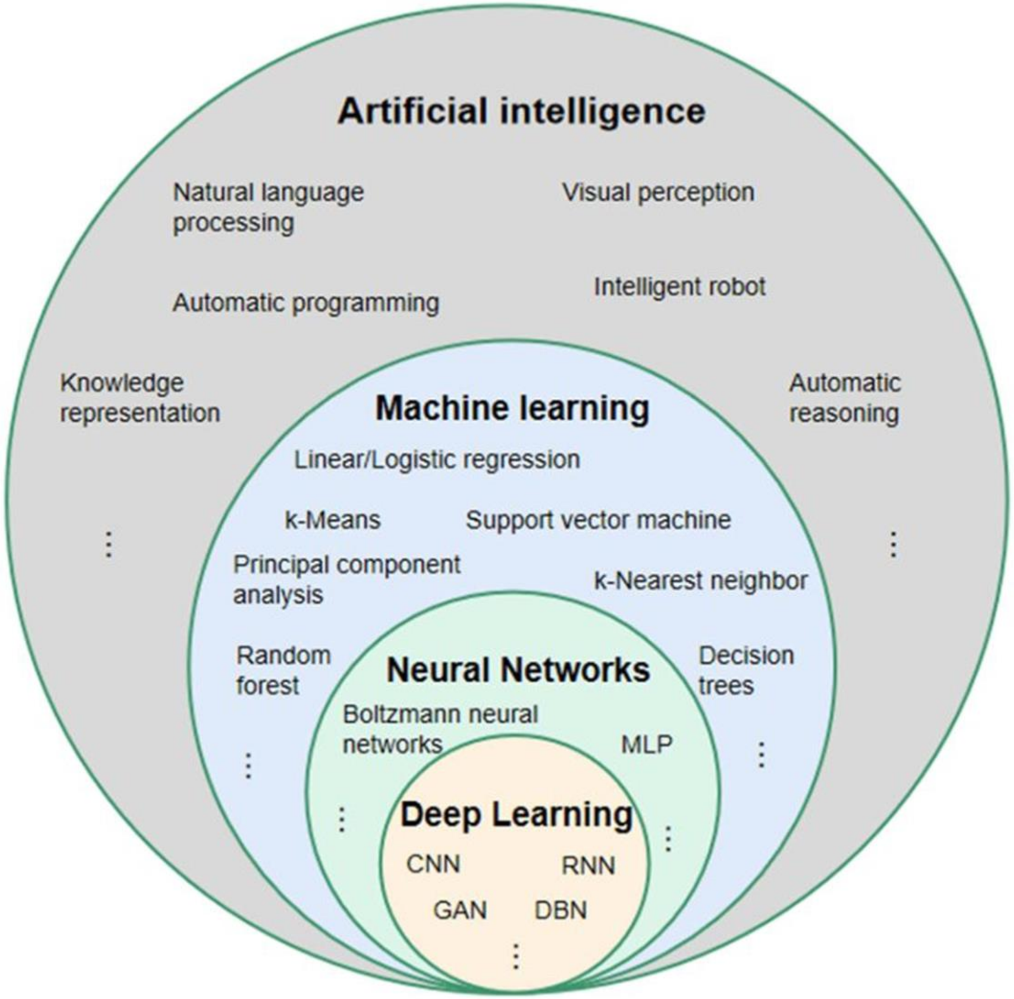
Decision tree

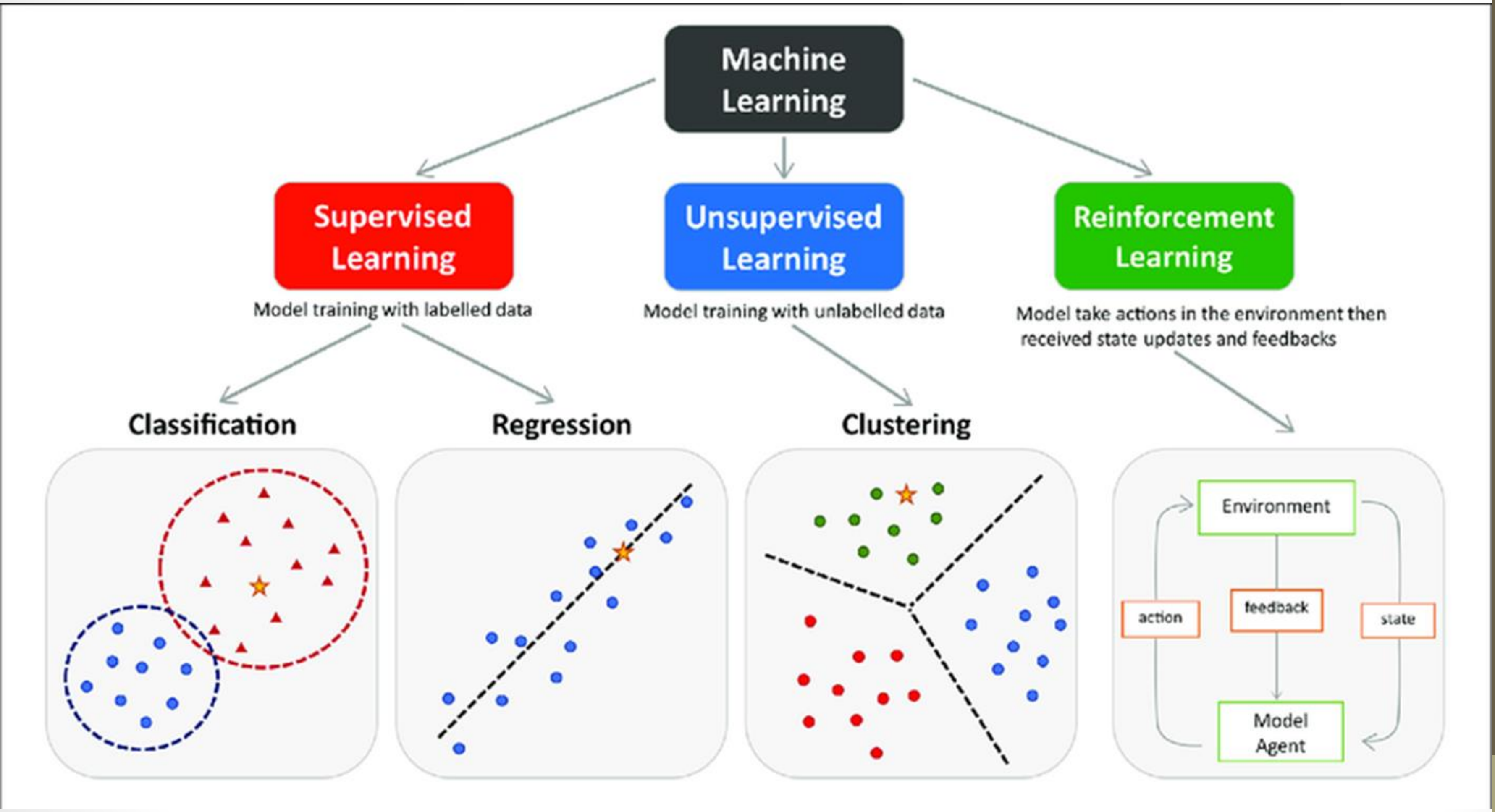
Kiana abdollahzadeh

Setayesh norouzi

What is ML?

- Machine learning (ML) is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy. ML develops algorithms by learning the hidden patterns of the datasets used it to make predictions on new similar type data, without being explicitly programmed for each task


Artificial intelligence
Natural language processing
Visual perception
Automatic programming
Intelligent robot
Knowledge representation
Automatic reasoning
⋮
⋮
Machine learning
Linear/Logistic regression
k-Means
Support vector machine
Principal component analysis
k-Nearest neighbor
Random forest
Decision trees
⋮
⋮
Neural Networks
Boltzmann neural networks
MLP
⋮
⋮
Deep Learning
CNN
RNN
GAN
DBN
⋮


Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Model training with labelled data
Model training with unlabelled data
Model take actions in the environment then received state updates and feedbacks
Classification
Regression
Clustering
Environment
action
feedback
state
Model Agent

Supervised learning

- Supervised learning is a type of machine learning algorithm that learns from labeled data. Labeled data is data that has been tagged with a correct answer or classification.

Key Points:

- Supervised learning involves training a machine from labeled data.
- Labeled data consists of examples with the correct answer or classification.
- The machine learns the relationship between inputs (fruit images) and outputs (fruit labels).
- The trained machine can then make predictions on new, unlabeled data.

Types of Supervised Learning

- Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.
- Classification: A classification problem is when the output variable is a category, such as “Red” or “blue” , “disease” or “no disease”.

Classification

- Classification is a process of categorizing data or objects into predefined classes or categories based on their features or attributes.

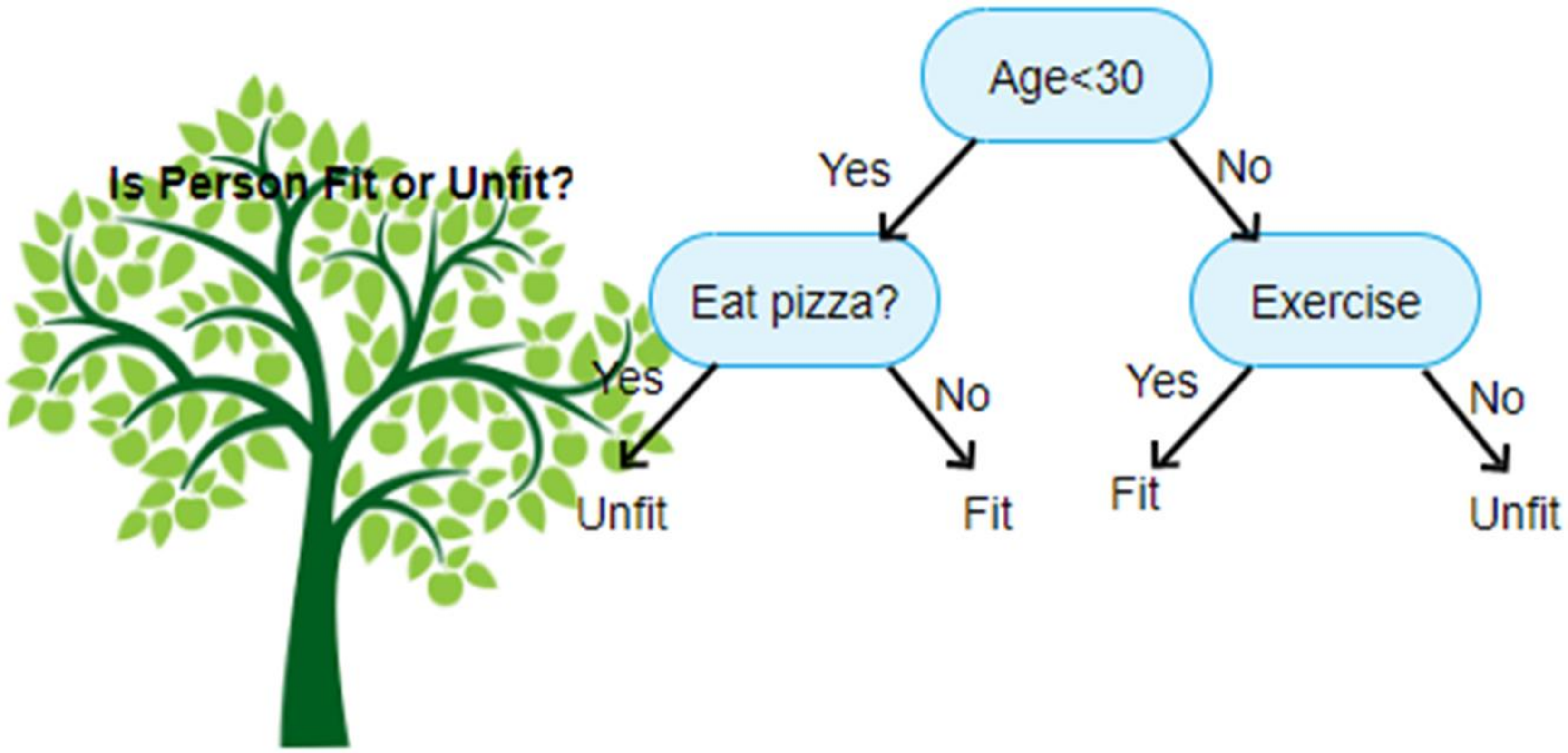
The main objective of classification machine learning is to build a model that can accurately assign a label or category to a new observation based on its features.

Classification Algorithms

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine (SVM)
- Naive Bayes
- K-Nearest Neighbors (KNN)

Decision Tree

A tree shaped diagram used to determine a course of action.
Each branch of the tree represents a possible decision, occurrence or reaction.



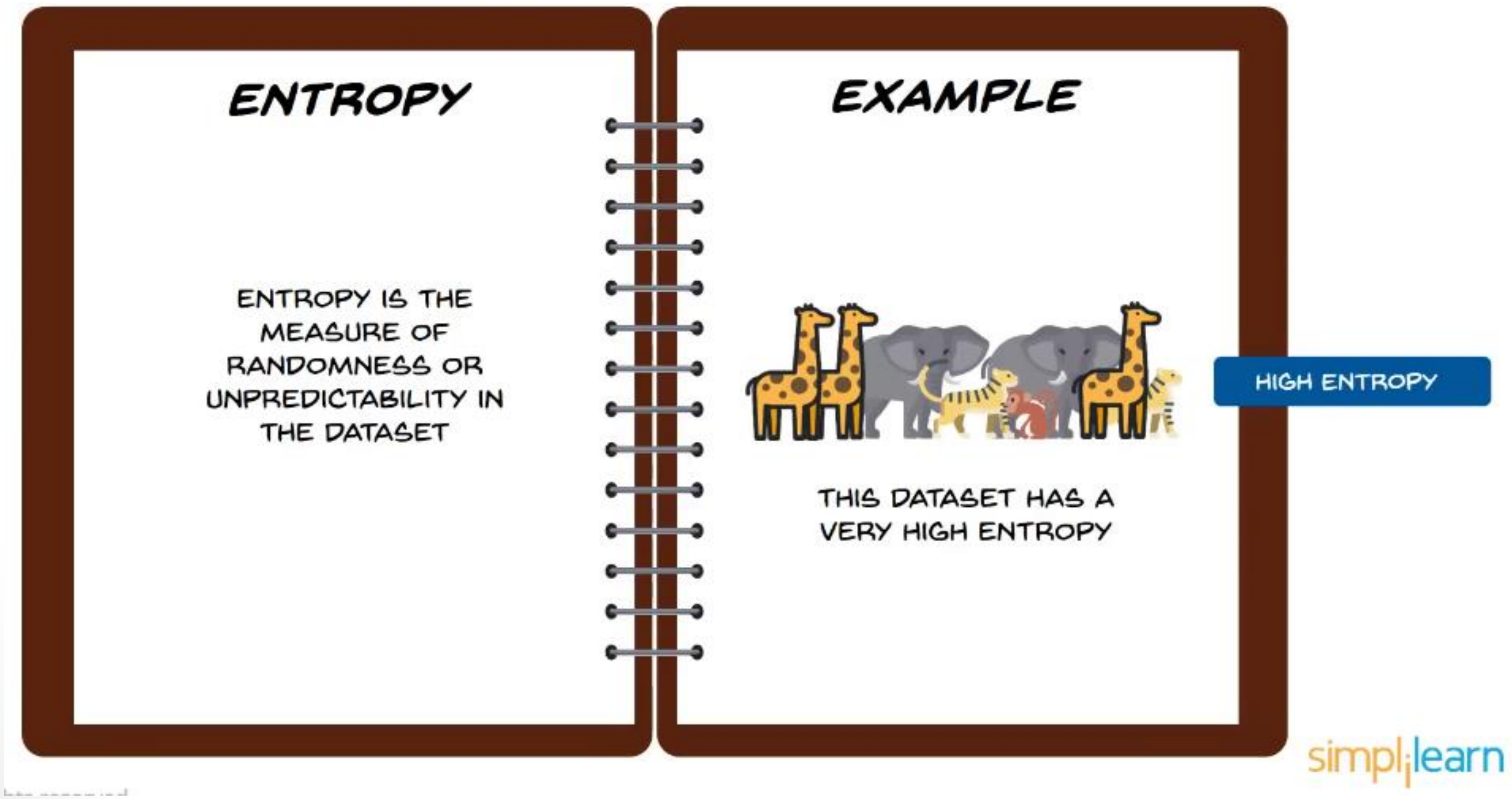
- Advantages: It’s simple to visualize

- Disadvantage :overfitting/low bias

TERMS:

- Entropy: Is a measure of randomness or unpredictability in the dataset
- Information gain: It lowers the entropy after the data is split
- Leaf node: the classification(the answer)
- Decision node: breaks the data into two parts
- Root node: top node

Important terms



ENTROPY

ENTROPY IS THE MEASURE OF RANDOMNESS OR UNPREDICTABILITY IN THE DATASET

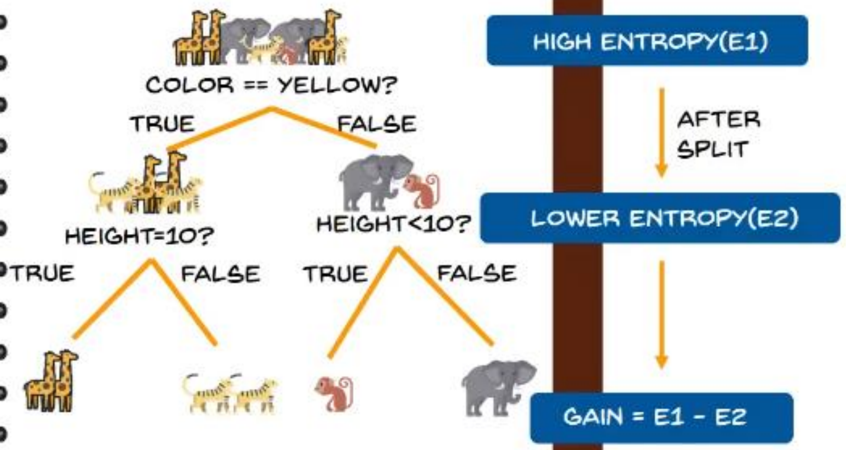
EXAMPLE

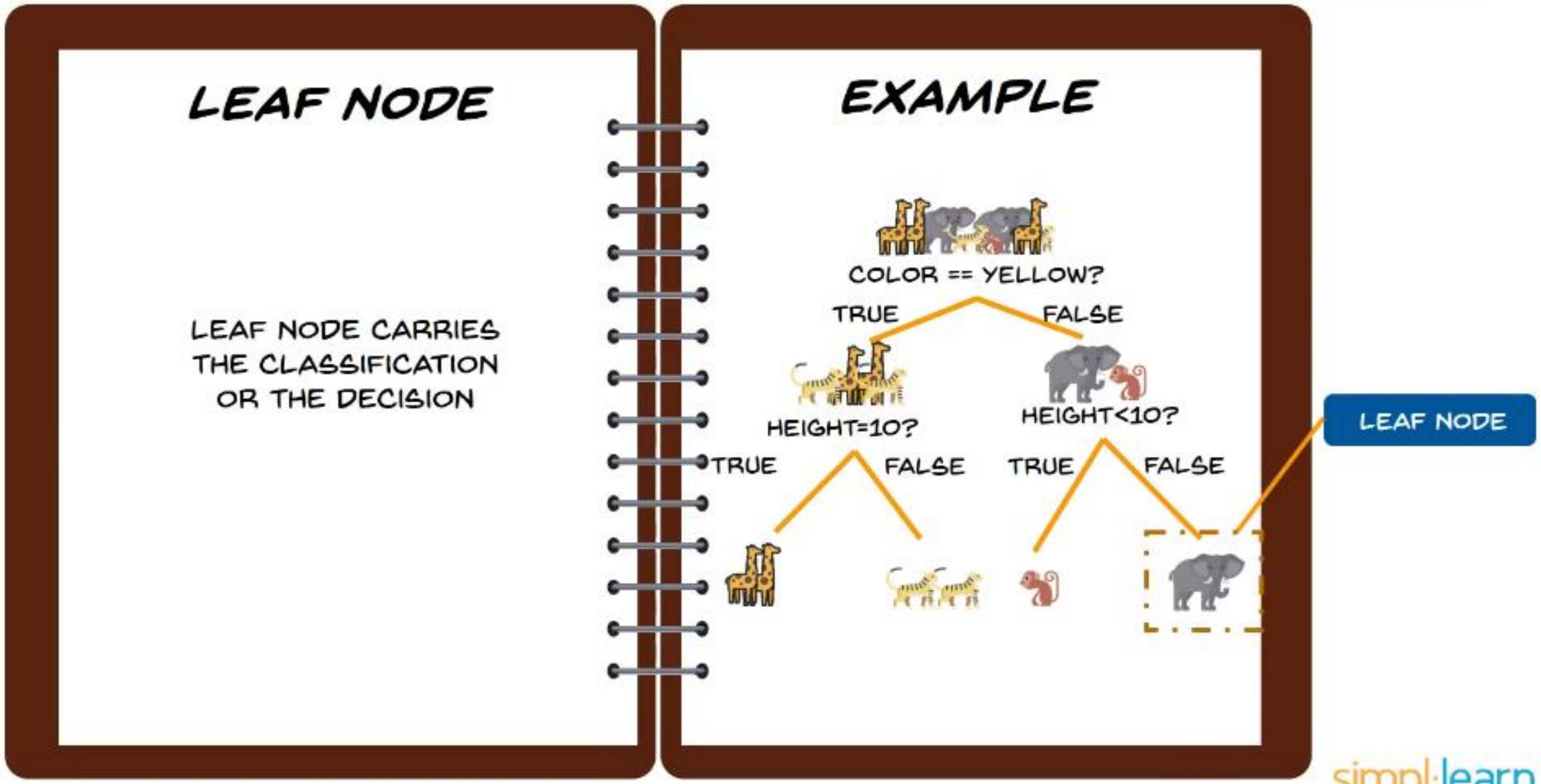
THIS DATASET HAS A VERY HIGH ENTROPY

INFORMATION GAIN

IT IS THE MEASURE OF DECREASE IN ENTROPY AFTER THE DATASET IS SPLIT

EXAMPLE





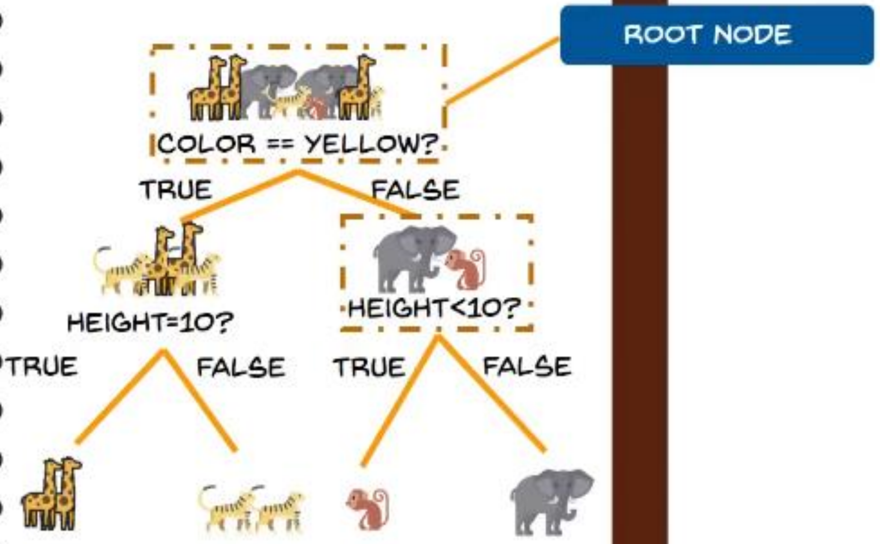
LEAF NODE

LEAF NODE CARRIES THE CLASSIFICATION OR THE DECISION

ROOT NODE

THE TOP MOST DECISION NODE IS KNOWN AS THE ROOT NODE

EXAMPLE



How does the decision tree work?

PROBLEM STATEMENT

TO CLASSIFY THE DIFFERENT TYPES OF ANIMALS BASED ON THEIR FEATURES USING DECISION TREE

HOW TO SPLIT THE DATA

WE HAVE TO FRAME THE CONDITIONS THAT SPLIT THE DATA IN SUCH A WAY THAT THE INFORMATION GAIN IS THE HIGHEST

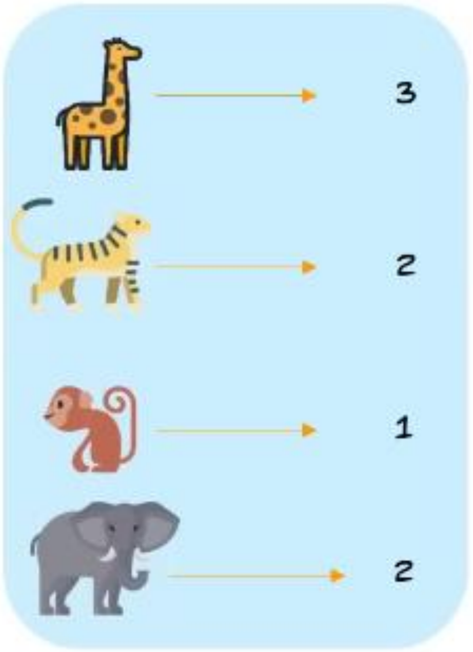
THE DATASET IS LOOKING QUITE MESSY AND THE ENTROPY IS HIGH IN THIS CASE

TRAINING DATASET

COLOR	HEIGHT	LABEL
GREY	10	ELEPHANT
YELLOW	10	GIRAFFE
BROWN	3	MONKEY
GREY	10	ELEPHANT
YELLOW	4	TIGER

FORMULA FOR ENTROPY

$$\sum_{i=1}^{k} P(valuei).log_2(P(value_i)$$



TOTAL 8

LET'S USE THE FORMULA

$$\sum_{i=1}^{k} P(valuei).log_2(P(value_i)$$

ENTROPY = $(\frac{3}{8})\, log_2(\frac{3}{8}) + (\frac{2}{8})\, log_2(\frac{2}{8}) + (\frac{1}{8})\, log_2(\frac{1}{8}) + (\frac{2}{8})\, log_2(\frac{2}{8})$

ENTROPY=0.571

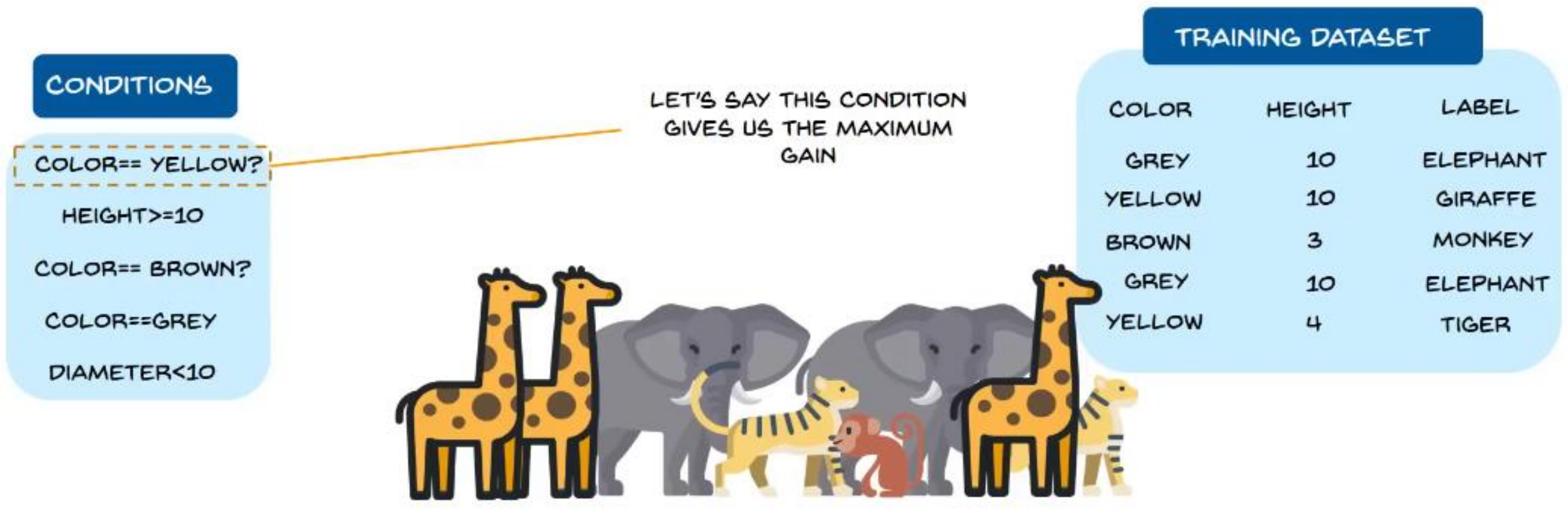
WE WILL CALCULATE THE ENTROPY OF THE DATASET SIMILARLY AFTER EVERY SPLIT TO CALCULATE THE GAIN

GAIN CAN BE CALCULATED BY FINDING THE DIFFERENCE OF THE SUBSEQUENT ENTROPY VALUES AFTER SPLIT

NOW WE WILL TRY TO CHOOSE A CONDITION THAT GIVES US THE HIGHEST GAIN

WE WILL DO THAT BY SPLITTING THE DATA USING EACH CONDITION AND CHECKING THE GAIN THAT WE GET OUT THEM.

THE CONDITION THAT GIVES US THE HIGHEST GAIN WILL BE USED TO MAKE THE FIRST SPLIT



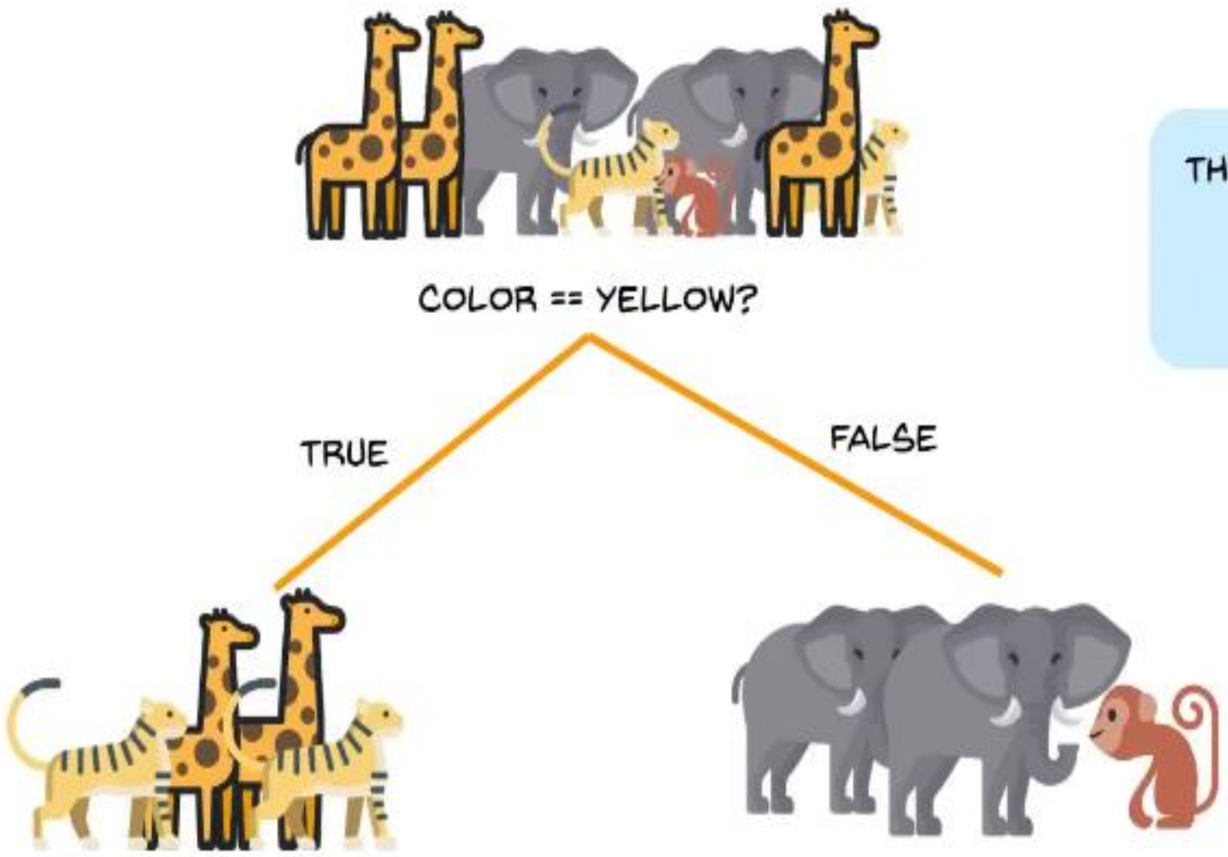
CONDITIONS

- COLOR== YELLOW?
- HEIGHT>=10
- COLOR== BROWN?
- COLOR==GREY
- DIAMETER<10

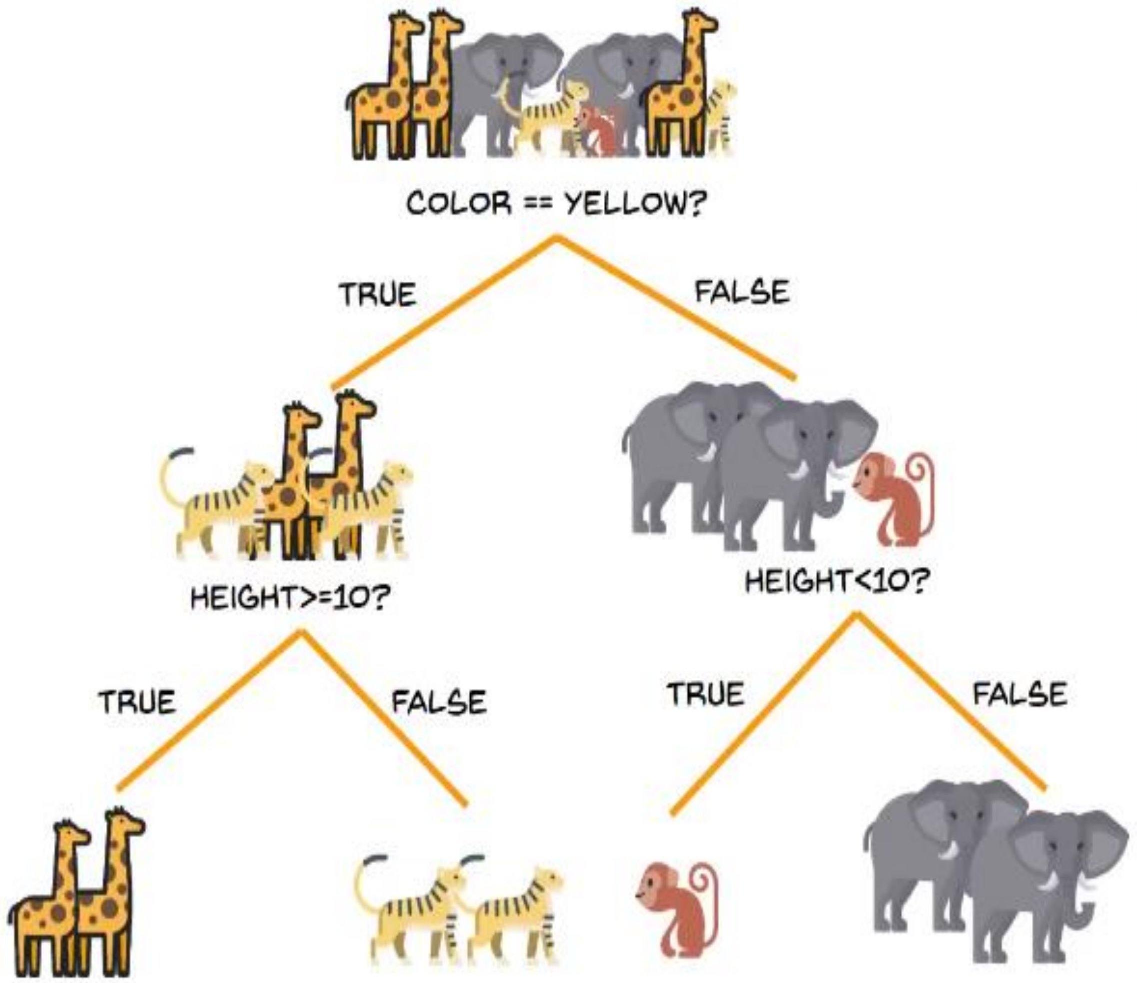
LET'S SAY THIS CONDITION GIVES US THE MAXIMUM GAIN

TRAINING DATASET

COLOR	HEIGHT	LABEL
GREY	10	ELEPHANT
YELLOW	10	GIRAFFE
BROWN	3	MONKEY
GREY	10	ELEPHANT
YELLOW	4	TIGER



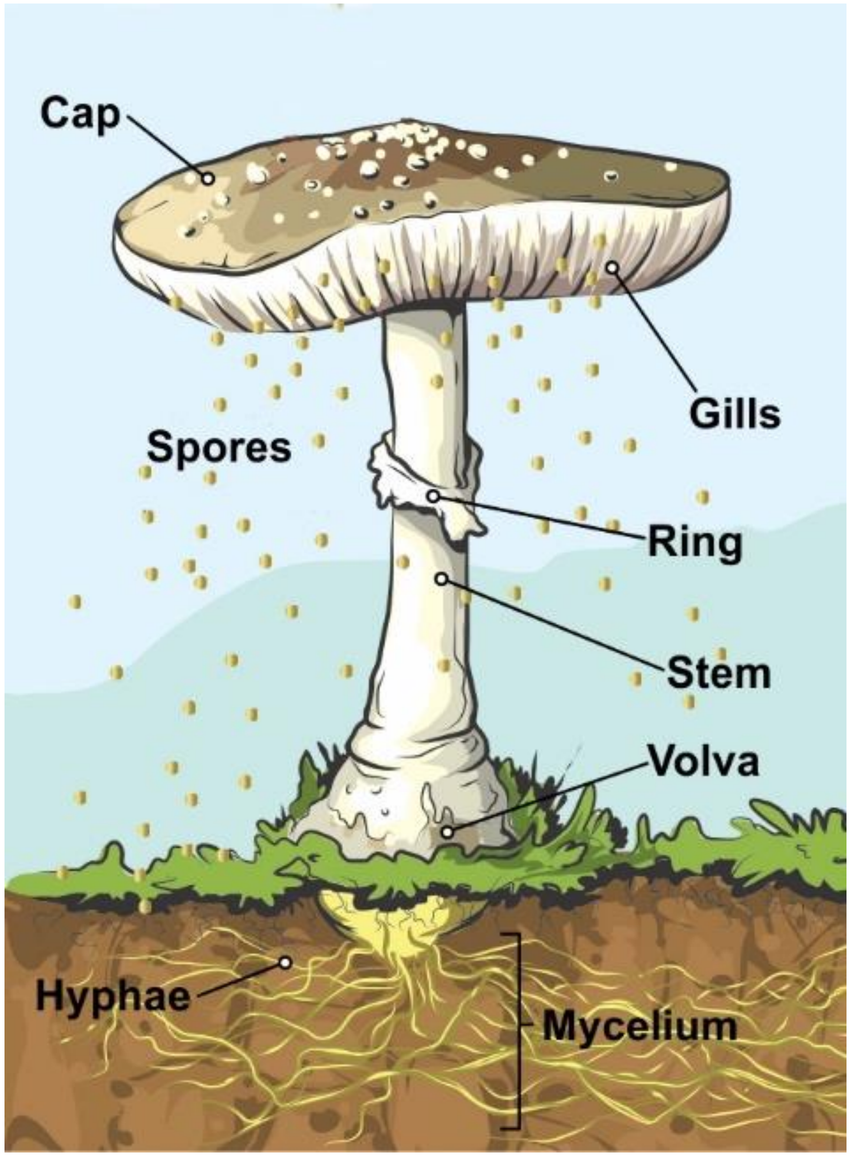
THE ENTROPY AFTER SPLITTING HAS DECREASED CONSIDERABLY



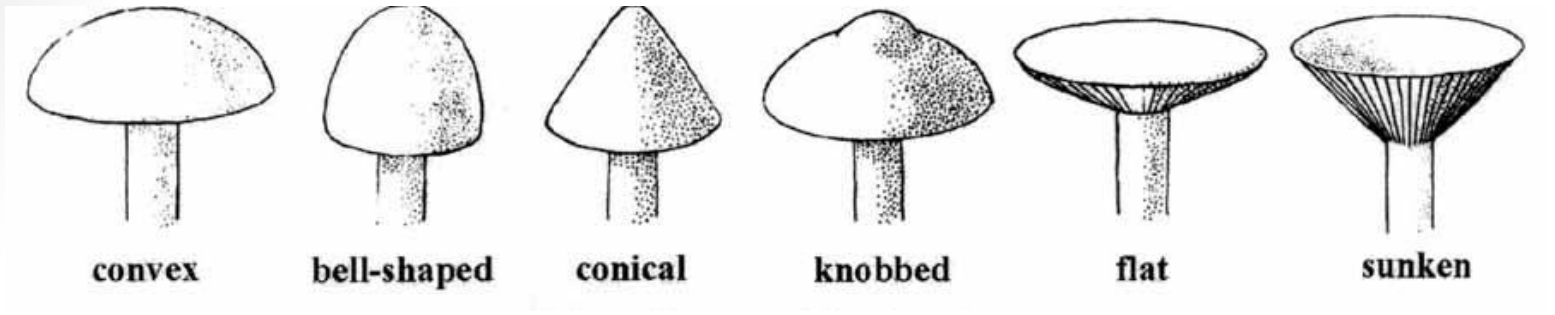
SINCE EVERY BRANCH NOW CONTAINS SINGLE LABEL TYPE, WE CAN SAY THAT THE ENTROPY IN THIS CASE HAS REACHED THE LEAST VALUE

Mushrooms: can I eat it?

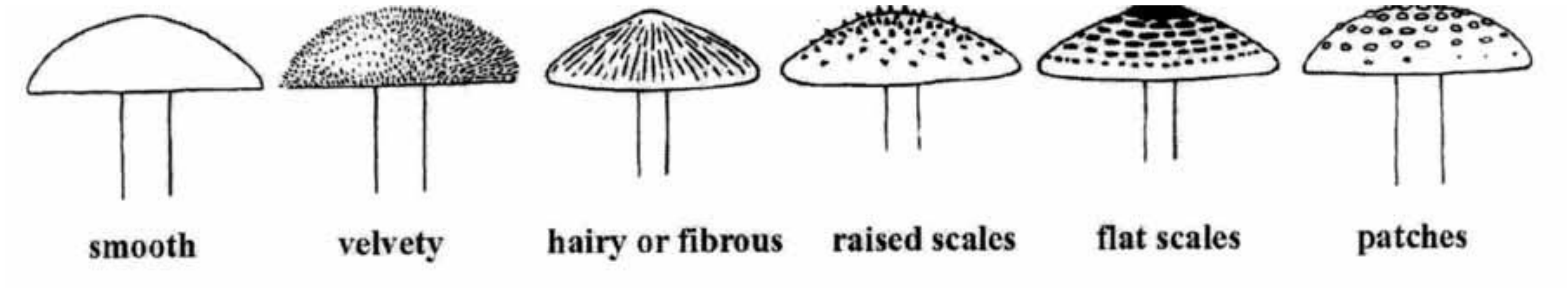
Mushroom parts



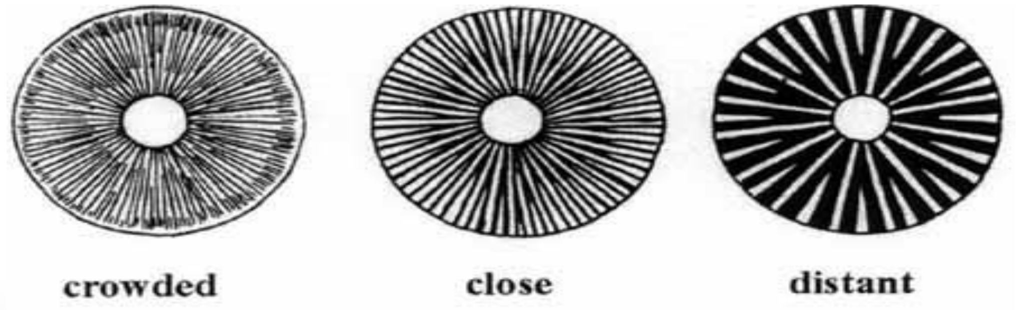
Mushroom cap shape:

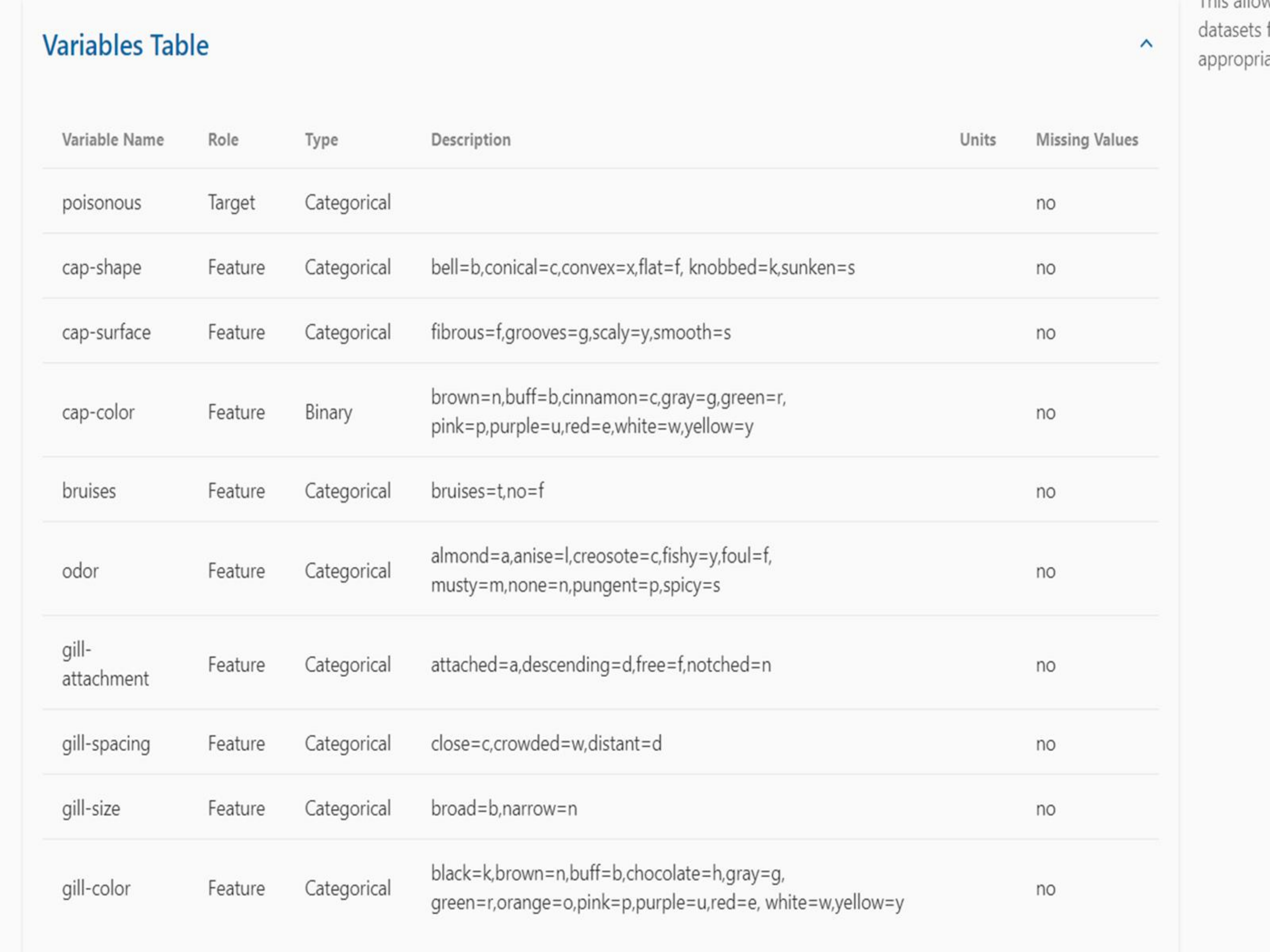


Mushroom cap surface:



Mushroom gill spacing:



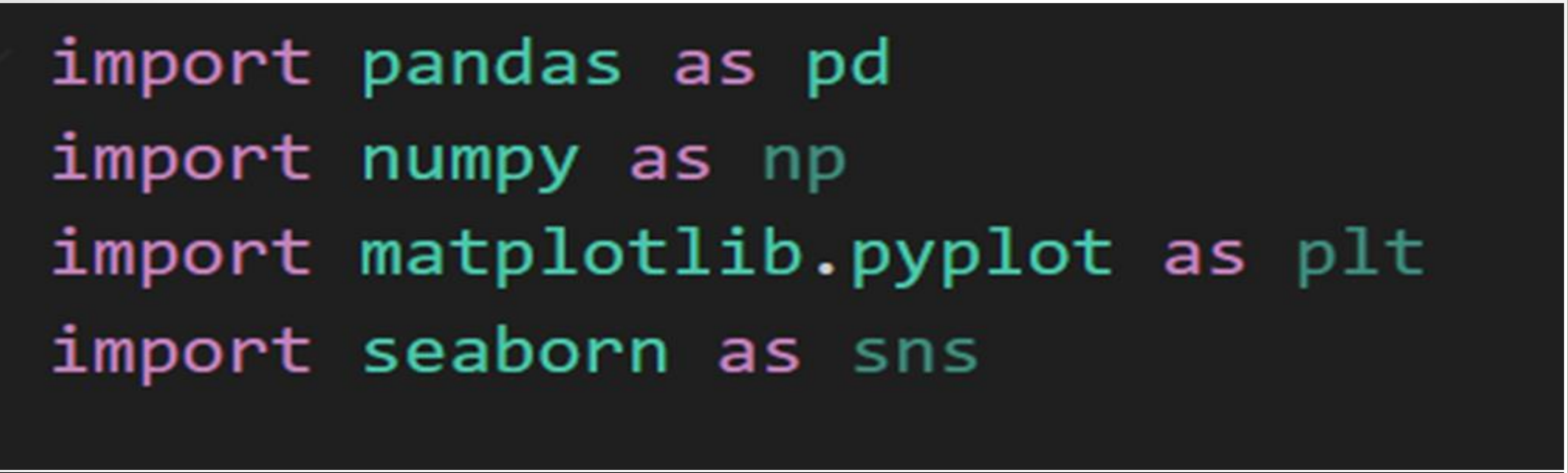


Variables Table

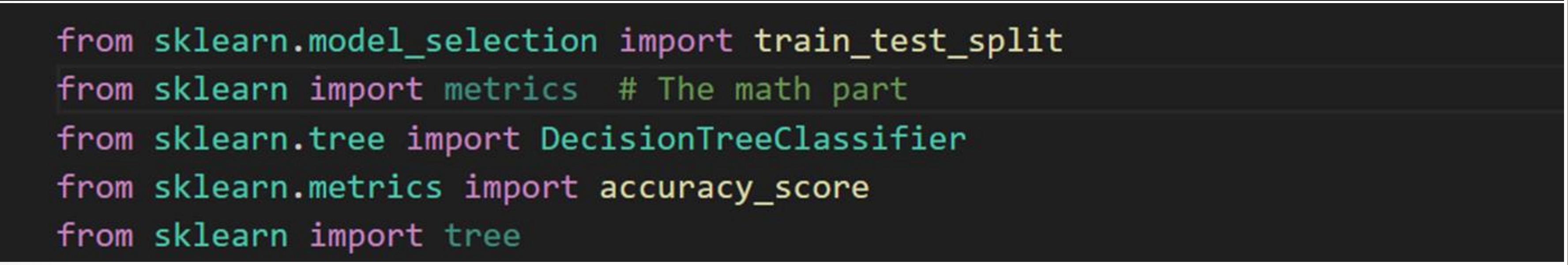
Variable Name	Role	Type	Description	Units	Missing Values
poisonous	Target	Categorical			no
cap-shape	Feature	Categorical	bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s		no
cap-surface	Feature	Categorical	fibrous=f,grooves=g,scaly=y,smooth=s		no
cap-color	Feature	Binary	brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y		no
bruises	Feature	Categorical	bruises=t,no=f		no
odor	Feature	Categorical	almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s		no
gill-attachment	Feature	Categorical	attached=a,descending=d,free=f,notched=n		no
gill-spacing	Feature	Categorical	close=c,crowded=w,distant=d		no
gill-size	Feature	Categorical	broad=b,narrow=n		no
gill-color	Feature	Categorical	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y		no

datasets f
appropria

Importing libraries

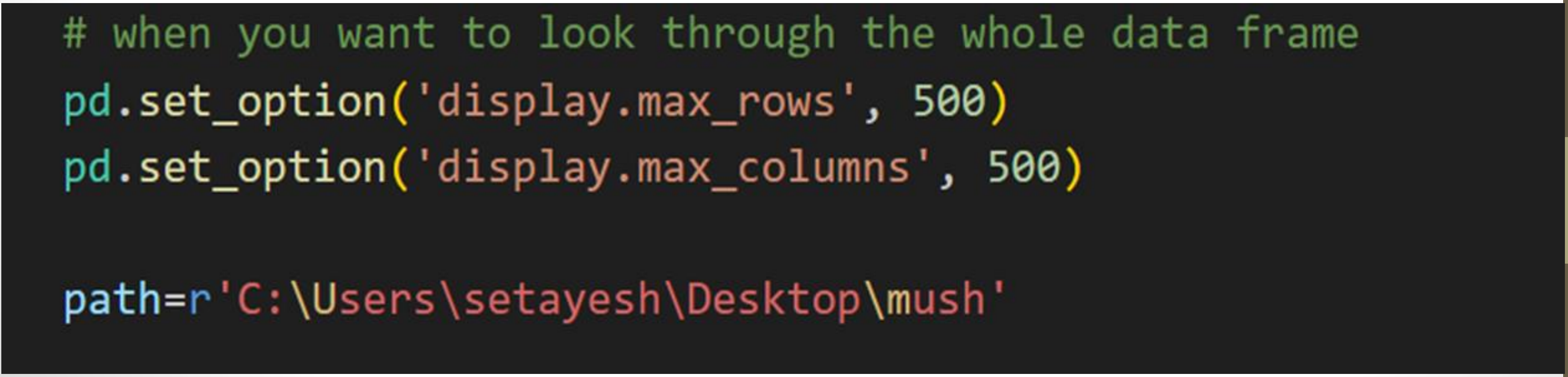


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```



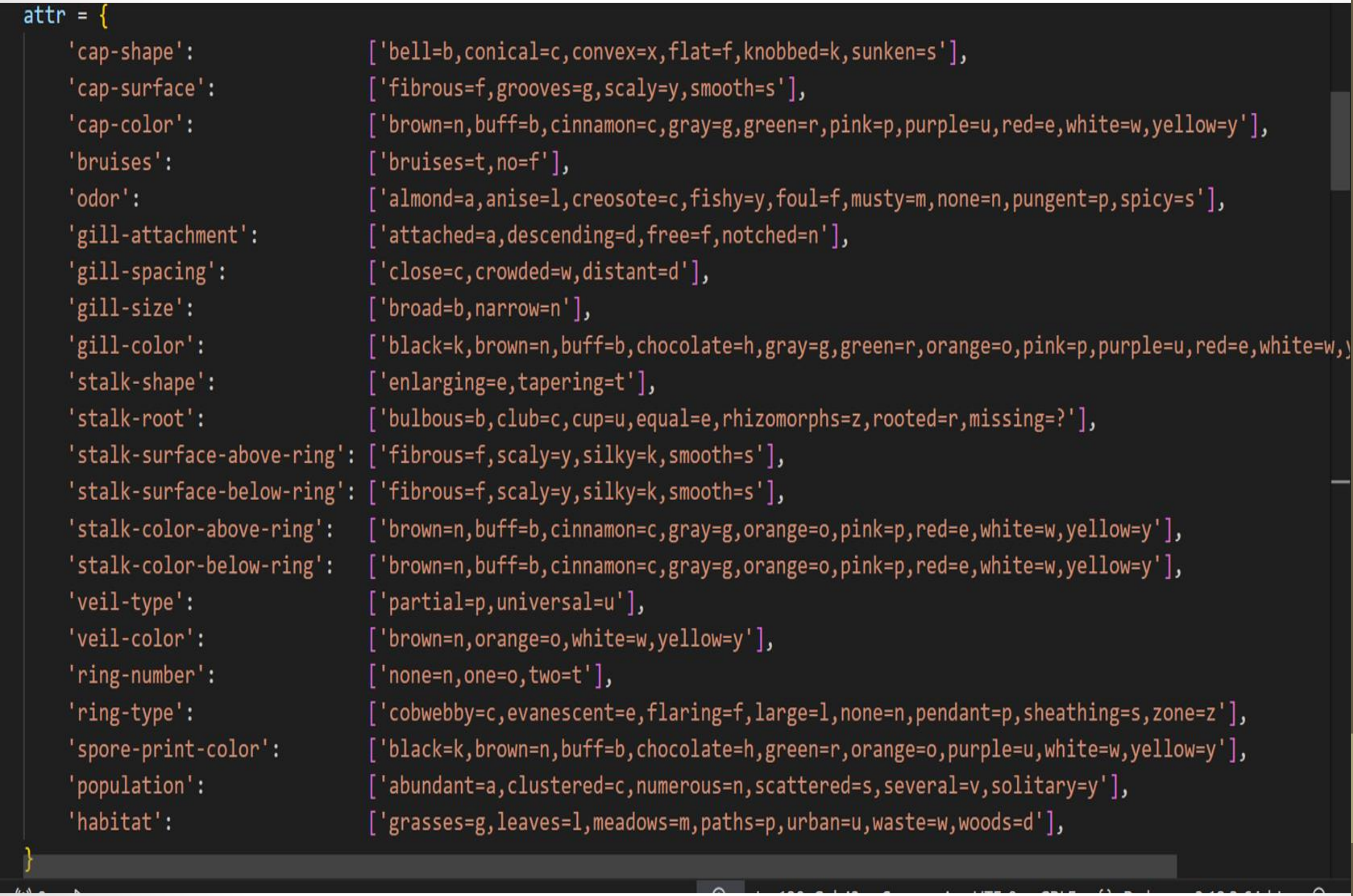
```
from sklearn.model_selection import train_test_split
from sklearn import metrics  # The math part
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

Showing the table 500*500



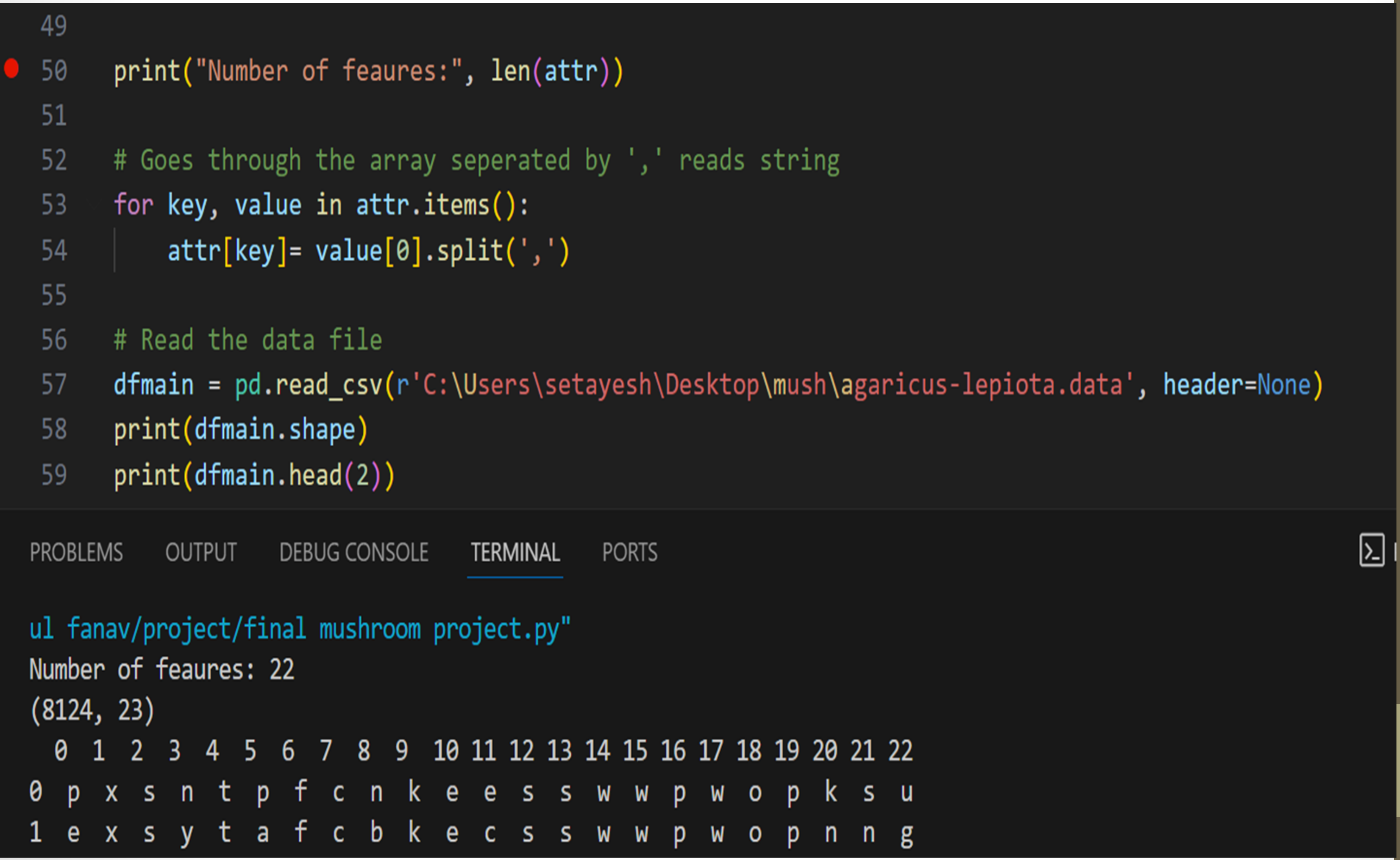
```
# when you want to look through the whole data frame
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)

path=r'C:\Users\setayesh\Desktop\mush'
```

Making a dictionary in python,including feature titles and their attributes



```
attr = {
    'cap-shape':                ['bell=b,conical=c,convex=x,flat=f,knobbed=k,sunken=s'],
    'cap-surface':              ['fibrous=f,grooves=g,scaly=y,smooth=s'],
    'cap-color':                ['brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y'],
    'bruises':                  ['bruises=t,no=f'],
    'odor':                     ['almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s'],
    'gill-attachment':          ['attached=a,descending=d,free=f,notched=n'],
    'gill-spacing':             ['close=c,crowded=w,distant=d'],
    'gill-size':                ['broad=b,narrow=n'],
    'gill-color':               ['black=k,brown=n,buff=b,chocolate=h,gray=g,green=r,orange=o,pink=p,purple=u,red=e,white=w,y
    'stalk-shape':              ['enlarging=e,tapering=t'],
    'stalk-root':               ['bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?'],
    'stalk-surface-above-ring': ['fibrous=f,scaly=y,silky=k,smooth=s'],
    'stalk-surface-below-ring': ['fibrous=f,scaly=y,silky=k,smooth=s'],
    'stalk-color-above-ring':   ['brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y'],
    'stalk-color-below-ring':   ['brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y'],
    'veil-type':                ['partial=p,universal=u'],
    'veil-color':               ['brown=n,orange=o,white=w,yellow=y'],
    'ring-number':              ['none=n,one=o,two=t'],
    'ring-type':                ['cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z'],
    'spore-print-color':        ['black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y'],
    'population':               ['abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y'],
    'habitat':                  ['grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d'],
}
```

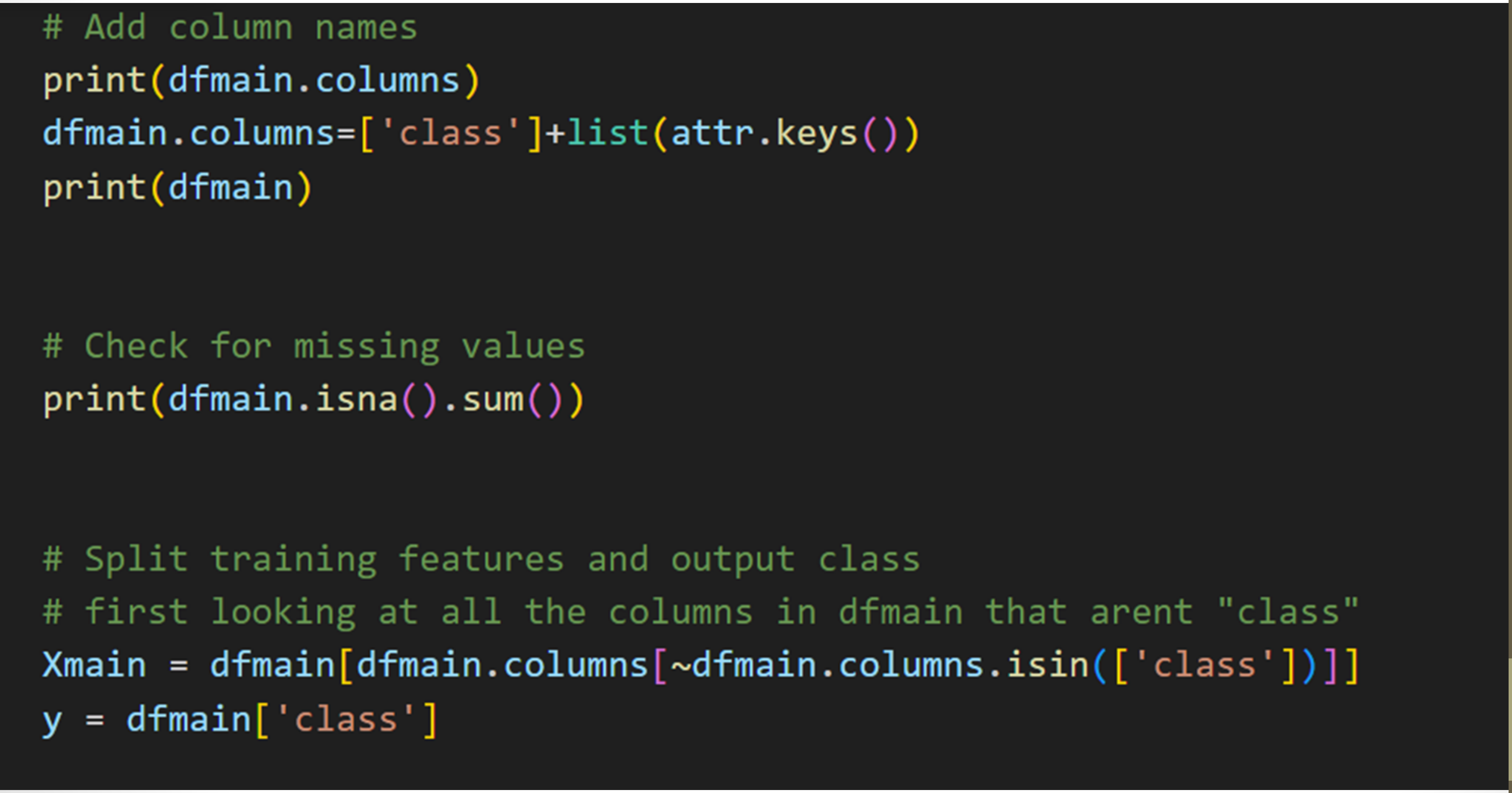
We have 8124 records (rows) and 23 columns (features)
Now we print the first two rows of our data: the columns are 1 to 22 (number of features)
With column 0being our target (edible or poisonous)



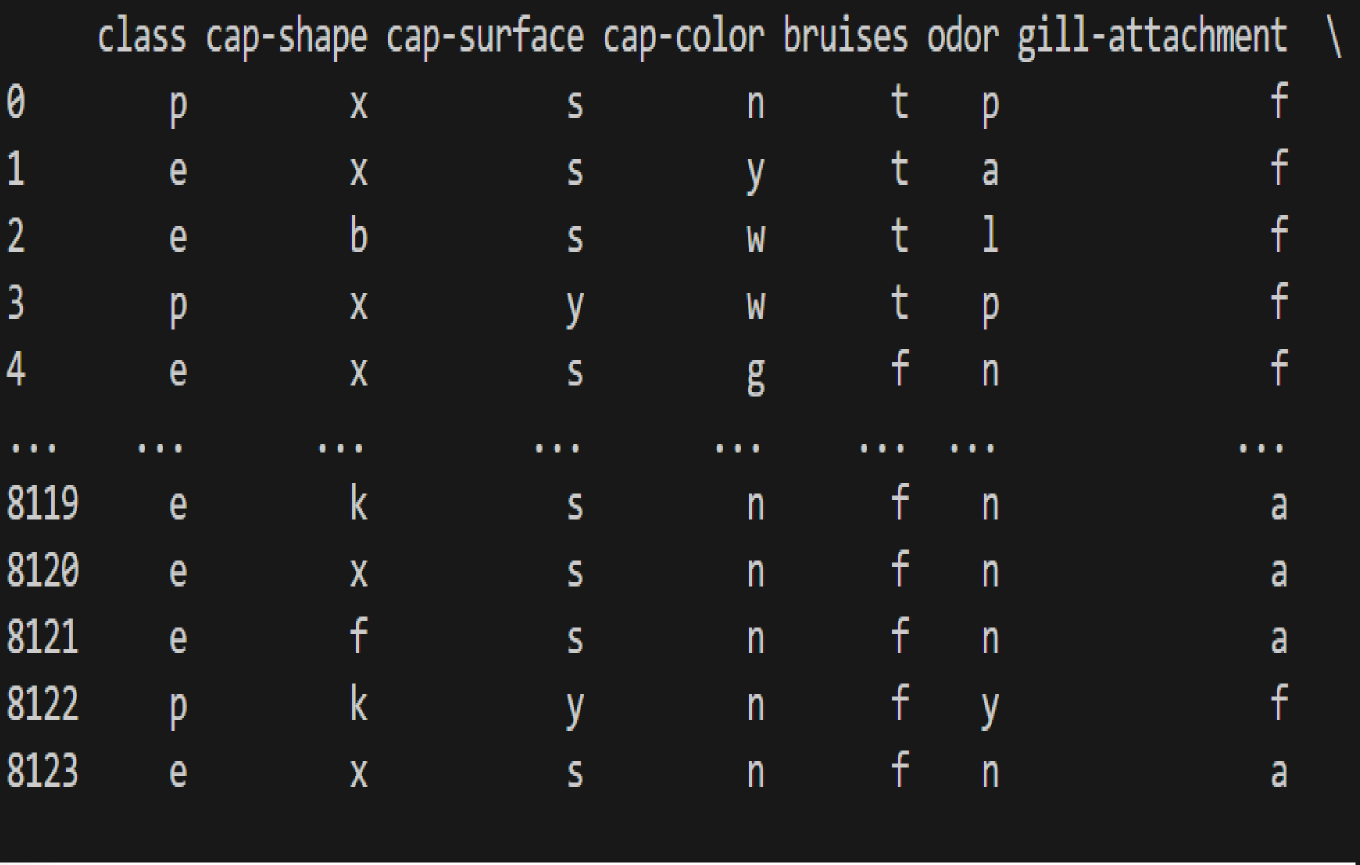
```python

print("Number of feaures:", len(attr))

# Goes through the array seperated by ',' reads string
for key, value in attr.items():
    attr[key]= value[0].split(',')

# Read the data file
dfmain = pd.read_csv(r'C:\Users\setayesh\Desktop\mush\agaricus-lepiota.data', header=None)
print(dfmain.shape)
print(dfmain.head(2))
```

```
ul fanav/project/final mushroom project.py"
Number of feaures: 22
(8124, 23)
  0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22
0 p  x  s  n  t  p  f  c  n  k  e  e  s  s  w  w  p  w  o  p  k  s  u
1 e  x  s  y  t  a  f  c  b  k  e  c  s  s  w  w  p  w  o  p  n  n  g
```

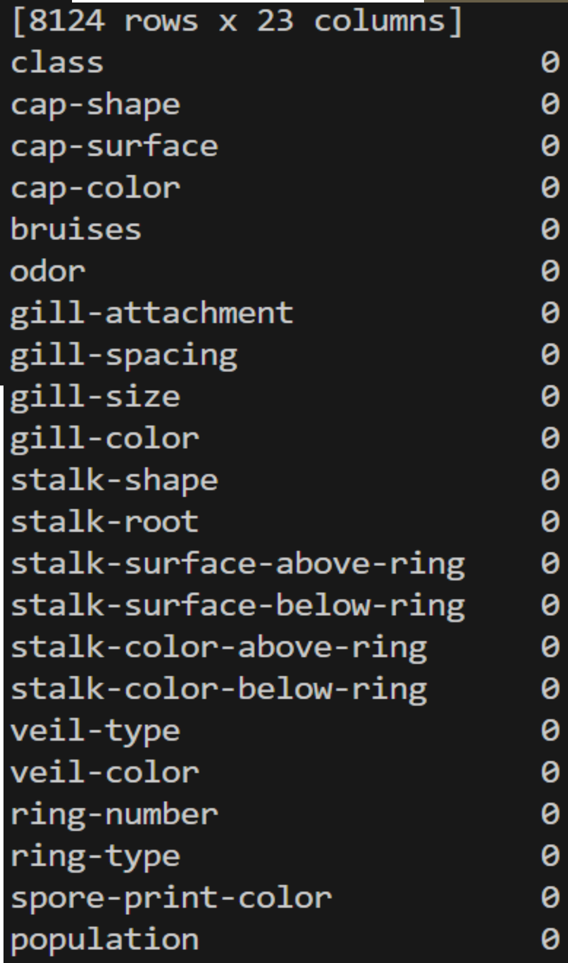
We first add names to our columns using the dictionary we made before.
Then we check for missing values number of null columns in our data which is none
We establish xmain as our features and y as our target class



```
# Add column names
print(dfmain.columns)
dfmain.columns=['class']+list(attr.keys())
print(dfmain)


# Check for missing values
print(dfmain.isna().sum())


# Split training features and output class
# first looking at all the columns in dfmain that arent "class"
Xmain = dfmain[dfmain.columns[~dfmain.columns.isin(['class'])]]
y = dfmain['class']
```



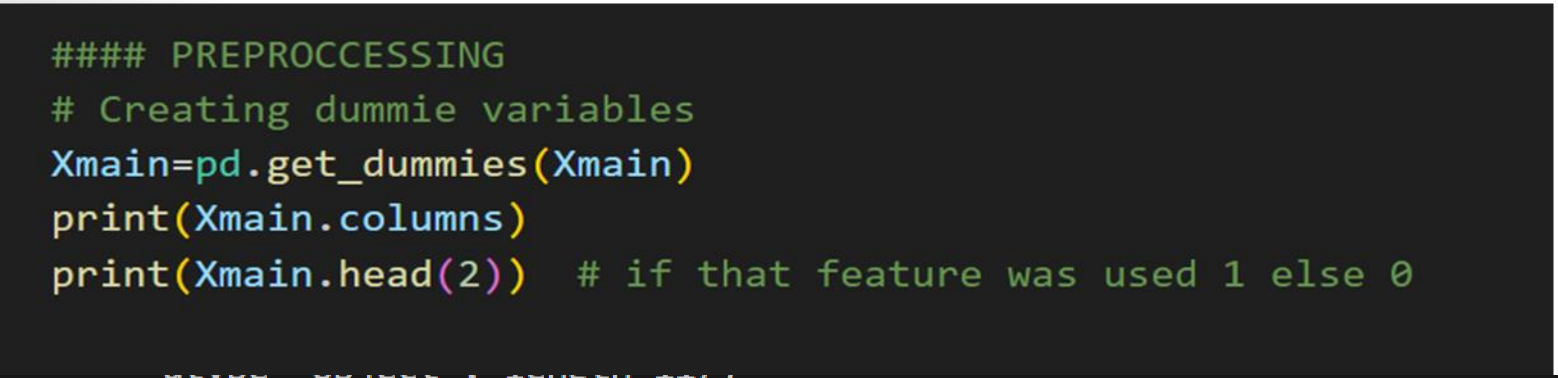
```
     class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0        p         x           s         n       t    p               f
1        e         x           s         y       t    a               f
2        e         b           s         w       t    l               f
3        p         x           y         w       t    p               f
4        e         x           s         g       f    n               f
...    ...       ...         ...       ...     ...  ...             ...
8119     e         k           s         n       f    n               a
8120     e         x           s         n       f    n               a
8121     e         f           s         n       f    n               a
8122     p         k           y         n       f    y               f
8123     e         x           s         n       f    n               a
```



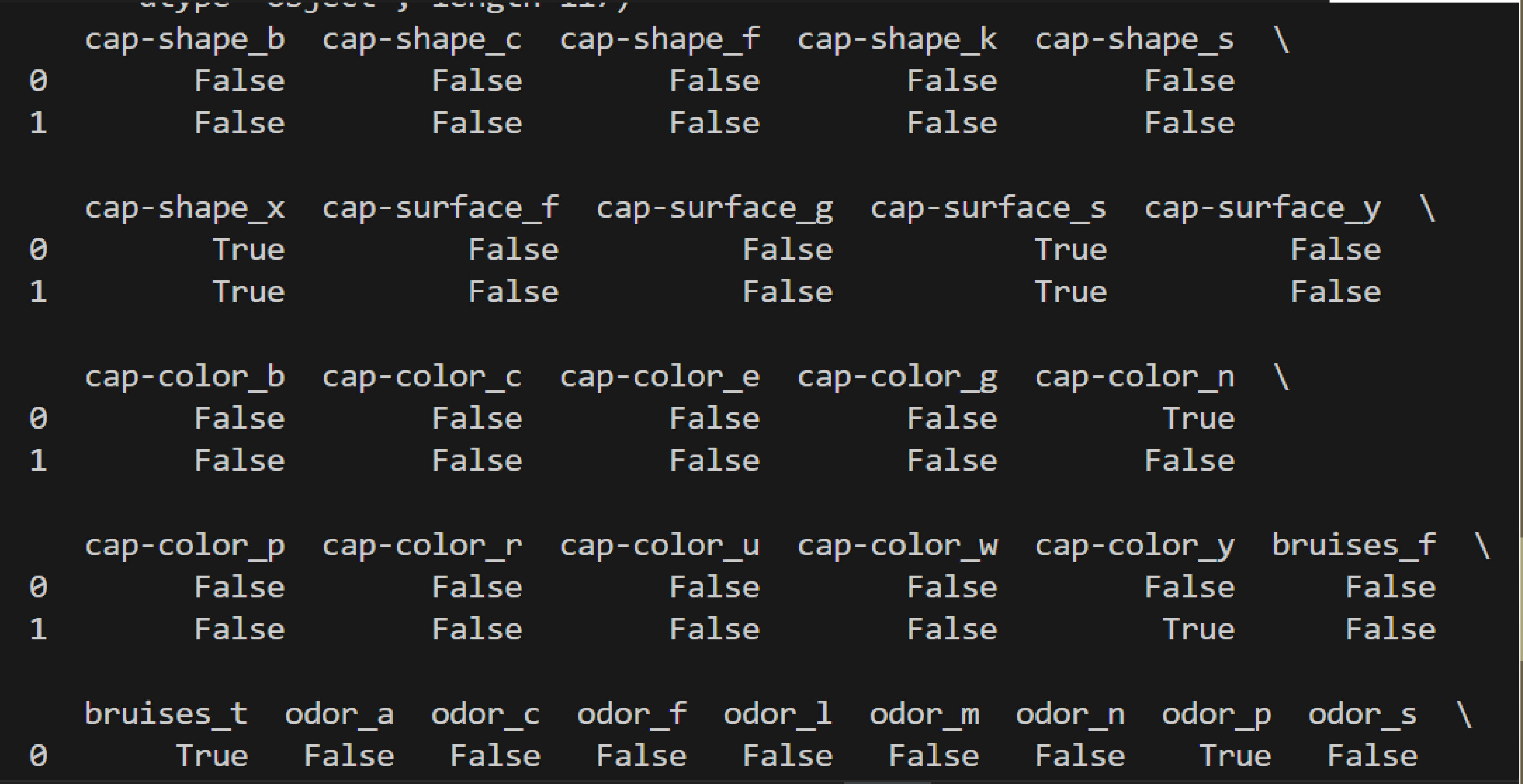
```
[8124 rows x 23 columns]
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
```

We then make dummie variables

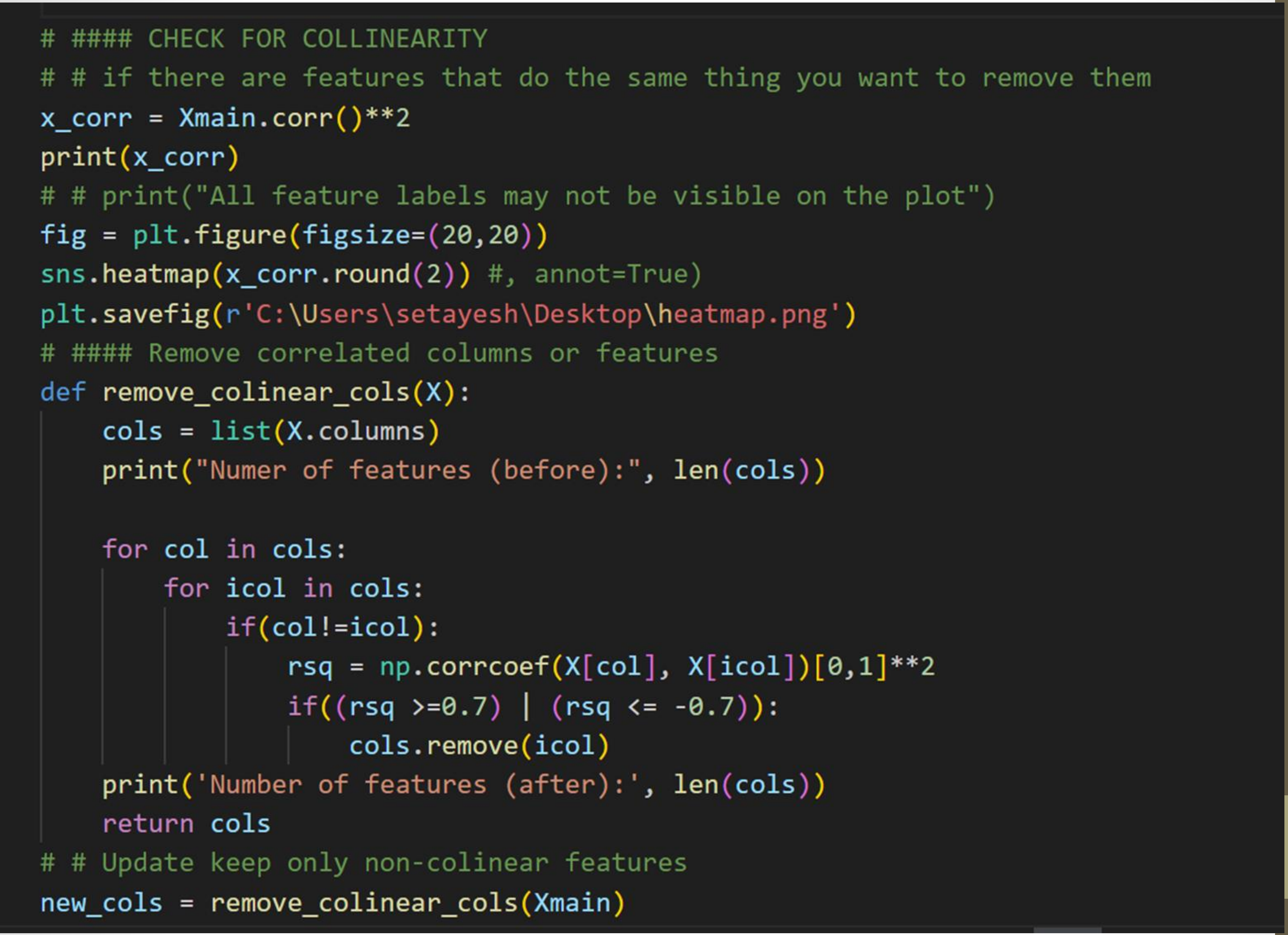
We basically transform our data into 0 and 1s



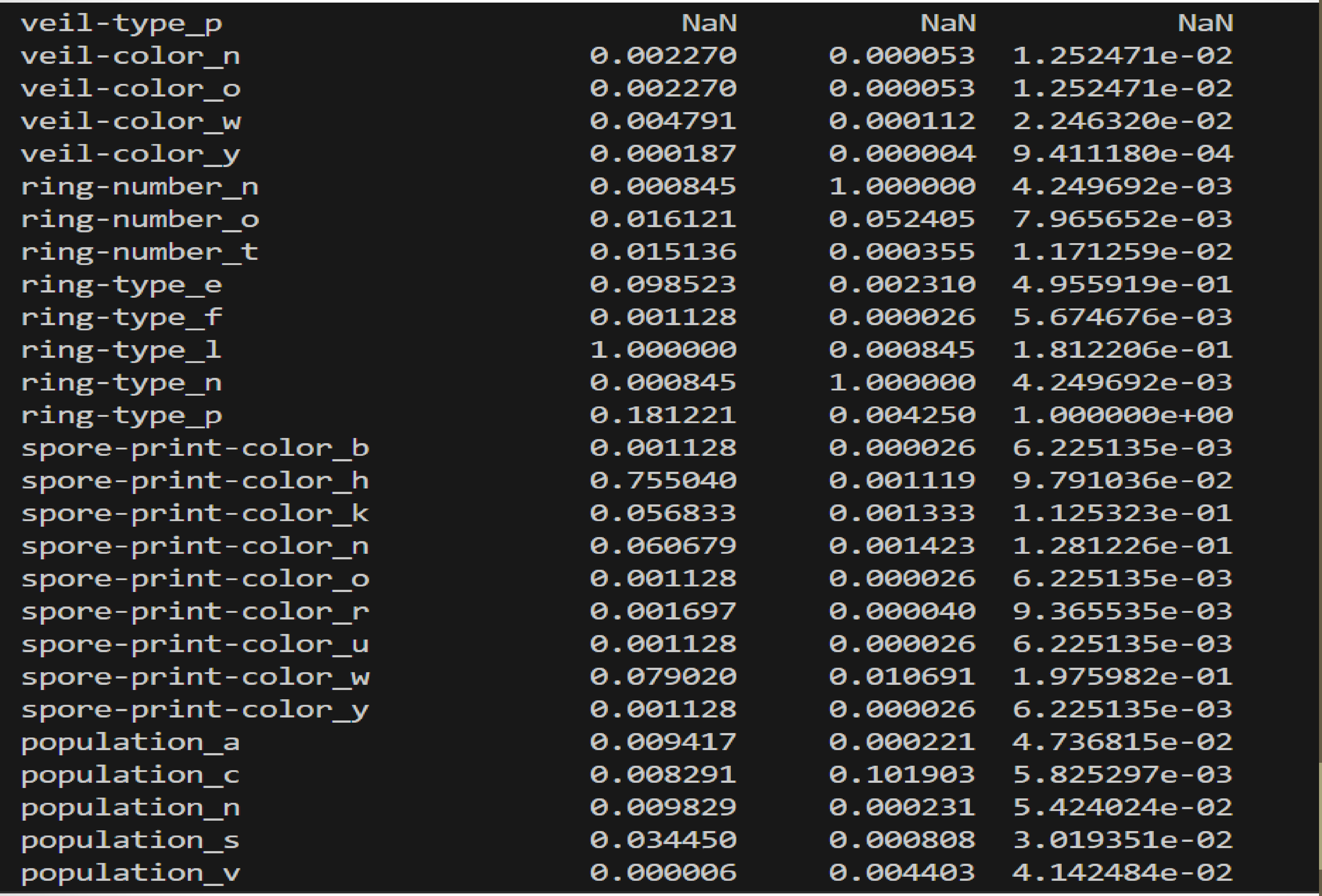
```
#### PREPROCCESSING
# Creating dummie variables
Xmain=pd.get_dummies(Xmain)
print(Xmain.columns)
print(Xmain.head(2))  # if that feature was used 1 else 0
```



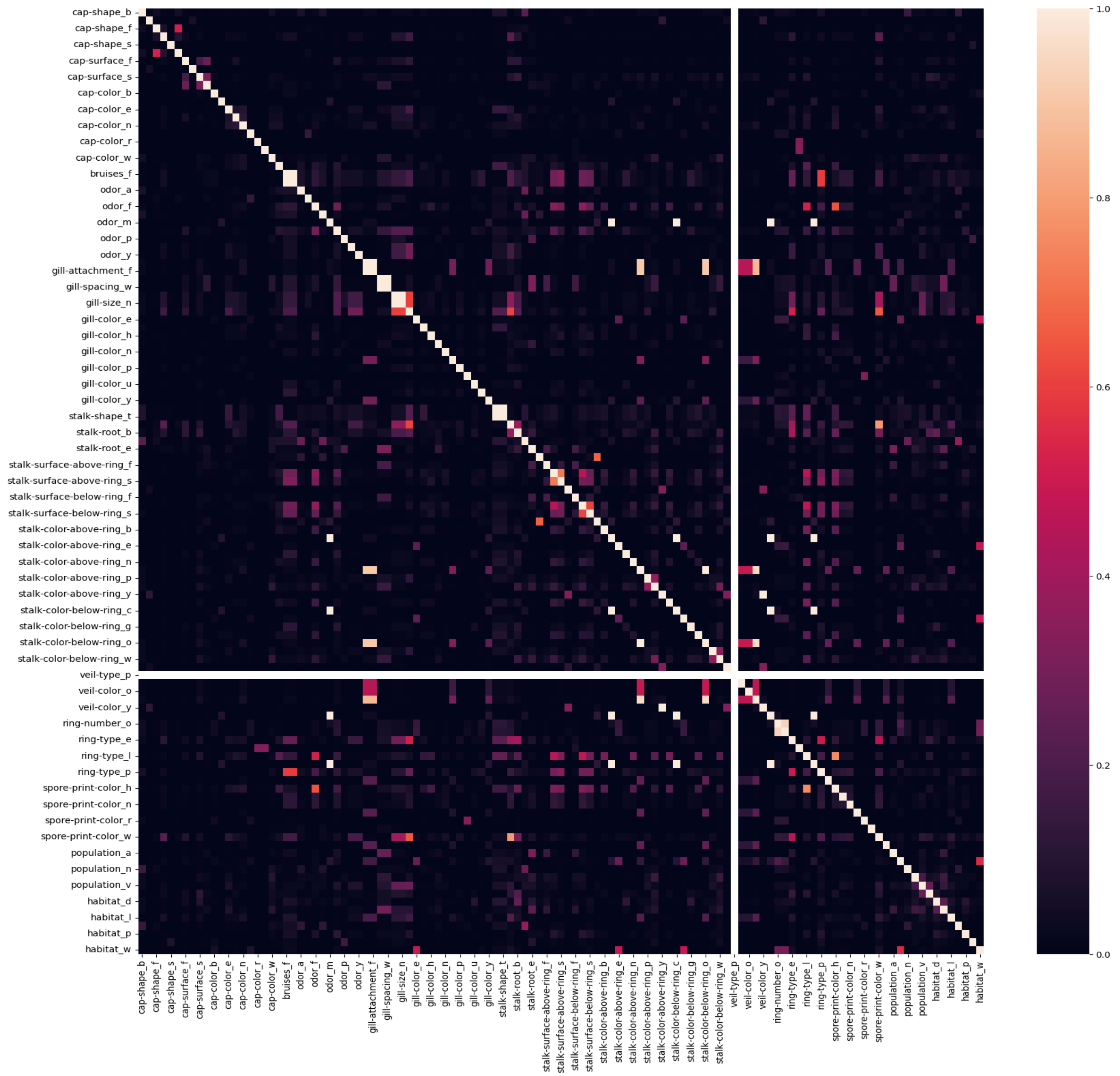
```
      dtype='object', length=117)
   cap-shape_b  cap-shape_c  cap-shape_f  cap-shape_k  cap-shape_s  \
0        False        False        False        False        False
1        False        False        False        False        False

   cap-shape_x  cap-surface_f  cap-surface_g  cap-surface_s  cap-surface_y  \
0         True          False          False           True          False
1         True          False          False           True          False

   cap-color_b  cap-color_c  cap-color_e  cap-color_g  cap-color_n  \
0        False        False        False        False         True
1        False        False        False        False        False

   cap-color_p  cap-color_r  cap-color_u  cap-color_w  cap-color_y  bruises_f  \
0        False        False        False        False        False      False
1        False        False        False        False         True      False

   bruises_t  odor_a  odor_c  odor_f  odor_l  odor_m  odor_n  odor_p  odor_s  \
0       True   False   False   False   False   False   False    True   False
```



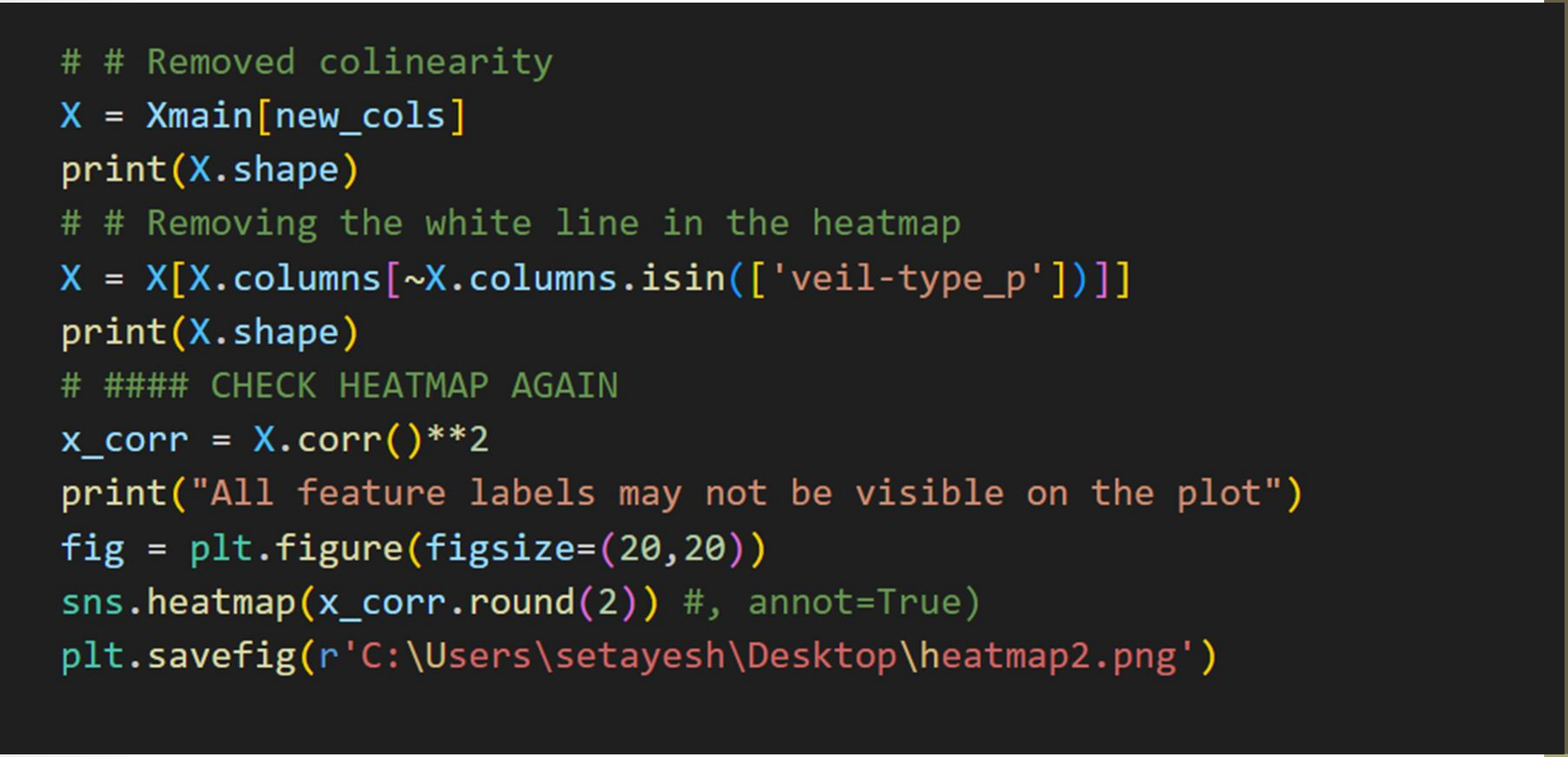
```
# #### CHECK FOR COLLINEARITY
# # if there are features that do the same thing you want to remove them
x_corr = Xmain.corr()**2
print(x_corr)
# # print("All feature labels may not be visible on the plot")
fig = plt.figure(figsize=(20,20))
sns.heatmap(x_corr.round(2)) #, annot=True)
plt.savefig(r'C:\Users\setayesh\Desktop\heatmap.png')
# #### Remove correlated columns or features
def remove_colinear_cols(X):
    cols = list(X.columns)
    print("Numer of features (before):", len(cols))

    for col in cols:
        for icol in cols:
            if(col!=icol):
                rsq = np.corrcoef(X[col], X[icol])[0,1]**2
                if((rsq >=0.7) | (rsq <= -0.7)):
                    cols.remove(icol)
    print('Number of features (after):', len(cols))
    return cols
# # Update keep only non-colinear features
new_cols = remove_colinear_cols(Xmain)
```



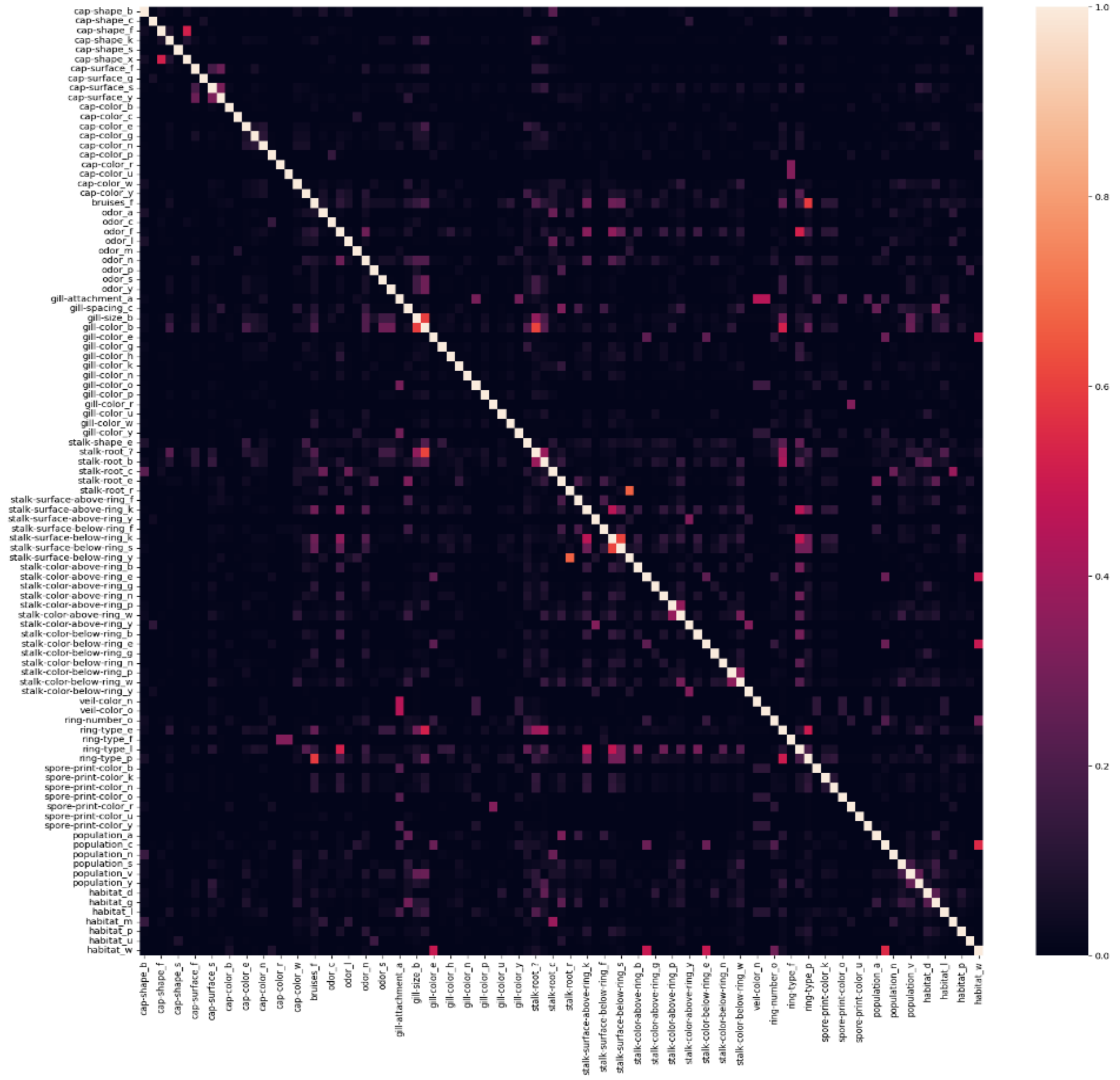
```
veil-type_p                    NaN       NaN            NaN
veil-color_n              0.002270  0.000053   1.252471e-02
veil-color_o              0.002270  0.000053   1.252471e-02
veil-color_w              0.004791  0.000112   2.246320e-02
veil-color_y              0.000187  0.000004   9.411180e-04
ring-number_n             0.000845  1.000000   4.249692e-03
ring-number_o             0.016121  0.052405   7.965652e-03
ring-number_t             0.015136  0.000355   1.171259e-02
ring-type_e               0.098523  0.002310   4.955919e-01
ring-type_f               0.001128  0.000026   5.674676e-03
ring-type_l               1.000000  0.000845   1.812206e-01
ring-type_n               0.000845  1.000000   4.249692e-03
ring-type_p               0.181221  0.004250   1.000000e+00
spore-print-color_b       0.001128  0.000026   6.225135e-03
spore-print-color_h       0.755040  0.001119   9.791036e-02
spore-print-color_k       0.056833  0.001333   1.125323e-01
spore-print-color_n       0.060679  0.001423   1.281226e-01
spore-print-color_o       0.001128  0.000026   6.225135e-03
spore-print-color_r       0.001697  0.000040   9.365535e-03
spore-print-color_u       0.001128  0.000026   6.225135e-03
spore-print-color_w       0.079020  0.010691   1.975982e-01
spore-print-color_y       0.001128  0.000026   6.225135e-03
population_a              0.009417  0.000221   4.736815e-02
population_c              0.008291  0.101903   5.825297e-03
population_n              0.009829  0.000231   5.424024e-02
population_s              0.034450  0.000808   3.019351e-02
population_v              0.000006  0.004403   4.142484e-02
```

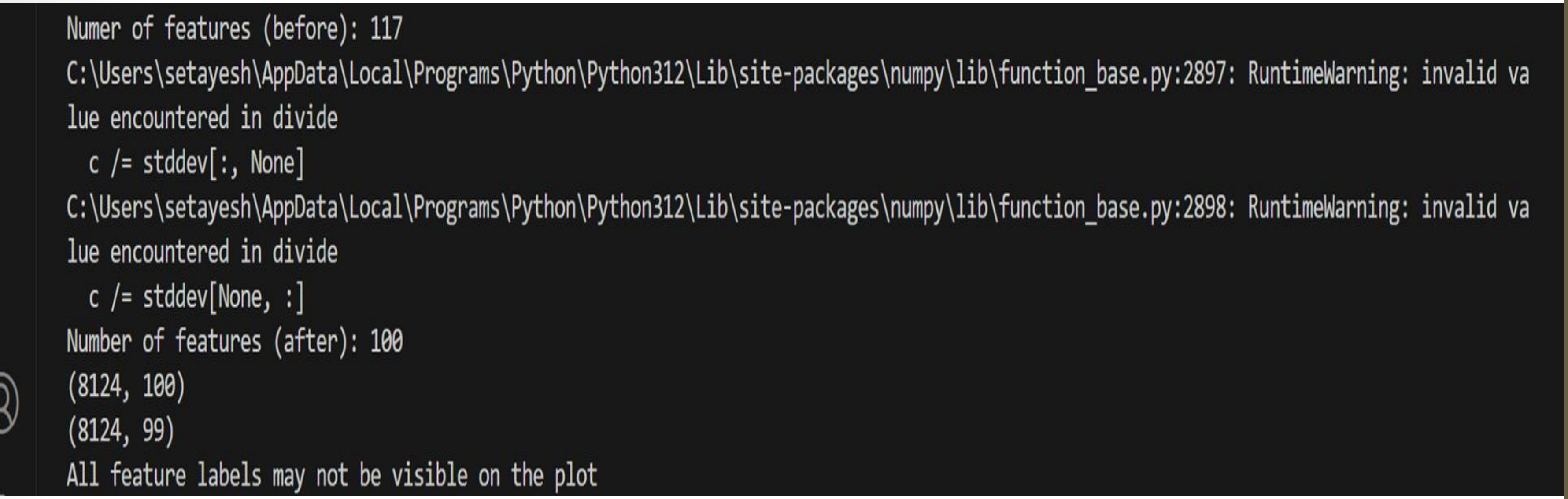


cap-shape_b
cap-shape_f
cap-shape_s
cap-surface_f
cap-surface_s
cap-color_b
cap-color_e
cap-color_n
cap-color_r
cap-color_w
bruises_f
odor_a
odor_f
odor_m
odor_p
odor_y
gill-attachment_f
gill-spacing_w
gill-size_n
gill-color_e
gill-color_h
gill-color_n
gill-color_p
gill-color_u
gill-color_y
stalk-shape_t
stalk-root_b
stalk-root_e
stalk-surface-above-ring_f
stalk-surface-above-ring_s
stalk-surface-below-ring_f
stalk-surface-below-ring_s
stalk-color-above-ring_b
stalk-color-above-ring_e
stalk-color-above-ring_n
stalk-color-above-ring_p
stalk-color-above-ring_y
stalk-color-below-ring_c
stalk-color-below-ring_g
stalk-color-below-ring_o
stalk-color-below-ring_w
veil-type_p
veil-color_o
veil-color_y
ring-number_o
ring-type_e
ring-type_l
ring-type_p
spore-print-color_h
spore-print-color_n
spore-print-color_r
spore-print-color_w
population_a
population_n
population_v
habitat_d
habitat_l
habitat_p
habitat_w
1.0
0.8
0.6
0.4
0.2
0.0



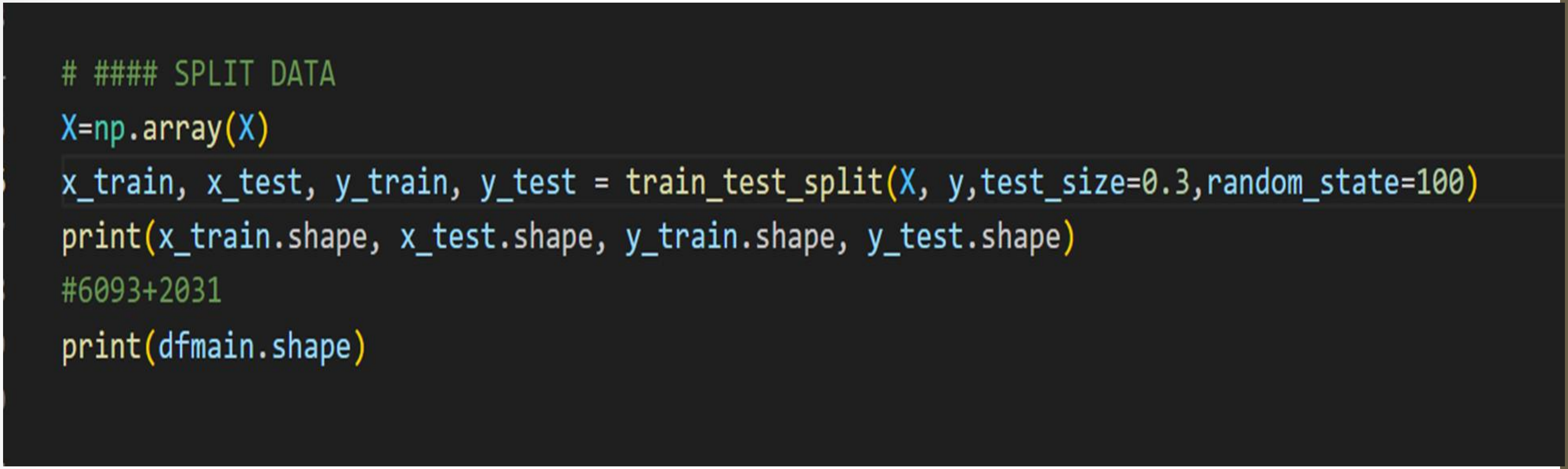
```python
# # Removed colinearity
X = Xmain[new_cols]
print(X.shape)
# # Removing the white line in the heatmap
X = X[X.columns[~X.columns.isin(['veil-type_p'])]]
print(X.shape)
# #### CHECK HEATMAP AGAIN
x_corr = X.corr()**2
print("All feature labels may not be visible on the plot")
fig = plt.figure(figsize=(20,20))
sns.heatmap(x_corr.round(2)) #, annot=True)
plt.savefig(r'C:\Users\setayesh\Desktop\heatmap2.png')
```


cap-shape_b
cap-shape_c
cap-shape_f
cap-shape_k
cap-shape_s
cap-shape_x
cap-surface_f
cap-surface_g
cap-surface_s
cap-surface_y
cap-color_b
cap-color_c
cap-color_e
cap-color_g
cap-color_n
cap-color_p
cap-color_r
cap-color_u
cap-color_w
cap-color_y
bruises_f
odor_a
odor_c
odor_f
odor_l
odor_m
odor_n
odor_p
odor_s
odor_y
gill-attachment_a
gill-spacing_c
gill-size_b
gill-color_b
gill-color_e
gill-color_g
gill-color_h
gill-color_k
gill-color_n
gill-color_o
gill-color_p
gill-color_r
gill-color_u
gill-color_w
gill-color_y
stalk-shape_e
stalk-root_?
stalk-root_b
stalk-root_c
stalk-root_e
stalk-root_r
stalk-surface-above-ring_f
stalk-surface-above-ring_k
stalk-surface-above-ring_y
stalk-surface-below-ring_f
stalk-surface-below-ring_k
stalk-surface-below-ring_s
stalk-surface-below-ring_y
stalk-color-above-ring_b
stalk-color-above-ring_e
stalk-color-above-ring_g
stalk-color-above-ring_n
stalk-color-above-ring_p
stalk-color-above-ring_w
stalk-color-above-ring_y
stalk-color-below-ring_b
stalk-color-below-ring_e
stalk-color-below-ring_g
stalk-color-below-ring_n
stalk-color-below-ring_p
stalk-color-below-ring_w
stalk-color-below-ring_y
veil-color_n
veil-color_o
ring-number_o
ring-type_e
ring-type_f
ring-type_l
ring-type_p
spore-print-color_b
spore-print-color_k
spore-print-color_n
spore-print-color_o
spore-print-color_r
spore-print-color_u
spore-print-color_y
population_a
population_c
population_n
population_s
population_v
population_y
habitat_d
habitat_g
habitat_l
habitat_m
habitat_p
habitat_u
habitat_w
1.0
0.8
0.6
0.4
0.2
0.0

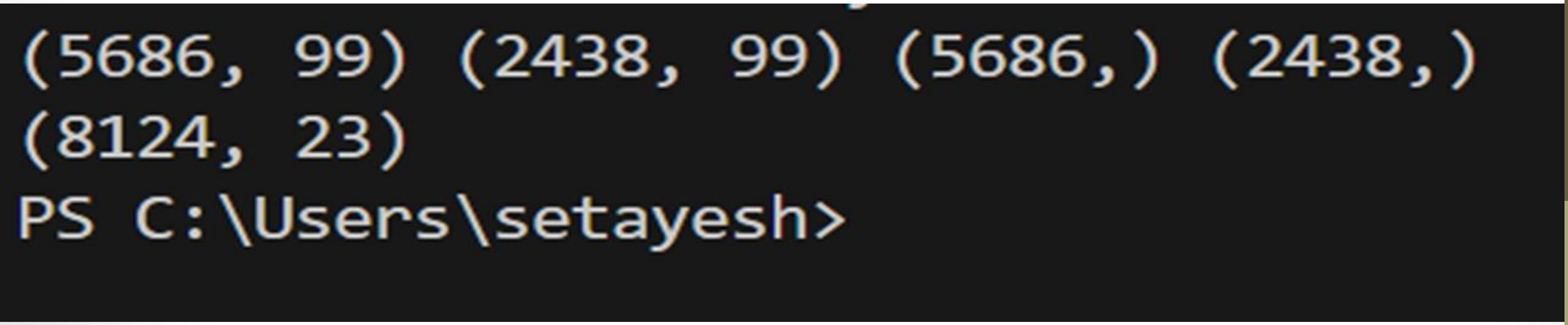


```
Numer of features (before): 117
C:\Users\setayesh\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\lib\function_base.py:2897: RuntimeWarning: invalid va
lue encountered in divide
  c /= stddev[:, None]
C:\Users\setayesh\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\lib\function_base.py:2898: RuntimeWarning: invalid va
lue encountered in divide
  c /= stddev[None, :]
Number of features (after): 100
(8124, 100)
(8124, 99)
All feature labels may not be visible on the plot
```

We split the data for our to phases (training, where we train our data and creat our Decision tree basedon iit and testing, where we test our data in our decision tree) we use 30% of our data for tseting
We split the training and testing data into x and y,
x being our features and y our target class

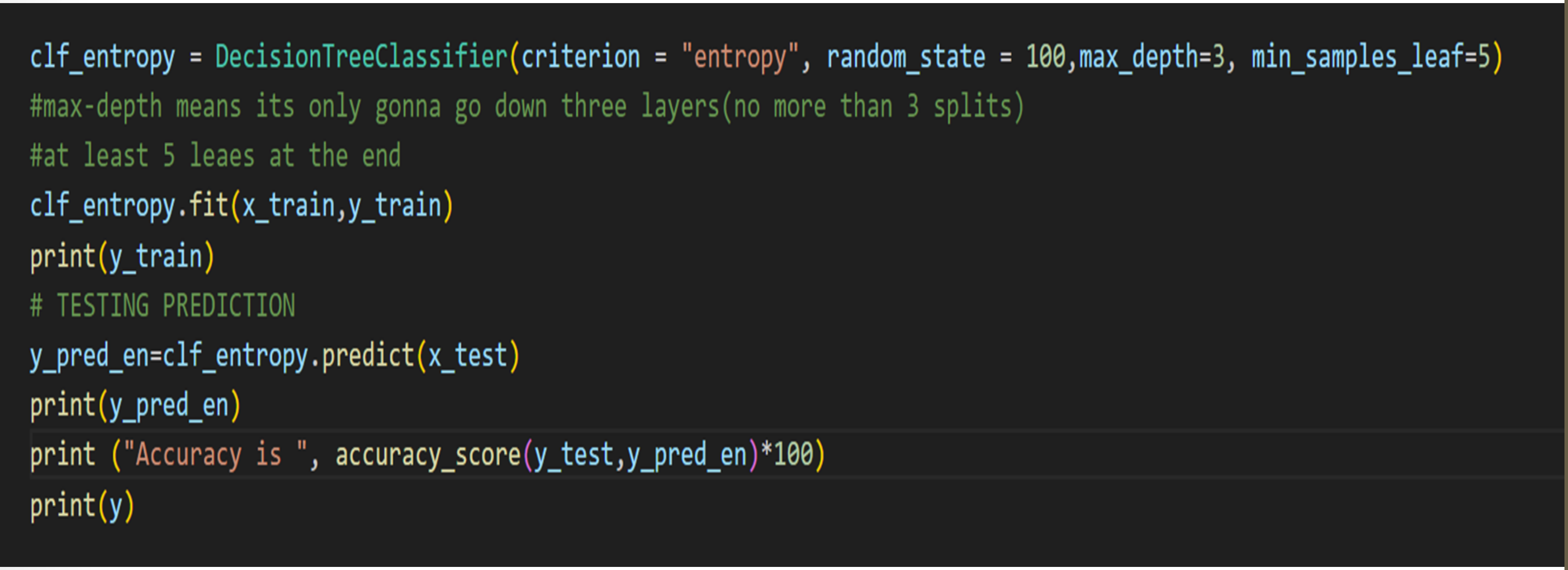


```
# #### SPLIT DATA
X=np.array(X)
x_train, x_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_state=100)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
#6093+2031
print(dfmain.shape)
```

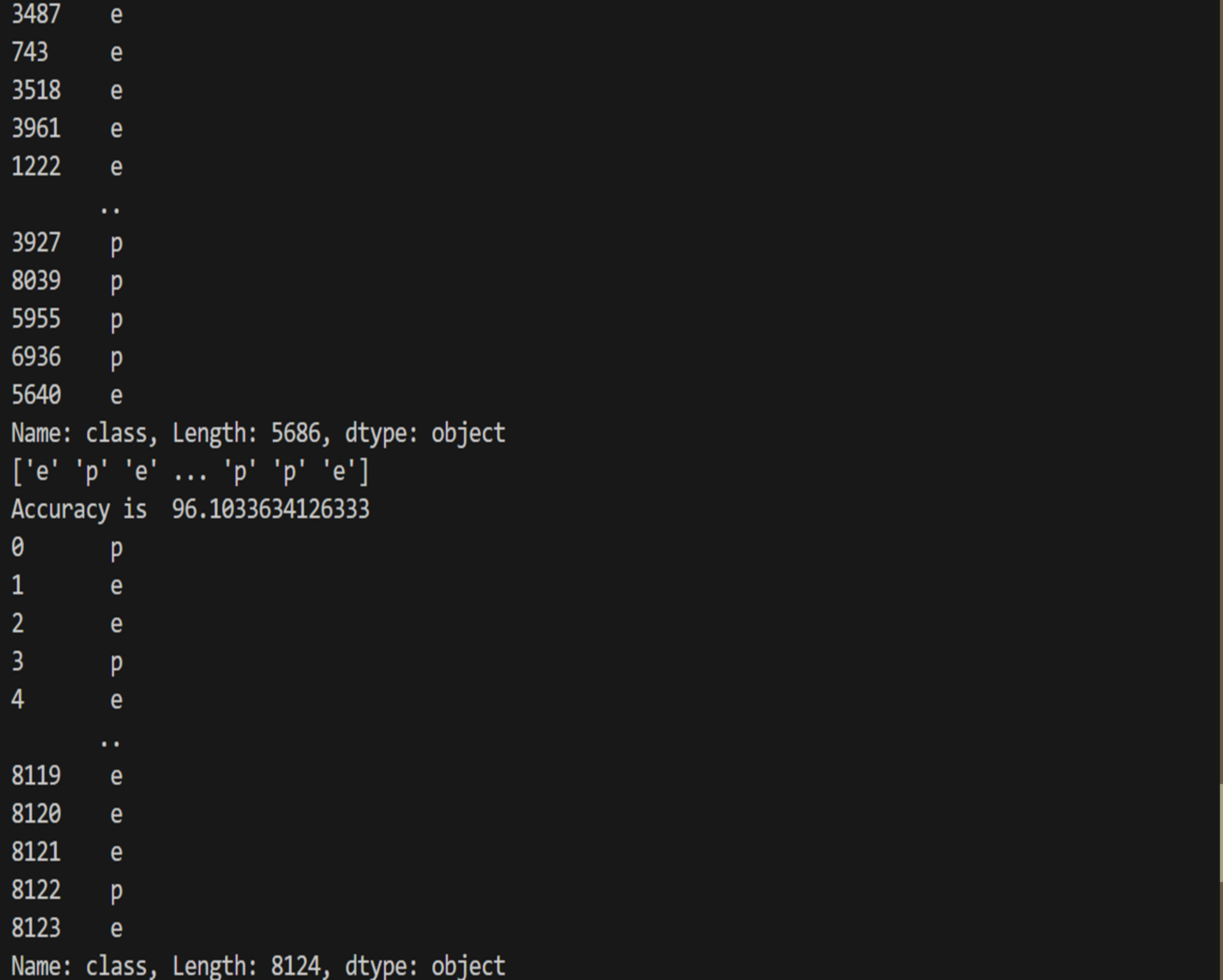


```
(5686, 99) (2438, 99) (5686,) (2438,)
(8124, 23)
PS C:\Users\setayesh>
```

We then call the DecisionTreeClassifier function to make our tree and put our xtrain and ytain Into the algorithm
Then we test our xtest using predict function
And at last for obtaining the accuracy of our tree we compare our ytest
(the target classes we got by predicting with our tree)and ytrain (the true answers)



```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,max_depth=3, min_samples_leaf=5)
#max-depth means its only gonna go down three layers(no more than 3 splits)
#at least 5 leaes at the end
clf_entropy.fit(x_train,y_train)
print(y_train)
# TESTING PREDICTION
y_pred_en=clf_entropy.predict(x_test)
print(y_pred_en)
print ("Accuracy is ", accuracy_score(y_test,y_pred_en)*100)
print(y)
```



```
3487    e
743     e
3518    e
3961    e
1222    e
       ..
3927    p
8039    p
5955    p
6936    p
5640    e
Name: class, Length: 5686, dtype: object
['e' 'p' 'e' ... 'p' 'p' 'e']
Accuracy is  96.1033634126333
0       p
1       e
2       e
3       p
4       e
       ..
8119    e
8120    e
8121    e
8122    p
8123    e
Name: class, Length: 8124, dtype: object
```

If we have missing values :

- 1. Deletion (Removing Data)
- 2. Imputation (Filling in Missing Data)
- 3. Using Algorithms that Support Missing Values
- 4. Using Data Augmentation Techniques

Deletion(removing data) :

•**List-wise Deletion**:
Remove any rows that contain missing values.
This method is simple and ensures that only complete cases are used,
but it can lead to a significant reduction in the dataset size,
potentially losing valuable information.

•**Pairwise Deletion**:
Use available data without removing entire rows,
applying analysis only on the available values.
This preserves more data but can result in inconsistent datasets for
different analyses.

Imputation(Filling in missing data):

•**Mean/Median/Mode Imputation**:
 Replace missing values with the mean, median, or mode of the respective column. This method is straightforward and maintains the dataset size but can reduce variance and may not be suitable for skewed data distributions.
•**Predictive Modeling**:
Use machine learning models like k-Nearest Neighbors (k-NN),
 regression, or more complex algorithms to predict and fill in missing values based on other variables.
This approach can provide more accurate imputed values but is computationally intensive.
•**Multiple Imputation**:
 Create several different imputed datasets and combine the results to account for the uncertainty around the missing data, leading to more robust
 results.

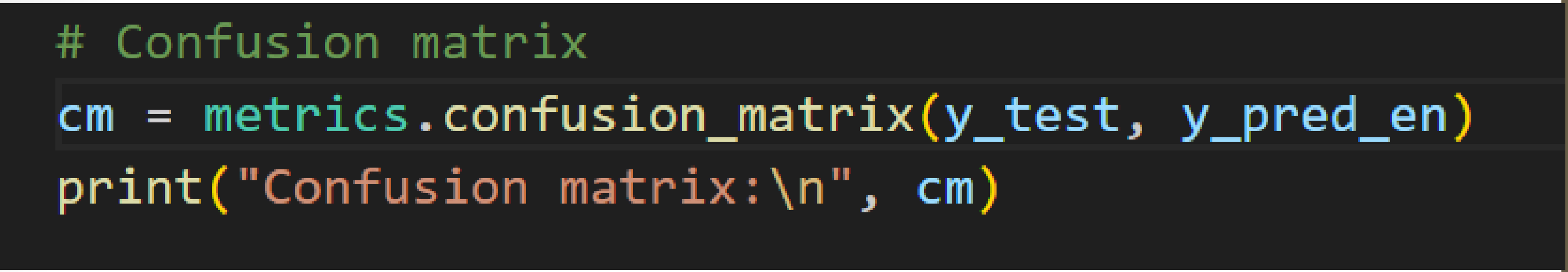
Using algorithms that support missing values :

- Certain algorithms, such as decision trees and ensemble methods like Random Forests, can handle missing values natively during the training process. These algorithms can bypass the need for explicit imputation by treating missing values as a separate category or by using surrogate splits.

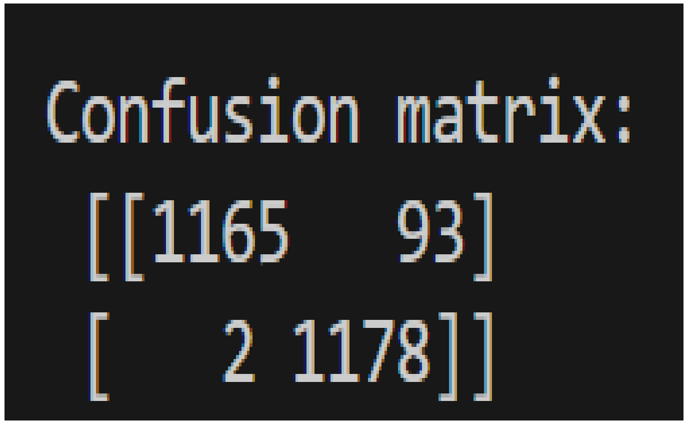
Using augmentation technique:

- Data Augmentation: Generate new data points by perturbing existing data, which can help mitigate the impact of missing values. Techniques like bootstrapping or creating synthetic data can enhance dataset robustness. Iterative Imputation: Iteratively predict and fill in missing values using models trained on other features. This method involves multiple rounds of imputation, with each round using the imputed values from the previous round, often converging to more accurate estimates.

Confusion matrix :



```python
# Confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred_en)
print("Confusion matrix:\n", cm)
```



```
Confusion matrix:
 [[1165   93]
 [   2 1178]]
```