prediction-of-poisonous-mushrooms

November 15, 2024

[1]:
```python
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

```
/kaggle/input/playground-series-s4e8/sample_submission.csv
/kaggle/input/playground-series-s4e8/train.csv
/kaggle/input/playground-series-s4e8/test.csv
```

[2]:
```python
data = pd.read_csv("/kaggle/input/playground-series-s4e8/train.csv")
data
```

[2]:
```
              id class  cap-diameter cap-shape cap-surface cap-color  \
0              0     e          8.80         f           s         u
1              1     p          4.51         x           h         o
2              2     e          6.94         f           s         b
3              3     e          3.88         f           y         g
4              4     e          5.85         x           l         w
…           …   …          …      …         …         …
```

```
3116940  3116940     e          9.29          f        NaN         n
3116941  3116941     e         10.88          s        NaN         w
3116942  3116942     p          7.82          x          e         e
3116943  3116943     e          9.45          p          i         n
3116944  3116944     p          3.20          x          s         g

        does-bruise-or-bleed gill-attachment gill-spacing gill-color  …  \
0                          f               a            c          w  …
1                          f               a            c          n  …
2                          f               x            c          w  …
3                          f               s          NaN          g  …
4                          f               d          NaN          w  …
…                        …             …          …         …  …
3116940                    t             NaN          NaN          w  …
3116941                    t               d            c          p  …
3116942                    f               a          NaN          w  …
3116943                    t               e          NaN          p  …
3116944                    f               d            c          w  …

         stem-root  stem-surface stem-color veil-type veil-color has-ring  \
0              NaN           NaN          w       NaN        NaN        f
1              NaN             y          o       NaN        NaN        t
2              NaN             s          n       NaN        NaN        f
3              NaN           NaN          w       NaN        NaN        f
4              NaN           NaN          w       NaN        NaN        f
…            …           …        …       …       …       …
3116940          b           NaN          w         u          w        t
3116941        NaN           NaN          w       NaN        NaN        f
3116942        NaN           NaN          y       NaN          w        t
3116943        NaN             y          w       NaN        NaN        t
3116944        NaN           NaN          w       NaN        NaN        f

        ring-type spore-print-color habitat season
0               f               NaN       d      a
1               z               NaN       d      w
2               f               NaN       l      w
3               f               NaN       d      u
4               f               NaN       g      a
…           …               …     …    …
3116940         g               NaN       d      u
3116941         f               NaN       d      u
3116942         z               NaN       d      a
3116943         p               NaN       d      u
3116944         f               NaN       g      u

[3116945 rows x 22 columns]
```

[3]:
```python
data.isnull().sum()
```

[3]:
```
id                            0
class                         0
cap-diameter                  4
cap-shape                    40
cap-surface              671023
cap-color                    12
does-bruise-or-bleed          8
gill-attachment          523936
gill-spacing            1258435
gill-color                   57
stem-height                   0
stem-width                    0
stem-root               2757023
stem-surface            1980861
stem-color                   38
veil-type               2957493
veil-color              2740947
has-ring                     24
ring-type                128880
spore-print-color       2849682
habitat                      45
season                        0
dtype: int64
```

0.0.1 Removing Columns with Many Missing Values

[4]:
```python
data.drop(columns=["spore-print-color", "stem-root", "gill-spacing",
    "veil-type", "stem-surface"], inplace=True)
data
```

[4]:
```
              id class  cap-diameter cap-shape cap-surface cap-color  \
0              0     e          8.80         f           s         u
1              1     p          4.51         x           h         o
2              2     e          6.94         f           s         b
3              3     e          3.88         f           y         g
4              4     e          5.85         x           l         w
…             …     …             …         …           …
3116940  3116940     e          9.29         f         NaN         n
3116941  3116941     e         10.88         s         NaN         w
3116942  3116942     p          7.82         x           e         e
3116943  3116943     e          9.45         p           i         n
3116944  3116944     p          3.20         x           s         g

        does-bruise-or-bleed gill-attachment gill-color  stem-height  \
0                          f               a          w         4.51
```

```
1                           f               a          n          4.79
2                           f               x          w          6.85
3                           f               s          g          4.16
4                           f               d          w          3.37
…                        …             …         …          …
3116940                     t             NaN          w         12.14
3116941                     t               d          p          6.65
3116942                     f               a          w          9.51
3116943                     t               e          p          9.13
3116944                     f               d          w          2.82

         stem-width stem-color veil-color has-ring ring-type habitat season
0             15.39          w        NaN        f         f       d      a
1              6.48          o        NaN        t         z       d      w
2              9.93          n        NaN        f         f       l      w
3              6.53          w        NaN        f         f       d      u
4              8.36          w        NaN        f         f       g      a
…              …          …        …       …       …       …     …
3116940       18.81          w          w        t         g       d      u
3116941       26.97          w        NaN        f         f       d      u
3116942       11.06          y          w        t         z       d      a
3116943       17.77          w        NaN        t         p       d      u
3116944        7.79          w        NaN        f         f       g      u

[3116945 rows x 17 columns]
```

0.0.2 Checking the null values

[5]:
```python
data.isnull().sum()
```

[5]:
```
id                            0
class                         0
cap-diameter                  4
cap-shape                    40
cap-surface              671023
cap-color                    12
does-bruise-or-bleed          8
gill-attachment          523936
gill-color                   57
stem-height                   0
stem-width                    0
stem-color                   38
veil-color              2740947
has-ring                     24
ring-type                128880
habitat                      45
season                        0
```

```
dtype: int64
```

0.0.3 Dropping the column "veil-color"

[6]:
```python
data.drop(columns=["veil-color"])
```

[6]:
```
              id class  cap-diameter cap-shape cap-surface cap-color  \
0              0     e          8.80         f           s         u
1              1     p          4.51         x           h         o
2              2     e          6.94         f           s         b
3              3     e          3.88         f           y         g
4              4     e          5.85         x           l         w
…            …   …           …       …       …       …
3116940  3116940     e          9.29         f         NaN         n
3116941  3116941     e         10.88         s         NaN         w
3116942  3116942     p          7.82         x           e         e
3116943  3116943     e          9.45         p           i         n
3116944  3116944     p          3.20         x           s         g

        does-bruise-or-bleed gill-attachment gill-color  stem-height  \
0                          f               a          w         4.51
1                          f               a          n         4.79
2                          f               x          w         6.85
3                          f               s          g         4.16
4                          f               d          w         3.37
…                       …             …        …          …
3116940                    t             NaN          w        12.14
3116941                    t               d          p         6.65
3116942                    f               a          w         9.51
3116943                    t               e          p         9.13
3116944                    f               d          w         2.82

         stem-width stem-color has-ring ring-type habitat season
0             15.39          w        f         f       d      a
1              6.48          o        t         z       d      w
2              9.93          n        f         f       l      w
3              6.53          w        f         f       d      u
4              8.36          w        f         f       g      a
…               …        …      …      …     …
3116940       18.81          w        t         g       d      u
3116941       26.97          w        f         f       d      u
3116942       11.06          y        t         z       d      a
3116943       17.77          w        t         p       d      u
3116944        7.79          w        f         f       g      u

[3116945 rows x 16 columns]
```

0.0.4 Now will replace missing value of the columns with their respectuive mode values

[7]:
```
mode_value = data['cap-shape'].mode()[0]

data['cap-shape'].fillna(mode_value, inplace=True)
```

```
/tmp/ipykernel_17/182498652.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['cap-shape'].fillna(mode_value, inplace=True)
```

[8]:
```
mode_value1 = data['cap-surface'].mode()[0]


data['cap-surface'].fillna(mode_value1, inplace=True)
```

```
/tmp/ipykernel_17/4275872860.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['cap-surface'].fillna(mode_value1, inplace=True)
```

[9]:
```
mode_value2 = data['cap-color'].mode()[0]

data['cap-color'].fillna(mode_value2, inplace=True)
```

```
/tmp/ipykernel_17/885123063.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
```

```
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['cap-color'].fillna(mode_value2, inplace=True)
```

[10]:
```python
mode_value3 = data['ring-type'].mode()[0]


data['ring-type'].fillna(mode_value3, inplace=True)
```

```
/tmp/ipykernel_17/2988485115.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['ring-type'].fillna(mode_value3, inplace=True)
```

[11]:
```python
mode_value4 = data['gill-attachment'].mode()[0]

data['gill-attachment'].fillna(mode_value4, inplace=True)
```

```
/tmp/ipykernel_17/1278174316.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['gill-attachment'].fillna(mode_value4, inplace=True)
```

[12]:
```
mode_value5 = data['gill-color'].mode()[0]


data['gill-color'].fillna(mode_value4, inplace=True)
```

```
/tmp/ipykernel_17/319217220.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['gill-color'].fillna(mode_value4, inplace=True)
```

[13]:
```
mode_value6 = data['has-ring'].mode()[0]

data['has-ring'].fillna(mode_value6, inplace=True)
```

```
/tmp/ipykernel_17/86164832.py:3: FutureWarning: A value is trying to be set on a
copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['has-ring'].fillna(mode_value6, inplace=True)
```

[14]:
```
mode_value7= data['habitat'].mode()[0]


data['habitat'].fillna(mode_value7, inplace=True)
```

```
/tmp/ipykernel_17/849800396.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['habitat'].fillna(mode_value7, inplace=True)
```

[15]:
```python
mode_value8 = data['stem-color'].mode()[0]


data['stem-color'].fillna(mode_value8, inplace=True)
```

```
/tmp/ipykernel_17/797109304.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['stem-color'].fillna(mode_value8, inplace=True)
```

[16]:
```python
mode_value9 = data['does-bruise-or-bleed'].mode()[0]


data['does-bruise-or-bleed'].fillna(mode_value9, inplace=True)
```

```
/tmp/ipykernel_17/2258548946.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['does-bruise-or-bleed'].fillna(mode_value9, inplace=True)
```

0.0.5

Now will replace missing value of the column "Cap-diameter" with it's median value

[17]:
```python
median_value = data['cap-diameter'].median()


data['cap-diameter'].fillna(median_value, inplace=True)
```

```
/tmp/ipykernel_17/2591166181.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data['cap-diameter'].fillna(median_value, inplace=True)
```

[18]:
```python
data.drop(columns=["veil-color"])
```

[18]:
```
              id class  cap-diameter cap-shape cap-surface cap-color  \
0              0     e          8.80         f           s         u
1              1     p          4.51         x           h         o
2              2     e          6.94         f           s         b
3              3     e          3.88         f           y         g
4              4     e          5.85         x           l         w
…            …     …            …         …         …
3116940  3116940     e          9.29         f           t         n
3116941  3116941     e         10.88         s           t         w
3116942  3116942     p          7.82         x           e         e
3116943  3116943     e          9.45         p           i         n
3116944  3116944     p          3.20         x           s         g

        does-bruise-or-bleed gill-attachment gill-color  stem-height  \
0                          f               a          w         4.51
1                          f               a          n         4.79
2                          f               x          w         6.85
3                          f               s          g         4.16
4                          f               d          w         3.37
…                      …             …          …         …
3116940                    t               a          w        12.14
3116941                    t               d          p         6.65
3116942                    f               a          w         9.51
```

```
3116943                    t               e          p         9.13
3116944                    f               d          w         2.82

         stem-width stem-color has-ring ring-type habitat season
0             15.39          w        f         f       d      a
1              6.48          o        t         z       d      w
2              9.93          n        f         f       l      w
3              6.53          w        f         f       d      u
4              8.36          w        f         f       g      a
…             …       …      …       …      …    …
3116940       18.81          w        t         g       d      u
3116941       26.97          w        f         f       d      u
3116942       11.06          y        t         z       d      a
3116943       17.77          w        t         p       d      u
3116944        7.79          w        f         f       g      u

[3116945 rows x 16 columns]
```

0.0.6 Label Encoding :

In machine learning, many algorithms require that the input features be numerical. However, categorical features are often represented as strings, which models cannot directly process.

Label encoding is a technique used to convert categorical variables into numerical values. This involves assigning each unique category a different integer value.

0.0.7 Reasons for Using Label Encoding:

1. Model Compatibility:

Many machine learning models, especially those based on mathematical computations (e.g., linear regression, SVM), require numerical input features.

2. Efficiency:

Label encoding converts categories to integers, which can be more efficient in terms of memory and computation compared to handling string data.

3. Order Preservation:

If the categorical variable has an inherent order , label encoding preserves this order, making it beneficial for models that assume ordered relationships.

However, label encoding can introduce a potential issue when applied to nominal categories (those without a natural order), as it may imply a false sense of hierarchy

or order to the machine learning model. In such cases, one-hot encoding might be a better alternative.

[19]:
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder


# List of categorical columns
categorical_columns = [
    'class', 'cap-shape', 'cap-surface', 'cap-color', 'does-bruise-or-bleed',
    'gill-attachment', 'gill-color', 'stem-color', 'has-ring',
    'ring-type', 'habitat', 'season',"veil-color"
]

# Option 1: Label Encoding
# This will convert categories to integers
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le  # Save the label encoder for potential␣
 ↪inverse_transform later
```

0.0.8 Checking data type of all the columns

[20]:
```python
column_data_types = data.dtypes

# Print the data types of all columns
print(column_data_types)
```

```
id                        int64
class                     int64
cap-diameter            float64
cap-shape                 int64
cap-surface               int64
cap-color                 int64
does-bruise-or-bleed      int64
gill-attachment           int64
gill-color                int64
stem-height             float64
stem-width              float64
stem-color                int64
veil-color                int64
has-ring                  int64
ring-type                 int64
habitat                   int64
season                    int64
dtype: object
```

[21]:
```
data.isnull().sum()
```

[21]:
```
id                      0
class                   0
cap-diameter            0
cap-shape               0
cap-surface             0
cap-color               0
does-bruise-or-bleed    0
gill-attachment         0
gill-color              0
stem-height             0
stem-width              0
stem-color              0
veil-color              0
has-ring                0
ring-type               0
habitat                 0
season                  0
dtype: int64
```

0.0.9 Separating Features and Target Variable

In this step, we're separating the features (`X`) from the target variable (`y`):

- `X`: This will contain all the features used for prediction. I drop the `class` column from the dataset to ensure that only the features are included.

- `y`: This will contain the target variable, which is the `class` column that I aim to predict.

[22]:
```
X= data.drop(['class'], axis=1)
y = data['class']
```

0.0.10 Reading the test dataset

[23]:
```
test_data = pd.read_csv("/kaggle/input/playground-series-s4e8/test.csv")
```

Checking the unique values

[24]:
```
data["class"].nunique()
```

[24]: 2

[25]:
```
data
```

[25]:

```
              id  class  cap-diameter  cap-shape  cap-surface  cap-color  \
0              0      0          8.80         53           72         72
1              1      1          4.51         71           56         64
2              2      0          6.94         53           72         49
3              3      0          3.88         53           81         57
4              4      0          5.85         71           65         74
…            …      …           …         …          …
3116940  3116940      0          9.29         53           76         63
3116941  3116941      0         10.88         67           76         74
3116942  3116942      1          7.82         71           53         55
3116943  3116943      0          9.45         64           59         63
3116944  3116944      1          3.20         71           72         57

         does-bruise-or-bleed  gill-attachment  gill-color  stem-height  \
0                           8               44          59         4.51
1                           8               44          46         4.79
2                           8               75          59         6.85
3                           8               70          37         4.16
4                           8               47          59         3.37
…                        …               …          …         …
3116940                    20               44          59        12.14
3116941                    20               47          48         6.65
3116942                     8               44          59         9.51
3116943                    20               52          48         9.13
3116944                     8               47          59         2.82

         stem-width  stem-color  veil-color  has-ring  ring-type  habitat  \
0             15.39          55          24         5         18       25
1              6.48          47          24        18         39       25
2              9.93          46          24         5         18       36
3              6.53          55          24         5         18       25
4              8.36          55          24         5         18       29
…               …          …          …         …        …        …
3116940       18.81          55          21        18         19       25
3116941       26.97          55          24         5         18       25
3116942       11.06          57          21        18         39       25
3116943       17.77          55          24        18         27       25
3116944        7.79          55          24         5         18       29

         season
0             0
1             3
2             3
3             2
4             0
…           …
3116940       2
```

```
3116941       2
3116942       0
3116943       2
3116944       2

[3116945 rows x 17 columns]
```

0.0.11 Random Forest Classifier: Training and Evaluation

In this section, I use the Random Forest algorithm to train a classification model on the dataset and evaluate its performance.

Steps:

1. **Importing Libraries**:
   - I start by importing the necessary libraries: `pandas` for data manipulation, `RandomForestClassifier` from `sklearn.ensemble` for building the model, and `accuracy_score`, `precision_score`, and `recall_score` from `sklearn.metrics` to evaluate the model's performance.
2. **Preparing the Data**:
   - The input features (`X`) are created by dropping the `id` and `class` columns from the dataset. The `id` column is typically a unique identifier and does not contribute to the prediction, while the `class` column is the target variable.
   - The target variable (`Y`) is the `class` column, which we aim to predict.
3. **Defining the Model**:
   - I define the `RandomForestClassifier` with 20 trees (`n_estimators=20`) and set a random seed (`random_state=42`) to ensure reproducibility.
4. **Training the Model**:
   - The model is trained on the entire dataset using the `fit` method, which takes the input features (`X`) and the target variable (`Y`).
5. **Evaluating the Model**:
   - Predictions are made on the training data using the `predict` method.
   - The model's performance is evaluated using three metrics:
     - **Accuracy**: The proportion of correct predictions.
     - **Precision**: The proportion of true positive predictions among all positive predictions (weighted average is used here).
     - **Recall**: The proportion of true positives correctly identified by the model (weighted average is used here).
   - The results are printed with 4 decimal places.

[26]:
```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Load your dataset (assuming it's already loaded into a DataFrame named 'data')
X = data.drop(columns=['id', 'class'])  # Drop the 'id' and 'class' columns to␣
 ↪create the input features
Y = data['class']  # The output variable is the 'class' column
```

```python
# Define the RandomForestClassifier model
random_forest = RandomForestClassifier(n_estimators=20, random_state=42)

# Fit the model to the entire dataset
random_forest.fit(X, Y)

# Evaluate the model on the training data
y_train_pred = random_forest.predict(X)
accuracy = accuracy_score(Y, y_train_pred)
precision = precision_score(Y, y_train_pred, average='weighted')
recall = recall_score(Y, y_train_pred, average='weighted')

print(f'Training Accuracy: {accuracy:.4f}')
print(f'Training Precision: {precision:.4f}')
print(f'Training Recall: {recall:.4f}')
```

```
Training Accuracy: 0.9993
Training Precision: 0.9993
Training Recall: 0.9993
```

Confusion Matrix: Evaluating Model Performance In this section, I use the confusion matrix to further evaluate the performance of the Random Forest model on the training data.

What is a Confusion Matrix? A confusion matrix is a table that is often used to describe the performance of a classification model. It provides a detailed breakdown of the model's predictions, showing the counts of:

- **True Positives (TP)**: Correctly predicted positive class.
- **True Negatives (TN)**: Correctly predicted negative class.
- **False Positives (FP)**: Incorrectly predicted as positive when it was actually negative.
- **False Negatives (FN)**: Incorrectly predicted as negative when it was actually positive.

This breakdown allows you to assess the model's ability to differentiate between classes more precisely.

[27]:
```python
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y, y_train_pred)
print(conf_matrix)
```

```
[[1411016     533]
 [   1675 1703721]]
```

[28]:
```python
from sklearn.model_selection import train_test_split
```

0.0.12 Splitting the Dataset and Evaluating the Model on Test Data

In this section, I split the dataset into training and testing sets to evaluate how well the Random Forest model performs on unseen data. This helps to ensure that the model generalizes well and is not overfitting to the training data.

Steps:

1. **Splitting the Data**:
   - I split the dataset into training and test sets using `train_test_split` from `sklearn.model_selection`.
   - The dataset is split such that 80% of the data is used for training and 20% is reserved for testing. The `random_state=42` ensures that the split is reproducible.

[29]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42)

# Define the RandomForestClassifier model
random_forest = RandomForestClassifier(n_estimators=20, random_state=42)

# Fit the model to the training data
random_forest.fit(X_train, Y_train)

# Make predictions on the test data
y_test_pred = random_forest.predict(X_test)

# Evaluate the model on the test data
accuracy = accuracy_score(Y_test, y_test_pred)
precision = precision_score(Y_test, y_test_pred, average='weighted')
recall = recall_score(Y_test, y_test_pred, average='weighted')

print(f'Test Accuracy: {accuracy:.4f}')
print(f'Test Precision: {precision:.4f}')
print(f'Test Recall: {recall:.4f}')
```

```
Test Accuracy: 0.9885
Test Precision: 0.9885
Test Recall: 0.9885
```

0.0.13 Preprocessing the test data

[30]:
```
test_data = pd.read_csv('/kaggle/input/playground-series-s4e8/test.csv')

# Print the test data values
print("Test Data:")
print(test_data.head())
```

```
Test Data:
        id  cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0  3116945          8.64         x         NaN         n                    t
1  3116946          6.90         o           t         o                    f
2  3116947          2.00         b           g         n                    f
```

```
3  3116948          3.47         x          t         n                 f
4  3116949          6.17         x          h         y                 f

  gill-attachment gill-spacing gill-color  stem-height  …  stem-root  \
0             NaN          NaN          w        11.13  …          b
1             NaN            c          y         1.27  …        NaN
2             NaN            c          n         6.18  …        NaN
3               s            c          n         4.98  …        NaN
4               p          NaN          y         6.73  …        NaN

  stem-surface stem-color veil-type veil-color has-ring ring-type  \
0          NaN          w         u          w        t         g
1          NaN          n       NaN        NaN        f         f
2          NaN          n       NaN        NaN        f         f
3          NaN          w       NaN          n        t         z
4          NaN          y       NaN          y        t       NaN

  spore-print-color habitat season
0               NaN       d      a
1               NaN       d      a
2               NaN       d      s
3               NaN       d      u
4               NaN       d      u

[5 rows x 21 columns]
```

Here I am following the same steps I have done for training data

[31]:
```
test_data.drop(columns=["veil-color"])
```

[31]:
```
              id  cap-diameter cap-shape cap-surface cap-color  \
0        3116945          8.64         x         NaN         n
1        3116946          6.90         o           t         o
2        3116947          2.00         b           g         n
3        3116948          3.47         x           t         n
4        3116949          6.17         x           h         y
…            …             …         …           …         …
2077959  5194904          0.88         x           g         w
2077960  5194905          3.12         x           s         w
2077961  5194906          5.73         x           e         e
2077962  5194907          5.03         b           g         n
2077963  5194908         15.51         f         NaN         w

        does-bruise-or-bleed gill-attachment gill-spacing gill-color  \
0                          t             NaN          NaN          w
1                          f             NaN            c          y
2                          f             NaN            c          n
3                          f               s            c          n
```

```
4                          f               p          NaN           y
…                     …             …          …        …
2077959                    f               a            d           w
2077960                    f               d            c           w
2077961                    f               a          NaN           w
2077962                    f               a            d           g
2077963                    f               d            c           y

         stem-height  stem-width stem-root stem-surface stem-color veil-type  \
0              11.13       17.12         b          NaN          w         u
1               1.27       10.75       NaN          NaN          n       NaN
2               6.18        3.14       NaN          NaN          n       NaN
3               4.98        8.51       NaN          NaN          w       NaN
4               6.73       13.70       NaN          NaN          y       NaN
…                …           …       …          …         …         …
2077959         2.67        1.35       NaN          NaN          e       NaN
2077960         2.69        7.38       NaN          NaN          w       NaN
2077961         6.16        9.74       NaN          NaN          y       NaN
2077962         6.00        3.46       NaN            s          g       NaN
2077963         2.69       17.71       NaN          NaN          w       NaN

        has-ring ring-type spore-print-color habitat season
0              t         g               NaN       d      a
1              f         f               NaN       d      a
2              f         f               NaN       d      s
3              t         z               NaN       d      u
4              t       NaN               NaN       d      u
…            …       …               …        …    …
2077959        f         f               NaN       d      u
2077960        f         f               NaN       g      a
2077961        t         z               NaN       d      a
2077962        f         f               NaN       d      a
2077963        f         f               NaN       d      w

[2077964 rows x 20 columns]
```

[32]:
```
test_data_new = test_data.
  ↪drop(columns=["stem-root","veil-type","stem-surface","gill-spacing","spore-print-color"])
test_data_new
```

[32]:
```
              id  cap-diameter cap-shape cap-surface cap-color  \
0        3116945          8.64         x         NaN         n
1        3116946          6.90         o           t         o
2        3116947          2.00         b           g         n
3        3116948          3.47         x           t         n
4        3116949          6.17         x           h         y
…           …            …        …          …          …
```

```
2077959  5194904          0.88          x          g          w
2077960  5194905          3.12          x          s          w
2077961  5194906          5.73          x          e          e
2077962  5194907          5.03          b          g          n
2077963  5194908         15.51          f        NaN          w

        does-bruise-or-bleed gill-attachment gill-color  stem-height  \
0                          t             NaN          w        11.13
1                          f             NaN          y         1.27
2                          f             NaN          n         6.18
3                          f               s          n         4.98
4                          f               p          y         6.73
…                        …             …         …          …
2077959                    f               a          w         2.67
2077960                    f               d          w         2.69
2077961                    f               a          w         6.16
2077962                    f               a          g         6.00
2077963                    f               d          y         2.69

         stem-width stem-color veil-color has-ring ring-type habitat season
0             17.12          w          w        t         g       d      a
1             10.75          n        NaN        f         f       d      a
2              3.14          n        NaN        f         f       d      s
3              8.51          w          n        t         z       d      u
4             13.70          y          y        t       NaN       d      u
…               …        …        …      …       …     …    …
2077959        1.35          e        NaN        f         f       d      u
2077960        7.38          w        NaN        f         f       g      a
2077961        9.74          y          w        t         z       d      a
2077962        3.46          g        NaN        f         f       d      a
2077963       17.71          w        NaN        f         f       d      w

[2077964 rows x 16 columns]
```

[33]:
```
mode_value11 = test_data_new['cap-shape'].mode()[0]


test_data_new['cap-shape'].fillna(mode_value11, inplace=True)
```

```
/tmp/ipykernel_17/2696952803.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
```

```
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['cap-shape'].fillna(mode_value11, inplace=True)
```

[34]:
```python
mode_value12= test_data_new['cap-surface'].mode()[0]


test_data_new['cap-surface'].fillna(mode_value12, inplace=True)
```

```
/tmp/ipykernel_17/2816731538.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['cap-surface'].fillna(mode_value12, inplace=True)
```

[35]:
```python
mode_value13 =test_data_new['cap-color'].mode()[0]

test_data_new['cap-color'].fillna(mode_value13, inplace=True)
```

```
/tmp/ipykernel_17/3596592312.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['cap-color'].fillna(mode_value13, inplace=True)
```

[36]:
```python
mode_value14 = test_data_new['ring-type'].mode()[0]


test_data_new['ring-type'].fillna(mode_value14, inplace=True)
```

```
/tmp/ipykernel_17/431551427.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['ring-type'].fillna(mode_value14, inplace=True)
```

[37]:
```python
mode_value4 = test_data_new['gill-attachment'].mode()[0]



test_data_new['gill-attachment'].fillna(mode_value4, inplace=True)
```

```
/tmp/ipykernel_17/1680588234.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['gill-attachment'].fillna(mode_value4, inplace=True)
```

[38]:
```python
mode_value15 = test_data_new['habitat'].mode()[0]

test_data_new['habitat'].fillna(mode_value15, inplace=True)
```

```
/tmp/ipykernel_17/2151780051.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  test_data_new['habitat'].fillna(mode_value15, inplace=True)
```

[39]:
```python
mode_value16 = test_data_new['does-bruise-or-bleed'].mode()[0]

test_data_new['does-bruise-or-bleed'].fillna(mode_value16, inplace=True)
```

```
/tmp/ipykernel_17/3838921477.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['does-bruise-or-bleed'].fillna(mode_value16, inplace=True)
```

[40]:
```python
mode_value17 = test_data_new['stem-color'].mode()[0]

test_data_new['stem-color'].fillna(mode_value17, inplace=True)
```

```
/tmp/ipykernel_17/1546258827.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['stem-color'].fillna(mode_value17, inplace=True)
```

[41]:
```python
median_value_new1 = test_data_new['cap-diameter'].median()



test_data_new['cap-diameter'].fillna(median_value_new1, inplace=True)
```

```
/tmp/ipykernel_17/3621429776.py:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
```

```
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['cap-diameter'].fillna(median_value_new1, inplace=True)
```

[42]:
```python
median_value_new1 = test_data_new['stem-height'].median()

test_data_new['stem-height'].fillna(median_value_new1, inplace=True)
```

```
/tmp/ipykernel_17/1185175250.py:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_data_new['stem-height'].fillna(median_value_new1, inplace=True)
```

[43]:
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming your data is in a DataFrame named 'data'

# List of categorical columns
categorical_columns_new = [
     'cap-shape', 'cap-surface', 'cap-color', 'does-bruise-or-bleed',
    'gill-attachment', 'gill-color', 'stem-color', 'has-ring',
    'ring-type', 'habitat', 'season',"veil-color"
]

#  Label Encoding
# This will convert categories to integers
label_encoders = {}
for column in categorical_columns_new:
    le = LabelEncoder()
    test_data_new[column] = le.fit_transform(test_data_new[column])
    label_encoders[column] = le
```

[44]:

```
test_data_new
```

[44]:

	id	cap-diameter	cap-shape	cap-surface	cap-color	\
0	3116945	8.64	59	53	44	
1	3116946	6.90	50	53	45	
2	3116947	2.00	36	38	44	
3	3116948	3.47	59	53	44	
4	3116949	6.17	59	39	55	
…	…	…	…	…	…	
2077959	5194904	0.88	59	38	53	
2077960	5194905	3.12	59	50	53	
2077961	5194906	5.73	59	36	34	
2077962	5194907	5.03	36	38	44	
2077963	5194908	15.51	41	53	53	

	does-bruise-or-bleed	gill-attachment	gill-color	stem-height	\
0	18	37	52	11.13	
1	5	37	54	1.27	
2	5	37	41	6.18	
3	5	57	41	4.98	
4	5	55	54	6.73	
…	…	…	…	…	
2077959	5	37	52	2.67	
2077960	5	41	52	2.69	
2077961	5	37	52	6.16	
2077962	5	37	31	6.00	
2077963	5	41	54	2.69	

	stem-width	stem-color	veil-color	has-ring	ring-type	habitat	\
0	17.12	51	21	17	15	16	
1	10.75	38	23	6	14	16	
2	3.14	38	23	6	14	16	
3	8.51	51	14	17	35	16	
4	13.70	53	22	17	14	16	
…	…	…	…	…	…	…	
2077959	1.35	29	23	6	14	16	
2077960	7.38	51	23	6	14	19	
2077961	9.74	53	21	17	35	16	
2077962	3.46	31	23	6	14	16	
2077963	17.71	51	23	6	14	16	

	season
0	0
1	0
2	1
3	2
4	2

```
…            …
2077959      2
2077960      0
2077961      0
2077962      0
2077963      3

[2077964 rows x 16 columns]
```

[45]:
```
test_data_new.isnull().sum()
```

[45]:
```
id                     0
cap-diameter           0
cap-shape              0
cap-surface            0
cap-color              0
does-bruise-or-bleed   0
gill-attachment        0
gill-color             0
stem-height            0
stem-width             0
stem-color             0
veil-color             0
has-ring               0
ring-type              0
habitat                0
season                 0
dtype: int64
```

0.0.14 Making Predictions on New Test Data and Creating a Submission File

After achieving good accuracy on the training and test data using the Random Forest model, I applied the model to new test data to make predictions. These predictions are then formatted for submission.

[46]:
```
X_new = test_data_new.drop(columns=['id'])  # Drop the 'id' column to create␣
 ↪the input features

# Make predictions on the new test data
y_test_pred = random_forest.predict(X_new)

# Map predictions to 'p' for 1 and 'e' for 0
class_map = {0: 'e', 1: 'p'}
y_test_pred_mapped = [class_map[pred] for pred in y_test_pred]


submission = pd.DataFrame({
    'id': test_data_new['id'],  # Include the 'id' column from the test data
```

```
    'class': y_test_pred_mapped
})

# Save the submission file
submission.to_csv('/kaggle/working/submission.csv', index=False)

print('Submission file created: /kaggle/working/submission.csv')
```

```
Submission file created: /kaggle/working/submission.csv
```