CSM148_Project3

March 4, 2023

You are exploring the wilderness of *Mushroomia*, a land populated by a plethora of diverse fauna and flora. In particular, *Mushroomia* is known for its unparalleled variety in mushrooms. However, not all the mushrooms in *Mushroomia* are edible. As you make your way through *Mushroomia*, you would like to know which mushrooms are edible, in order to forage for supplies for your daily mushroom soup.

You have access to: * *Shroomster Pro Max TM* - a state of the art data collection device, developed by *Mushroomia*, that allows you to collect various data points about any mushroom you encounter in the wild * *The National Archives on Mushrooms* - a dataset collected over the years by the government of *Mushroomia*

To address this problem, you decide to use the skills you learnt in CSM148 and train machine learning models on the *The National Archives on Mushrooms* in order to use your *Shroomster Pro Max TM* to determine whether the mushrooms you encounter on your adventure can be added to your daily mushroom soup.

This project will be more unstructured than the previous two projects in order to allow you to experience how data science problems are solved in practice. There are two parts to this project: a Jupyter Notebook with your code (where you explore, visualize, process your data and train machine learning models) and a report (where you explain the various choices you make in your implementation and analyze the final performance of your models).

1 1. Loading and Viewing Data

[ ]:
```
# import packages
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph
import os
import seaborn as sns # used for plot interactive graph.
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
```

```
import sklearn.metrics.cluster as smc
from sklearn.model_selection import KFold

from matplotlib import pyplot
import itertools

%matplotlib inline
```

[ ]:
```
# load training dataset
train_df = pd.read_csv("mushroom_train.csv", sep = ";")
train_df.head()
```

[ ]:
```
  class  cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0     p         15.26         x           g         o                    f
1     p         16.60         x           g         o                    f
2     p         14.07         x           g         o                    f
3     p         14.17         f           h         e                    f
4     p         14.64         x           h         o                    f

  gill-attachment gill-spacing gill-color  stem-height  …  stem-root  \
0               e          NaN          w        16.95  …          s
1               e          NaN          w        17.99  …          s
2               e          NaN          w        17.80  …          s
3               e          NaN          w        15.77  …          s
4               e          NaN          w        16.53  …          s

  stem-surface stem-color veil-type veil-color has-ring ring-type  \
0            y          w         u          w        t         g
1            y          w         u          w        t         g
2            y          w         u          w        t         g
3            y          w         u          w        t         p
4            y          w         u          w        t         p

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

[5 rows x 21 columns]
```

[ ]:
```
# load testing dataset
test_df = pd.read_csv("mushroom_test.csv", sep = ";")
test_df.head()
```

[ ]:
```
  class  cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0     p          2.50         b         NaN         k                    f
1     p          3.07         b         NaN         k                    f
2     p          3.30         b         NaN         n                    f
3     p          3.49         b         NaN         k                    f
4     p          2.79         b         NaN         n                    f

  gill-attachment gill-spacing gill-color  stem-height  …  stem-root  \
0               a          NaN          k         8.42  …        NaN
1               a          NaN          n         7.24  …        NaN
2               a          NaN          n        10.22  …        NaN
3               a          NaN          k        11.00  …        NaN
4               a          NaN          n         6.97  …        NaN

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

  spore-print-color habitat season
0                 k       g      u
1                 k       g      a
2                 k       g      u
3                 k       g      a
4                 k       g      u

[5 rows x 21 columns]
```

2 2. Splitting Data into Features and Labels

[ ]:
```
# drop labels from training dataset
train_labels = train_df["class"].copy()
train_df = train_df.drop("class", axis=1)
train_df.head()
```

[ ]:
```
   cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0         15.26         x           g         o                    f
1         16.60         x           g         o                    f
2         14.07         x           g         o                    f
3         14.17         f           h         e                    f
4         14.64         x           h         o                    f

  gill-attachment gill-spacing gill-color  stem-height  stem-width stem-root  \
0               e          NaN          w        16.95       17.09         s
```

```
1                e          NaN          w        17.99       18.19         s
2                e          NaN          w        17.80       17.74         s
3                e          NaN          w        15.77       15.98         s
4                e          NaN          w        16.53       17.20         s

  stem-surface stem-color veil-type veil-color has-ring ring-type  \
0            y          w         u          w        t         g
1            y          w         u          w        t         g
2            y          w         u          w        t         g
3            y          w         u          w        t         p
4            y          w         u          w        t         p

  spore-print-color habitat season
0               NaN       d      w
1               NaN       d      u
2               NaN       d      w
3               NaN       d      w
4               NaN       d      w
```

[ ]:
```
# convert the "class" column into labels: p (poisonous) -> 0, e (edible) -> 1
train_labels[train_labels == 'p'] = 0
train_labels[train_labels == 'e'] = 1
```

[ ]:
```
train_labels.value_counts()
```

[ ]:
```
0    29242
1    20971
Name: class, dtype: int64
```

[ ]:
```
train_labels = pd.Series(train_labels, dtype = "int32")
```

[ ]:
```
# drop labels from testing dataset
test_labels = test_df["class"].copy()
test_df = test_df.drop("class", axis=1)
test_df.head()
```

[ ]:
```
   cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0          2.50         b         NaN         k                    f
1          3.07         b         NaN         k                    f
2          3.30         b         NaN         n                    f
3          3.49         b         NaN         k                    f
4          2.79         b         NaN         n                    f

  gill-attachment gill-spacing gill-color  stem-height  stem-width stem-root  \
0               a          NaN          k         8.42        2.46       NaN
1               a          NaN          n         7.24        2.41       NaN
2               a          NaN          n        10.22        2.53       NaN
```

```
3              a          NaN          k        11.00        2.81       NaN
4              a          NaN          n         6.97        2.37       NaN

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

  spore-print-color habitat season
0                 k       g      u
1                 k       g      a
2                 k       g      u
3                 k       g      a
4                 k       g      u
```

[ ]:
```
# convert the "class" column into labels: p (poisonous) -> 0, e (edible) -> 1
test_labels[test_labels == 'p'] = 0
test_labels[test_labels == 'e'] = 1
```

[ ]:
```
test_labels.value_counts()
```

[ ]:
```
1    6210
0    4646
Name: class, dtype: int64
```

[ ]:
```
test_labels = pd.Series(test_labels, dtype = "int32")
```

3 3. Data Processing, Exploration, Visualization, and Augmentation

[ ]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50213 entries, 0 to 50212
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   cap-diameter          50213 non-null  float64
 1   cap-shape             50213 non-null  object
 2   cap-surface           37915 non-null  object
 3   cap-color             50213 non-null  object
 4   does-bruise-or-bleed  50213 non-null  object
 5   gill-attachment       42447 non-null  object
 6   gill-spacing          31064 non-null  object
 7   gill-color            50213 non-null  object
```

```
 8   stem-height           50213 non-null  float64
 9   stem-width            50213 non-null  float64
 10  stem-root             7413 non-null   object
 11  stem-surface          19912 non-null  object
 12  stem-color            50213 non-null  object
 13  veil-type             3177 non-null   object
 14  veil-color            6297 non-null   object
 15  has-ring              50213 non-null  object
 16  ring-type             48448 non-null  object
 17  spore-print-color     4532 non-null   object
 18  habitat               50213 non-null  object
 19  season                50213 non-null  object
dtypes: float64(3), object(17)
memory usage: 7.7+ MB
```

[ ]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10856 entries, 0 to 10855
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   cap-diameter          10856 non-null  float64
 1   cap-shape             10856 non-null  object
 2   cap-surface           9034 non-null   object
 3   cap-color             10856 non-null  object
 4   does-bruise-or-bleed  10856 non-null  object
 5   gill-attachment       8738 non-null   object
 6   gill-spacing          4942 non-null   object
 7   gill-color            10856 non-null  object
 8   stem-height           10856 non-null  float64
 9   stem-width            10856 non-null  float64
 10  stem-root             2118 non-null   object
 11  stem-surface          3033 non-null   object
 12  stem-color            10856 non-null  object
 13  veil-type             0 non-null      float64
 14  veil-color            1116 non-null   object
 15  has-ring              10856 non-null  object
 16  ring-type             10150 non-null  object
 17  spore-print-color     1822 non-null   object
 18  habitat               10856 non-null  object
 19  season                10856 non-null  object
dtypes: float64(4), object(16)
memory usage: 1.7+ MB
```

3.0.1 Initial Data Processing (Impute Nulls)

[ ]:
```
# check for nulls in the entire dataset
nulls = train_df[train_df.isnull().any(axis=1)]
nulls
```

[ ]:
```
       cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0             15.26         x           g         o                    f
1             16.60         x           g         o                    f
2             14.07         x           g         o                    f
3             14.17         f           h         e                    f
4             14.64         x           h         o                    f
…               …       …         …       …                  …
50208          1.18         s           s         y                    f
50209          1.27         f           s         y                    f
50210          1.27         s           s         y                    f
50211          1.24         f           s         y                    f
50212          1.17         s           s         y                    f

      gill-attachment gill-spacing gill-color  stem-height  stem-width  \
0                   e          NaN          w        16.95       17.09
1                   e          NaN          w        17.99       18.19
2                   e          NaN          w        17.80       17.74
3                   e          NaN          w        15.77       15.98
4                   e          NaN          w        16.53       17.20
…                 …          …         …          …         …
50208               f            f          f         3.93        6.22
50209               f            f          f         3.18        5.43
50210               f            f          f         3.86        6.37
50211               f            f          f         3.56        5.44
50212               f            f          f         3.25        5.45

      stem-root stem-surface stem-color veil-type veil-color has-ring  \
0             s            y          w         u          w        t
1             s            y          w         u          w        t
2             s            y          w         u          w        t
3             s            y          w         u          w        t
4             s            y          w         u          w        t
…         …          …        …         …        …
50208       NaN          NaN          y       NaN        NaN        f
50209       NaN          NaN          y       NaN        NaN        f
50210       NaN          NaN          y       NaN        NaN        f
50211       NaN          NaN          y       NaN        NaN        f
50212       NaN          NaN          y       NaN        NaN        f

      ring-type spore-print-color habitat season
0             g               NaN       d      w
```

```
1              g                NaN       d       u
2              g                NaN       d       w
3              p                NaN       d       w
4              p                NaN       d       w
…          …                    …     …     …
50208          f                NaN       d       a
50209          f                NaN       d       a
50210          f                NaN       d       u
50211          f                NaN       d       u
50212          f                NaN       d       u

[50213 rows x 20 columns]
```

[ ]:
```
# remove columns where the number of non-null observations is small
train_df = train_df.drop(["stem-root", "stem-surface", "veil-type",␣
 ↪"veil-color", "spore-print-color"], axis = 1)
train_df
```

[ ]:
```
       cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0             15.26         x           g         o                    f
1             16.60         x           g         o                    f
2             14.07         x           g         o                    f
3             14.17         f           h         e                    f
4             14.64         x           h         o                    f
…               …       …           …         …                  …
50208          1.18         s           s         y                    f
50209          1.27         f           s         y                    f
50210          1.27         s           s         y                    f
50211          1.24         f           s         y                    f
50212          1.17         s           s         y                    f

      gill-attachment gill-spacing gill-color  stem-height  stem-width  \
0                   e          NaN          w        16.95       17.09
1                   e          NaN          w        17.99       18.19
2                   e          NaN          w        17.80       17.74
3                   e          NaN          w        15.77       15.98
4                   e          NaN          w        16.53       17.20
…                 …          …        …          …         …
50208               f            f          f         3.93        6.22
50209               f            f          f         3.18        5.43
50210               f            f          f         3.86        6.37
50211               f            f          f         3.56        5.44
50212               f            f          f         3.25        5.45

      stem-color has-ring ring-type habitat season
0              w        t         g       d      w
1              w        t         g       d      u
```

```
2              w         t          g       d      w
3              w         t          p       d      w
4              w         t          p       d      w
…            …       …        …      …     …
50208          y         f          f       d      a
50209          y         f          f       d      a
50210          y         f          f       d      u
50211          y         f          f       d      u
50212          y         f          f       d      u

[50213 rows x 15 columns]
```

[ ]:
```
# remove same columns from testing dataset
test_df = test_df.drop(["stem-root", "stem-surface", "veil-type", "veil-color",␣
↪"spore-print-color"], axis = 1)
test_df
```

[ ]:
```
       cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0              2.50         b         NaN         k                    f
1              3.07         b         NaN         k                    f
2              3.30         b         NaN         n                    f
3              3.49         b         NaN         k                    f
4              2.79         b         NaN         n                    f
…               …       …         …       …                  …
10851         52.41         o           y         y                    f
10852         54.81         o           y         y                    f
10853         49.95         o           y         y                    f
10854         53.16         o           y         y                    f
10855         49.78         o           y         y                    f

      gill-attachment gill-spacing gill-color  stem-height  stem-width  \
0                   a          NaN          k         8.42        2.46
1                   a          NaN          n         7.24        2.41
2                   a          NaN          n        10.22        2.53
3                   a          NaN          k        11.00        2.81
4                   a          NaN          n         6.97        2.37
…                 …          …        …          …         …
10851               p          NaN          y         5.47       25.02
10852               p          NaN          y         6.67       22.15
10853               p          NaN          y         6.43       26.35
10854               p          NaN          y         6.99       40.29
10855               p          NaN          y         5.77       18.26

      stem-color has-ring ring-type habitat season
0              g        f         f       g      u
1              n        f         f       g      a
2              n        f         f       g      u
```

```
3              n         f          f        g       a
4              g         f          f        g       u
…            …         …          …        …       …
10851          k         f          f        d       u
10852          k         f          f        d       s
10853          n         f          f        d       u
10854          k         f          f        d       s
10855          k         f          f        d       u

[10856 rows x 15 columns]
```

[ ]:
```
# use mode for other columns with null observations
train_df["cap-surface"] = train_df["cap-surface"].
 ↪fillna(train_df["cap-surface"].mode()[0])
train_df["gill-attachment"] = train_df["gill-attachment"].
 ↪fillna(train_df["gill-attachment"].mode()[0])
train_df["gill-spacing"] = train_df["gill-spacing"].
 ↪fillna(train_df["gill-spacing"].mode()[0])
# train_df["stem-surface"] = train_df["stem-surface"].
 ↪fillna(train_df["stem-surface"].mode()[0])
train_df["ring-type"] = train_df["ring-type"].fillna(train_df["ring-type"].
 ↪mode()[0])
```

[ ]:
```
# use mode from training dataset for other columns with null observations
test_df["cap-surface"] = test_df["cap-surface"].fillna(train_df["cap-surface"].
 ↪mode()[0])
test_df["gill-attachment"] = test_df["gill-attachment"].
 ↪fillna(train_df["gill-attachment"].mode()[0])
test_df["gill-spacing"] = test_df["gill-spacing"].
 ↪fillna(train_df["gill-spacing"].mode()[0])
# test_df["stem-surface"] = test_df["stem-surface"].
 ↪fillna(train_df["stem-surface"].mode()[0])
test_df["ring-type"] = test_df["ring-type"].fillna(train_df["ring-type"].
 ↪mode()[0])
```

[ ]:
```
# check to make sure all nulls have been removed
train_df[train_df.isnull().any(axis=1)]
```

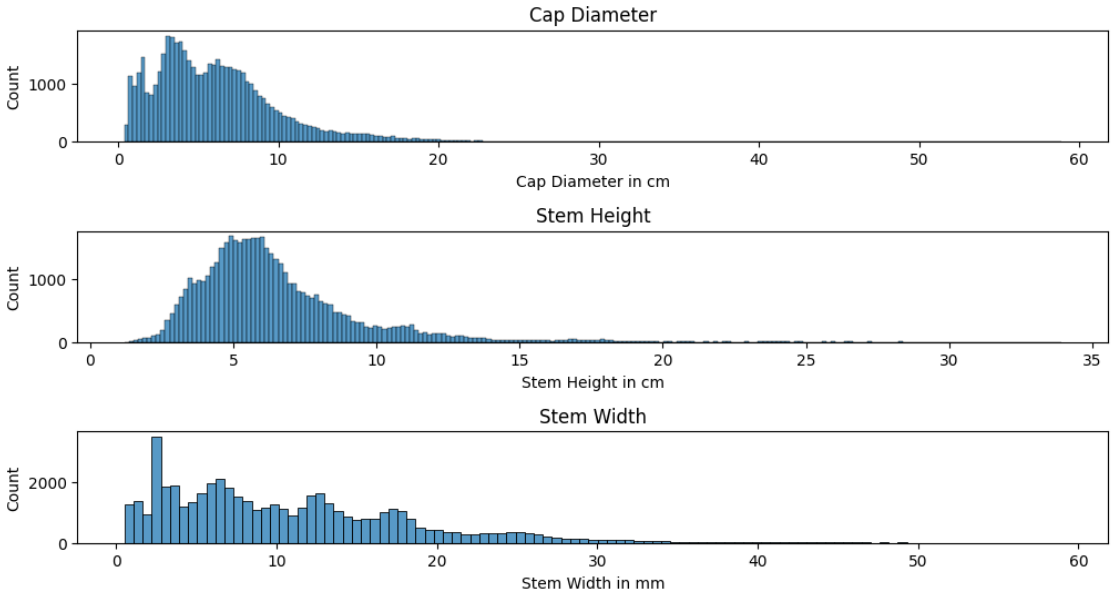
[ ]:
```
Empty DataFrame
Columns: [cap-diameter, cap-shape, cap-surface, cap-color, does-bruise-or-bleed,
gill-attachment, gill-spacing, gill-color, stem-height, stem-width, stem-color,
has-ring, ring-type, habitat, season]
Index: []
```

[ ]:
```
test_df[test_df.isnull().any(axis=1)]
```

[ ]:
```
Empty DataFrame
Columns: [cap-diameter, cap-shape, cap-surface, cap-color, does-bruise-or-bleed,
gill-attachment, gill-spacing, gill-color, stem-height, stem-width, stem-color,
has-ring, ring-type, habitat, season]
Index: []
```

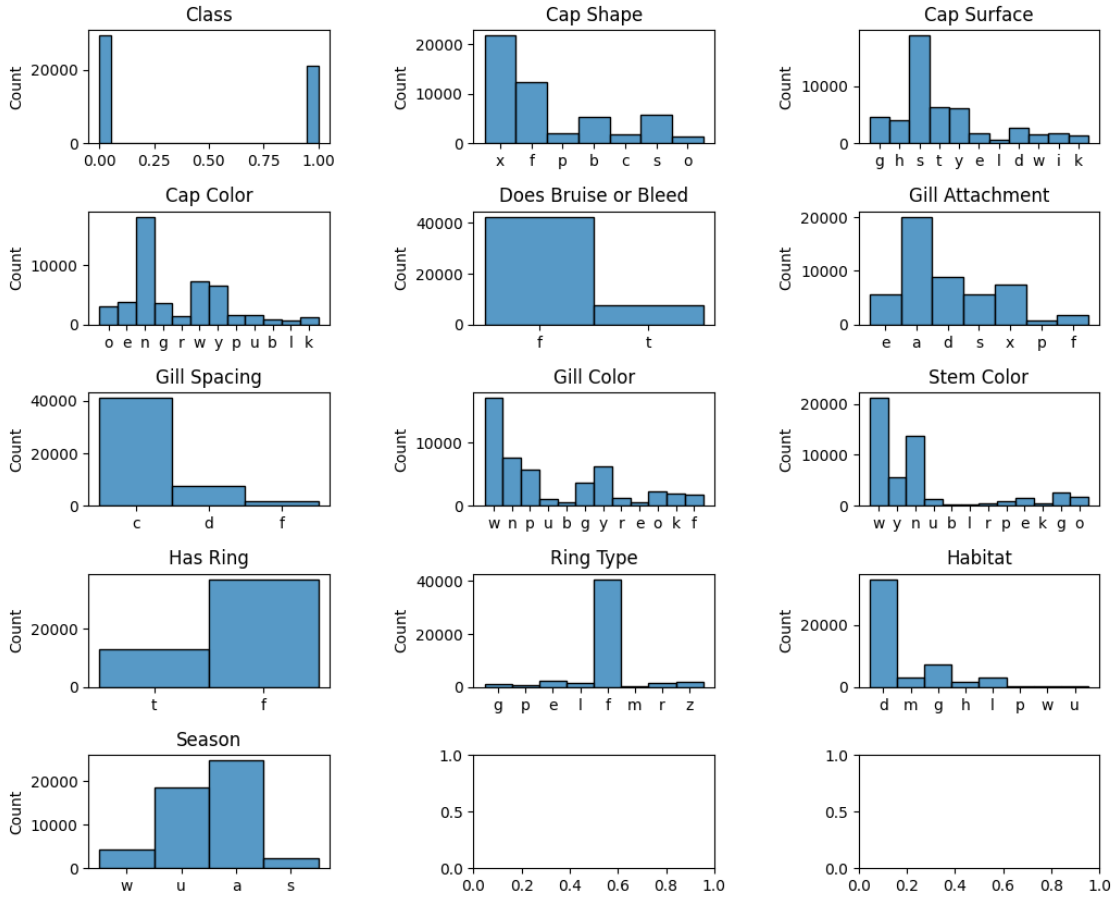
3.0.2 Data Visualization

[ ]:
```
# plot numerical features
fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12,6))
sns.histplot(train_df["cap-diameter"], ax = ax1)
ax1.set_title("Cap Diameter")
ax1.set_xlabel("Cap Diameter in cm")
sns.histplot(train_df["stem-height"], ax = ax2)
ax2.set_title("Stem Height")
ax2.set_xlabel("Stem Height in cm")
sns.histplot(train_df["stem-width"], ax = ax3)
ax3.set_title("Stem Width")
ax3.set_xlabel("Stem Width in mm")
plt.subplots_adjust(
                    wspace=0.8,
                    hspace=0.8)
```



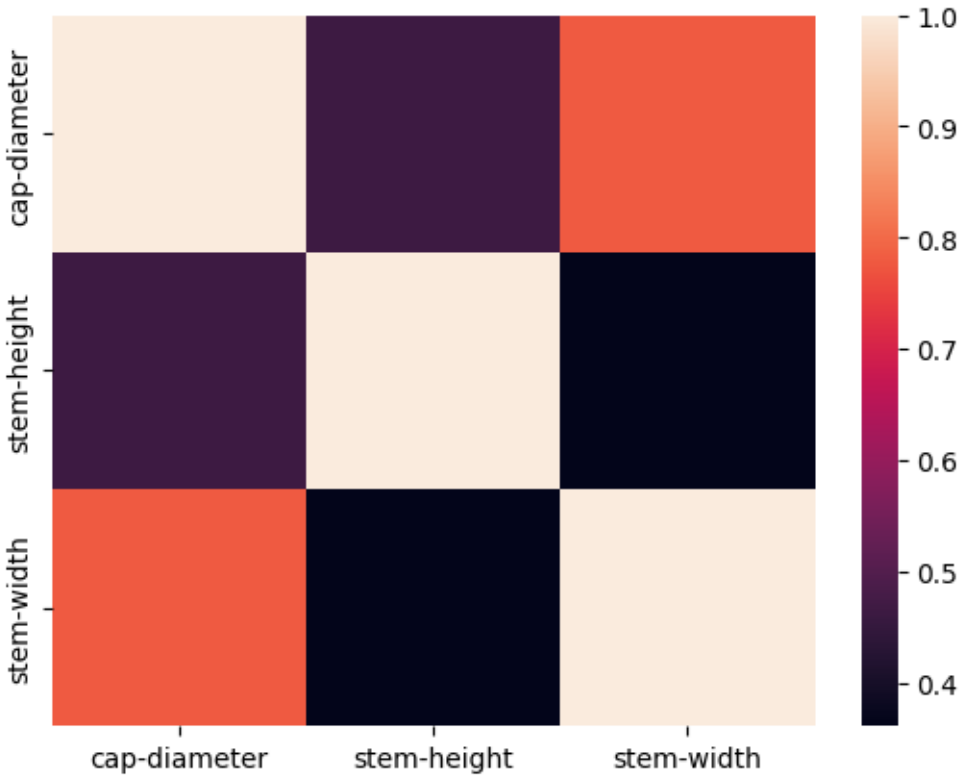
[ ]:
```
# plot categorical features
fig, ax = plt.subplots(5, 3, figsize=(12,10))
sns.histplot(train_labels, ax = ax[0][0]).set(xlabel=None)
ax[0][0].set_title("Class")
```

```
sns.histplot(train_df["cap-shape"], ax = ax[0][1]).set(xlabel=None)
ax[0][1].set_title("Cap Shape")
sns.histplot(train_df["cap-surface"], ax = ax[0][2]).set(xlabel=None)
ax[0][2].set_title("Cap Surface")
sns.histplot(train_df["cap-color"], ax = ax[1][0]).set(xlabel=None)
ax[1][0].set_title("Cap Color")
sns.histplot(train_df["does-bruise-or-bleed"], ax = ax[1][1]).set(xlabel=None)
ax[1][1].set_title("Does Bruise or Bleed")
sns.histplot(train_df["gill-attachment"], ax = ax[1][2]).set(xlabel=None)
ax[1][2].set_title("Gill Attachment")
sns.histplot(train_df["gill-spacing"], ax = ax[2][0]).set(xlabel=None)
ax[2][0].set_title("Gill Spacing")
sns.histplot(train_df["gill-color"], ax = ax[2][1]).set(xlabel=None)
ax[2][1].set_title("Gill Color")
# sns.histplot(train_df["stem-surface"], ax = ax[2][2]).set(xlabel=None)
# ax[2][2].set_title("Stem Surface")
sns.histplot(train_df["stem-color"], ax = ax[2][2]).set(xlabel=None)
ax[2][2].set_title("Stem Color")
sns.histplot(train_df["has-ring"], ax = ax[3][0]).set(xlabel=None)
ax[3][0].set_title("Has Ring")
sns.histplot(train_df["ring-type"], ax = ax[3][1]).set(xlabel=None)
ax[3][1].set_title("Ring Type")
sns.histplot(train_df["habitat"], ax = ax[3][2]).set(xlabel=None)
ax[3][2].set_title("Habitat")
sns.histplot(train_df["season"], ax = ax[4][0]).set(xlabel=None)
ax[4][0].set_title("Season")
plt.subplots_adjust(
                    wspace=0.6,
                    hspace=0.6)
```



[ ]:
```
# correlation heatmap
sns.heatmap(train_df.corr())
```

[ ]: <Axes: >



[ ]:
```
train_df.head()
```

[ ]:
```
   cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0         15.26         x           g         o                    f
1         16.60         x           g         o                    f
2         14.07         x           g         o                    f
3         14.17         f           h         e                    f
4         14.64         x           h         o                    f

  gill-attachment gill-spacing gill-color  stem-height  stem-width stem-color  \
0               e            c          w        16.95       17.09          w
1               e            c          w        17.99       18.19          w
2               e            c          w        17.80       17.74          w
3               e            c          w        15.77       15.98          w
4               e            c          w        16.53       17.20          w

  has-ring ring-type habitat season
0        t         g       d      w
1        t         g       d      u
2        t         g       d      w
3        t         p       d      w
4        t         p       d      w
```

3.0.3 Data Augmentation

[ ]:
```
train_df["stem-area"] = train_df["stem-height"] * train_df["stem-width"]
test_df["stem-area"] = test_df["stem-height"] * test_df["stem-width"]
```

[ ]:
```
# # fill NAs caused by dividing by 0
# train_df["stem-area"].fillna(0, inplace=True)
# test_df["stem-area"].fillna(0, inplace=True)
```

[ ]:
```
# train_df["cap-circumference"] = train_df["cap-diameter"] * np.pi
# test_df["cap-circumference"] = test_df["cap-diameter"] * np.pi
```

[ ]:
```
train_df["cap-diameter-cat"] = pd.cut(train_df["cap-diameter"],
                                      bins=[0, 2, 4, 6, 8., np.inf],
                                      labels=[1, 2, 3, 4, 5])

train_df["cap-diameter-cat"].value_counts()
```

[ ]:
```
5    12943
2    11312
3    10069
4     9817
1     6072
Name: cap-diameter-cat, dtype: int64
```

[ ]:
```
test_df["cap-diameter-cat"] = pd.cut(test_df["cap-diameter"],
                                     bins=[0, 2, 4, 6, 8., np.inf],
                                     labels=[1, 2, 3, 4, 5])

test_df["cap-diameter-cat"].value_counts()
```

[ ]:
```
5    4972
4    2002
3    1908
2    1620
1     354
Name: cap-diameter-cat, dtype: int64
```

[ ]:
```
train_df.head()
```

[ ]:
```
   cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0         15.26         x           g         o                    f
1         16.60         x           g         o                    f
2         14.07         x           g         o                    f
3         14.17         f           h         e                    f
4         14.64         x           h         o                    f

  gill-attachment gill-spacing gill-color  stem-height  stem-width stem-color  \
```

0	e	c	w	16.95	17.09	w
1	e	c	w	17.99	18.19	w
2	e	c	w	17.80	17.74	w
3	e	c	w	15.77	15.98	w
4	e	c	w	16.53	17.20	w

	has-ring	ring-type	habitat	season	stem-area	cap-diameter-cat
0	t	g	d	w	289.6755	5
1	t	g	d	u	327.2381	5
2	t	g	d	w	315.7720	5
3	t	p	d	w	252.0046	5
4	t	p	d	w	284.3160	5

3.0.4 More Data Processing (Pipelining)

[ ]:
```
# Hints:
# 1. Convert the "class" column into labels: 'p' (poisonous) -> 0,  'e'␣
 ↪(edible) -> 1
# 2. You can drop columns if you see fit
# 3. See any imcomplete data? We learned how to deal with them in project 1.

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer

# do processing
data_num = train_df[["cap-diameter", "stem-height", "stem-width", "stem-area"]]

num_pipeline = Pipeline([
    ("std_scaler", StandardScaler())
])

data_num_train = num_pipeline.fit_transform(data_num)
numerical_features = list(data_num)
# "stem-root", "veil-type", "veil-color", "spore-print-color", "stem-surface",
categorical_features = ["cap-shape", "cap-surface", "cap-color",␣
 ↪"does-bruise-or-bleed", "gill-attachment", "gill-spacing", "gill-color",
                        "stem-color", "has-ring", "ring-type", "habitat",␣
 ↪"season", "cap-diameter-cat"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(categories='auto', handle_unknown='ignore'),␣
 ↪categorical_features),
    ])

prepared_train = full_pipeline.fit_transform(train_df).toarray()
```

[ ]:
```
# Pipeline my test data
prepared_test = full_pipeline.transform(test_df).toarray()
```

4 4. Logistic Regression & Statistical Hypothesis Testing

[ ]:
```
# run logistic regression model
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver = "liblinear")
log_reg.fit(prepared_train, train_labels)
```

[ ]:
```
LogisticRegression(solver='liblinear')
```

[ ]:
```
# predict on test data and print metrics
y_pred = log_reg.predict(prepared_test)
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred))
print("Precision:",metrics.precision_score(test_labels, y_pred))
print("Recall:",metrics.recall_score(test_labels, y_pred))
print("F1 Score:",metrics.f1_score(test_labels, y_pred))
```

```
Accuracy: 0.5048820928518791
Precision: 0.7572396796056685
Recall: 0.19790660225442835
F1 Score: 0.3138005872590323
```

[ ]:
```
# # repipeline data for statistical hypothesis testing
# train_df_numerical = train_df[["cap-diameter", "stem-height", "stem-width",␣
 ↪"stem-area"]]

# num_pipeline = Pipeline([
#     ("std_scaler", StandardScaler())
# ])

# data_num_train = num_pipeline.fit_transform(train_df_numerical)
# numerical_features = list(train_df_numerical)
# # "stem-root", "veil-type", "veil-color", "spore-print-color",␣
 ↪"stem-surface",
# # categorical_features = ["cap-shape", "cap-surface", "cap-color",␣
 ↪"does-bruise-or-bleed", "gill-attachment", "gill-spacing", "gill-color",
# #                         "stem-color", "has-ring", "ring-type", "habitat",␣
 ↪"season"]

# full_pipeline = ColumnTransformer([
#         ("num", num_pipeline, numerical_features)
#         # ("cat", OneHotEncoder(categories='auto', handle_unknown='ignore'),␣
 ↪categorical_features),
#     ])

# prepared_train_numerical = full_pipeline.fit_transform(train_df_numerical)
```

[ ]:
```
train_df_numerical = train_df[["cap-diameter", "stem-height", "stem-width",␣
 ↪"stem-area"]]
prepared_train_numerical = num_pipeline.fit_transform(train_df_numerical)
```

[ ]:
```
prepared_train_numerical.shape
```

[ ]:
```
(50213, 4)
```

[ ]:
```
import statsmodels.api as sm
from statsmodels.genmod.generalized_linear_model import GLM
```

```
from statsmodels.genmod import families

# GLM Model
x = sm.add_constant(prepared_train_numerical)
res = sm.GLM(train_labels, x, family = families.Binomial()).fit()
print(res.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                  class   No. Observations:                50213
Model:                            GLM   Df Residuals:                    50208
Model Family:                Binomial   Df Model:                            4
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -33387.
Date:                Sat, 04 Mar 2023   Deviance:                       66774.
Time:                        16:24:15   Pearson chi2:                 5.01e+04
No. Iterations:                     4   Pseudo R-squ. (CS):            0.02880
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3330      0.009    -36.238      0.000      -0.351      -0.315
x1             0.4387      0.020     21.520      0.000       0.399       0.479
x2             0.0531      0.025      2.133      0.033       0.004       0.102
x3             0.0230      0.028      0.832      0.405      -0.031       0.077
x4            -0.1266      0.038     -3.316      0.001      -0.201      -0.052
==============================================================================
```

5 5. Dimensionality Reduction using PCA

[ ]:
```
# PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.
↪PCA.html
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca_train = pca.fit_transform(prepared_train)
```

[ ]:
```
# Logistic Regression
log_reg2 = LogisticRegression(solver='liblinear')
log_reg2.fit(pca_train, train_labels)
```

[ ]:
```
LogisticRegression(solver='liblinear')
```

[ ]:
```
# PCA our Test data
pca_test = pca.transform(prepared_test)
```

[ ]:
```
y_pred2 = log_reg2.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred2))
```

```
print("Precision:",metrics.precision_score(test_labels, y_pred2))
print("Recall:",metrics.recall_score(test_labels, y_pred2))
print("F1 Score:",metrics.f1_score(test_labels, y_pred2))
```

```
Accuracy: 0.5273581429624171
Precision: 0.6331112755983223
Recall: 0.41320450885668275
F1 Score: 0.5000487186982364
```

6 6. Experiment with any 2 other models (Non-Ensemble)

[ ]:
```
# Models: https://scikit-learn.org/stable/supervised_learning.html
# run knn model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(pca_train, train_labels)
```

[ ]:
```
KNeighborsClassifier(n_neighbors=20)
```

[ ]:
```
y_pred3 = knn.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred3))
print("Precision:",metrics.precision_score(test_labels, y_pred3))
print("Recall:",metrics.recall_score(test_labels, y_pred3))
print("F1 Score:",metrics.f1_score(test_labels, y_pred3))
```

```
Accuracy: 0.5878776713338246
Precision: 0.6911052399823866
Recall: 0.505475040257649
F1 Score: 0.5838913690476191
```

[ ]:
```
# run decision tree model
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(pca_train, train_labels)
```

[ ]:
```
DecisionTreeClassifier()
```

[ ]:
```
y_pred4 = clf.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred4))
print("Precision:",metrics.precision_score(test_labels, y_pred4))
print("Recall:",metrics.recall_score(test_labels, y_pred4))
print("F1 Score:",metrics.f1_score(test_labels, y_pred4))
```

```
Accuracy: 0.5737840825350037
Precision: 0.6596087920951805
Recall: 0.5267310789049919
F1 Score: 0.5857283552690482
```

7 7. Experiment with 1 Ensemble Method

[ ]:
```
# run adaboost classifier model
# Ensemble Methods: https://scikit-learn.org/stable/modules/ensemble.html
from sklearn.ensemble import AdaBoostClassifier
clf2 = AdaBoostClassifier(n_estimators=100)
clf2.fit(pca_train, train_labels)
```

[ ]:
```
AdaBoostClassifier(n_estimators=100)
```

[ ]:
```
y_pred5 = clf2.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred5))
print("Precision:",metrics.precision_score(test_labels, y_pred5))
print("Recall:",metrics.recall_score(test_labels, y_pred5))
print("F1 Score:",metrics.f1_score(test_labels, y_pred5))
```

```
Accuracy: 0.5876934414148858
Precision: 0.7087144920558498
Recall: 0.4740740740740741
F1 Score: 0.5681204168274798
```

8 8. Cross-Validation & Hyperparameter Tuning for All 3 Models

[ ]:
```
# Cross-Validation: https://scikit-learn.org/stable/modules/cross_validation.
 ↪html
# Hyperparameter Tuning: https://scikit-learn.org/stable/modules/grid_search.
 ↪html
# use gridsearchcv to tune hyperparamaters and apply cross-validation
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': [i for i in range(1, 21)], 'weights': ('uniform',␣
 ↪'distance'),
              'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute')}
gs = GridSearchCV(knn, param_grid, cv = 10)
gs.fit(pca_train, train_labels)
```

[ ]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(n_neighbors=20),
             param_grid={'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute'),
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20],
                         'weights': ('uniform', 'distance')})
```

[ ]:
```
gs.best_params_
```

[ ]:
```
{'algorithm': 'auto', 'n_neighbors': 2, 'weights': 'uniform'}
```

[ ]:
```
gs.best_score_
```

[ ]: 0.551091845947538

[ ]:
```
cv_results = pd.DataFrame(gs.cv_results_)
cv_results.head()
```

[ ]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.043676      0.012715         0.229043        0.060899
1       0.039901      0.013705         0.027185        0.005360
2       0.031235      0.005533         0.176925        0.037223
3       0.035034      0.005196         0.026496        0.004665
4       0.039246      0.008577         0.196674        0.027455

  param_algorithm param_n_neighbors param_weights  \
0            auto                 1       uniform
1            auto                 1      distance
2            auto                 2       uniform
3            auto                 2      distance
4            auto                 3       uniform

                                              params  split0_test_score  \
0  {'algorithm': 'auto', 'n_neighbors': 1, 'weigh…           0.683194
1  {'algorithm': 'auto', 'n_neighbors': 1, 'weigh…           0.683194
2  {'algorithm': 'auto', 'n_neighbors': 2, 'weigh…           0.694544
3  {'algorithm': 'auto', 'n_neighbors': 2, 'weigh…           0.683194
4  {'algorithm': 'auto', 'n_neighbors': 3, 'weigh…           0.694345

   split1_test_score  …  split3_test_score  split4_test_score  \
0           0.489048  …           0.435969           0.528978
1           0.489048  …           0.435969           0.528978
2           0.486061  …           0.463653           0.549492
3           0.489048  …           0.435969           0.528978
4           0.465950  …           0.446923           0.540131

   split5_test_score  split6_test_score  split7_test_score  split8_test_score  \
0           0.532962           0.479785           0.508464           0.689703
1           0.532962           0.479785           0.508464           0.689703
2           0.538140           0.486556           0.523003           0.727345
3           0.532962           0.479785           0.508464           0.689703
4           0.544314           0.481577           0.515037           0.689106

   split9_test_score  mean_test_score  std_test_score  rank_test_score
0           0.535551         0.534721        0.081676              149
1           0.535551         0.534721        0.081676              149
2           0.545708         0.551251        0.084551                1
3           0.535551         0.534721        0.081676              149
4           0.521012         0.538665        0.082023              145
```

```
[5 rows x 21 columns]
```

[ ]:
```
param_grid2 = {'criterion': ("gini", "entropy"), 'splitter': ("best", "random"),
               'max_depth': [i for i in range(1, 11)]}
gs2 = GridSearchCV(clf, param_grid2, cv = 10)
gs2.fit(pca_train, train_labels)
```

[ ]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ('gini', 'entropy'),
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'splitter': ('best', 'random')})
```

[ ]:
```
gs2.best_params_
```

[ ]:
```
{'criterion': 'gini', 'max_depth': 5, 'splitter': 'random'}
```

[ ]:
```
gs2.best_score_
```

[ ]:
```
0.6146783152337244
```

[ ]:
```
cv_results2 = pd.DataFrame(gs2.cv_results_)
cv_results2.head()
```

[ ]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.028043      0.005497         0.000000        0.000000
1       0.005981      0.003631         0.000055        0.000165
2       0.064268      0.017190         0.002408        0.003410
3       0.013156      0.003418         0.000982        0.001379
4       0.083320      0.022284         0.001202        0.002564

  param_criterion param_max_depth param_splitter  \
0            gini               1           best
1            gini               1         random
2            gini               2           best
3            gini               2         random
4            gini               3           best

                                              params  split0_test_score  \
0  {'criterion': 'gini', 'max_depth': 1, 'splitte…           0.582437
1  {'criterion': 'gini', 'max_depth': 1, 'splitte…           0.582437
2  {'criterion': 'gini', 'max_depth': 2, 'splitte…           0.634409
3  {'criterion': 'gini', 'max_depth': 2, 'splitte…           0.635006
4  {'criterion': 'gini', 'max_depth': 3, 'splitte…           0.588411

   split1_test_score  …  split3_test_score  split4_test_score  \
0           0.582437  …           0.582354           0.582354
1           0.582437  …           0.582354           0.582354
2           0.723815  …           0.578769           0.358893
```

```
3           0.582437  …           0.551683           0.581956
4           0.466149  …           0.578769           0.266082

   split5_test_score  split6_test_score  split7_test_score  split8_test_score  \
0           0.582354           0.582354           0.582354           0.582354
1           0.582354           0.582354           0.582354           0.574985
2           0.583947           0.582354           0.590321           0.662418
3           0.571002           0.582354           0.582354           0.614818
4           0.583947           0.582354           0.590321           0.662418

   split9_test_score  mean_test_score  std_test_score  rank_test_score
0           0.582354         0.582359        0.000052                2
1           0.517427         0.575130        0.019360                7
2           0.475204         0.563836        0.102561               12
3           0.527584         0.581143        0.028073                5
4           0.459869         0.522655        0.108356               31

[5 rows x 21 columns]
```

[ ]:
```
param_grid3 = {'n_estimators': [i for i in range(2, 50)],
               'algorithm': ("SAMME", "SAMME.R")}
gs3 = GridSearchCV(clf2, param_grid3, cv = 10)
gs3.fit(pca_train, train_labels)
```

[ ]:
```
GridSearchCV(cv=10, estimator=AdaBoostClassifier(n_estimators=100),
             param_grid={'algorithm': ('SAMME', 'SAMME.R'),
                         'n_estimators': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                          13, 14, 15, 16, 17, 18, 19, 20, 21,
                                          22, 23, 24, 25, 26, 27, 28, 29, 30,
                                          31, …]})
```

[ ]:
```
gs3.best_params_
```

[ ]:
```
{'algorithm': 'SAMME.R', 'n_estimators': 11}
```

[ ]:
```
gs3.best_score_
```

[ ]:
```
0.6039862049721715
```

[ ]:
```
cv_results3 = pd.DataFrame(gs3.cv_results_)
cv_results3.head()
```

[ ]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.053702      0.008339         0.001515        0.000757
1       0.065064      0.003064         0.001338        0.000414
2       0.083556      0.003411         0.001644        0.000511
3       0.123610      0.022795         0.001842        0.000668
4       0.136732      0.009533         0.001964        0.000580
```

```
  param_algorithm param_n_estimators  \
0          SAMME                   2
1          SAMME                   3
2          SAMME                   4
3          SAMME                   5
4          SAMME                   6

                                    params  split0_test_score  \
0  {'algorithm': 'SAMME', 'n_estimators': 2}           0.582437
1  {'algorithm': 'SAMME', 'n_estimators': 3}           0.597571
2  {'algorithm': 'SAMME', 'n_estimators': 4}           0.582437
3  {'algorithm': 'SAMME', 'n_estimators': 5}           0.597571
4  {'algorithm': 'SAMME', 'n_estimators': 6}           0.582437

   split1_test_score  split2_test_score  split3_test_score  split4_test_score  \
0           0.582437           0.378734           0.353316           0.358893
1           0.582437           0.449024           0.574388           0.358893
2           0.582437           0.582238           0.574388           0.582354
3           0.726802           0.449024           0.574388           0.358893
4           0.726802           0.449024           0.574388           0.358893

   split5_test_score  split6_test_score  split7_test_score  split8_test_score  \
0           0.582354           0.582354           0.582354           0.582354
1           0.590918           0.582354           0.521211           0.647281
2           0.590918           0.582354           0.521211           0.582354
3           0.590918           0.582354           0.521211           0.647281
4           0.590918           0.703047           0.521211           0.582354

   split9_test_score  mean_test_score  std_test_score  rank_test_score
0           0.582354         0.516759        0.100413               96
1           0.384585         0.528866        0.093025               79
2           0.582354         0.576305        0.018733               12
3           0.475204         0.552365        0.099693               51
4           0.582354         0.567143        0.102326               27
```

9 9. Report Final Results

[ ]:
```
# e.g. Accuracy, Precision etc.
knn_tuned = KNeighborsClassifier(n_neighbors=2, algorithm="auto",␣
 ↪weights="uniform")
knn_tuned.fit(pca_train, train_labels)
```

[ ]:
```
KNeighborsClassifier(n_neighbors=2)
```

[ ]:
```
knn_tuned_pred = knn_tuned.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, knn_tuned_pred))
print("Precision:",metrics.precision_score(test_labels, knn_tuned_pred))
print("Recall:",metrics.recall_score(test_labels, knn_tuned_pred))
print("F1 Score:",metrics.f1_score(test_labels, knn_tuned_pred))
```

```
Accuracy: 0.5584008843036109
Precision: 0.6783375314861461
Recall: 0.43365539452495977
F1 Score: 0.5290766208251473
```

[ ]:
```
clf_tuned = tree.DecisionTreeClassifier(criterion="gini", max_depth=5,␣
↪splitter="random")
clf_tuned.fit(pca_train, train_labels)
```

[ ]:
```
DecisionTreeClassifier(max_depth=5, splitter='random')
```

[ ]:
```
clf_tuned_pred = clf_tuned.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, clf_tuned_pred))
print("Precision:",metrics.precision_score(test_labels, clf_tuned_pred))
print("Recall:",metrics.recall_score(test_labels, clf_tuned_pred))
print("F1 Score:",metrics.f1_score(test_labels, clf_tuned_pred))
```

```
Accuracy: 0.7430913780397936
Precision: 0.8003511852502195
Recall: 0.7339774557165861
F1 Score: 0.7657286854262915
```

[ ]:
```
clf2_tuned = AdaBoostClassifier(n_estimators=11)
clf2_tuned.fit(pca_train, train_labels)
```

[ ]:
```
AdaBoostClassifier(n_estimators=11)
```

[ ]:
```
clf2_tuned_pred = clf2_tuned.predict(pca_test)
print("Accuracy:",metrics.accuracy_score(test_labels, clf2_tuned_pred))
print("Precision:",metrics.precision_score(test_labels, clf2_tuned_pred))
print("Recall:",metrics.recall_score(test_labels, clf2_tuned_pred))
print("F1 Score:",metrics.f1_score(test_labels, clf2_tuned_pred))
```

```
Accuracy: 0.6625829034635224
Precision: 0.7222901029848141
Recall: 0.6663446054750403
F1 Score: 0.6931903844543094
```