PROJECT PRESENTATION

GROUP – 1 - Adusumilli, S.; Aswar, S.; Boda, S.; Dammalapati, S.; Madala, M.; Mohammed Mukarram Muneer Ahmed; Shah V

FA22 – INTRODUCTION TO INFORMATICS – INFO – I501 – 23087

INSTRUCTOR – DR. SAPTARSHI PURKAYASTHA

PROJECT TITLE

Classification Of Mushrooms In The Seasons They Grow In And Analyzing The Characteristics Based On The Classification

INTRODUCTION

- Mushrooms are one of those bell-shaped fungi which are known for its antioxidant, anti-inflammatory and anti-cancer effects.
- It is known about mushrooms that they possess healing and cleansing properties. Mushrooms are great source of multivitamins such as Vitamin B2, B3, Folate, B5, Phosphorus, Selenium etc. (Harvard University, T.H. Chan).
- According to the Market Analysis Report by Grand View Research (2022-2030), The global mushroom market size was valued at USD 50.3 billion in 2021 and is expected to expand at a compound annual growth rate (CAGR) of 9.7% from 2022 to 2030.
- The U.S. was the second-largest producer accounting for approximately 375 million kg in the year 2019.
- Mushroom is one of the protein-rich vegan sources as it offers nearly 3.3 g of protein per 100 g of serving.
- A secondary mushroom dataset is being used in this project where there are 21 attributes such as class of mushrooms, cap diameter, cap shape, stem height, stem width, habitat, season, gill attachment etc.
- The dataset comprises of mushrooms from 173 species of 23 families.
- The data comprises of one binary class, 17 nominal variables and three quantitative variables.
- Goal is to build a machine learning model by finding correlation between attributes provided in the dataset by doing classification this species of mushrooms with respect to the seasons.

AIM

The aim of this study is to build and evaluate different machine learning models that classify the mushroom species and predict the best season to grow them

PURPOSE

- The purpose of the study is to understand how features of a mushroom and how these features are related to the season that they grow in.
- Will a mushroom that grows in the monsoon have larger stem-height? or will it have a larger cap diameter?
- What will be the average width of the mushroom stem that grows in summer?
- Hence, we aim to examine the factors (variables).
- We plan to build a classifier that predicts the best season for mushroom growth.

RESEARCH DESIGN

- *Approach used* – Quantitative Approach
- *Type of Design* – Correlational Design
- *Type of Study* - Classification Study and Correlational Study

METHODOLOGY

- DATA DESCRIPTION
- DATA COLLECTION
- RESEARCH HYPOTHESIS
- SELECTION OF RESEARCH METHOD
- APPLICATION OF RESEARCH METHOD
- DATA ANALYSIS
- APPLICATION OF MODELS
- PERFORMANCE ANALYSIS

DATA DESCRIPTION

- External Secondary Data Research
- This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family.
- Each species is identified as definitely edible, poisonous, or of unknown edibility and not recommended.
- The latter class was combined with the poisonous one.

DATA COLLECTION

- *Data Collection* – Secondary Data from Mushroom Dataset from UCI Irvine Machine Learning Repository
- *Number of Records* – 61069 Hypothetical Mushrooms with caps based on 173 species (353 mushrooms per species)
- *Origin of the Source* - Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf

NOMINAL VARIABLES	
Shape of the mushroom cap	Color of the stem
Surface of the mushroom cap	Type of veil
Color of the mushroom cap	Type of color of the veil
Whether the mushroom bruise or bleed	Ring is present or not
Attachment of gill to the mushroom	Type of ring
Spacing of gill to the mushroom	Color of the spore
Color of the gill	Habitat
Stem root	Season

METRIC VARIABLES
Diameter of the mushroom cap
Width of the stem of mushroom
Height of the stem of mushroom

ATTRIBUTES OF VARIABLES

Sr. No.	Column Names	Attributes
1.	class	poisonous=p
		edible=e
2.	cap-shape	bell = b
		conical = c
		convex = x
		flat = f
		sunken = s
		spherical = p
		others = o
3.	cap-surface	fibrous=i
		grooves=g
		scaly=y
		smooth=s
		shiny=h
		leathery=l

Sr. No.	Column Names	Attributes
	Cap-surface	silky=k
		sticky=t
		wrinkled=w
		fleshy=e
		dented=d
4.	Cap-color	brown=n
		buff=b
		gray=g
		green=r
		pink=p
		purple=u
		red=e
		white=w
		yellow=y

Sr. No.	Column Names	Attributes
	Cap-color	blue=l
		orange=o
		black=k
5.	does-bruise-or-bleed	bruises-or-bleeding=t
		no=f
6.	gill attachment	adnate=a
		adnexed=x
		deccurent=d
		free=e
		sinuates=s
		pores=p
		none=f
		unknown=?

Sr. No.	Column Names	Attributes
7.	gill-spacing	close=c
		distant=d
		none=f
8.	gill-color	brown=n
		buff=b
		gray=g
		green=r
		pink=p
		purple=u
		red=e
		white=w
		yellow=y
		blue=l
		orange=o
		black=k
		none=f

Sr. No.	Column Names	Attributes
9.	has ring	ring=t
		none=f
10.	ring-type	cobwebby=c
		evanescent=e
		flaring=r
		grooved=g
		large=l
		pendant=p
		sheathings=s
		zone=z
		scaly=y
		movable=m
		none=f
		unknown=?

Sr. No.	Column Names	Attributes
11.	habitat	grasses=g
		leaves=l
		meadows=m
		paths=p
		heaths=h
		urban=u
		waste=w
		woods=d
12.	seasons	spring=s
		summer=u
		autumn=a
		winter=w

Sr. No.	Column Names	Attributes
13.	Stem-color	brown=n
		buff=b
		gray=g
		green=r
		pink=p
		purple=u
		red=e
		white=w
		yellow=y
		blue=l
		orange=o
		black=k
		none=f

RESEARCH HYPOTHESIS

***Null Hypothesis**:*

The characteristics of Mushroom examined do not show correlation or any association with the season in which they grow, and the characteristics cannot classify the Mushrooms in seasons.

***Alternate Hypothesis**:*

The characteristics examined show a correlation or association with the season in which the mushroom grows, and those characteristics can predict or classify the season a specific Mushroom will grow in.

SELECTION OF RESEARCH METHOD

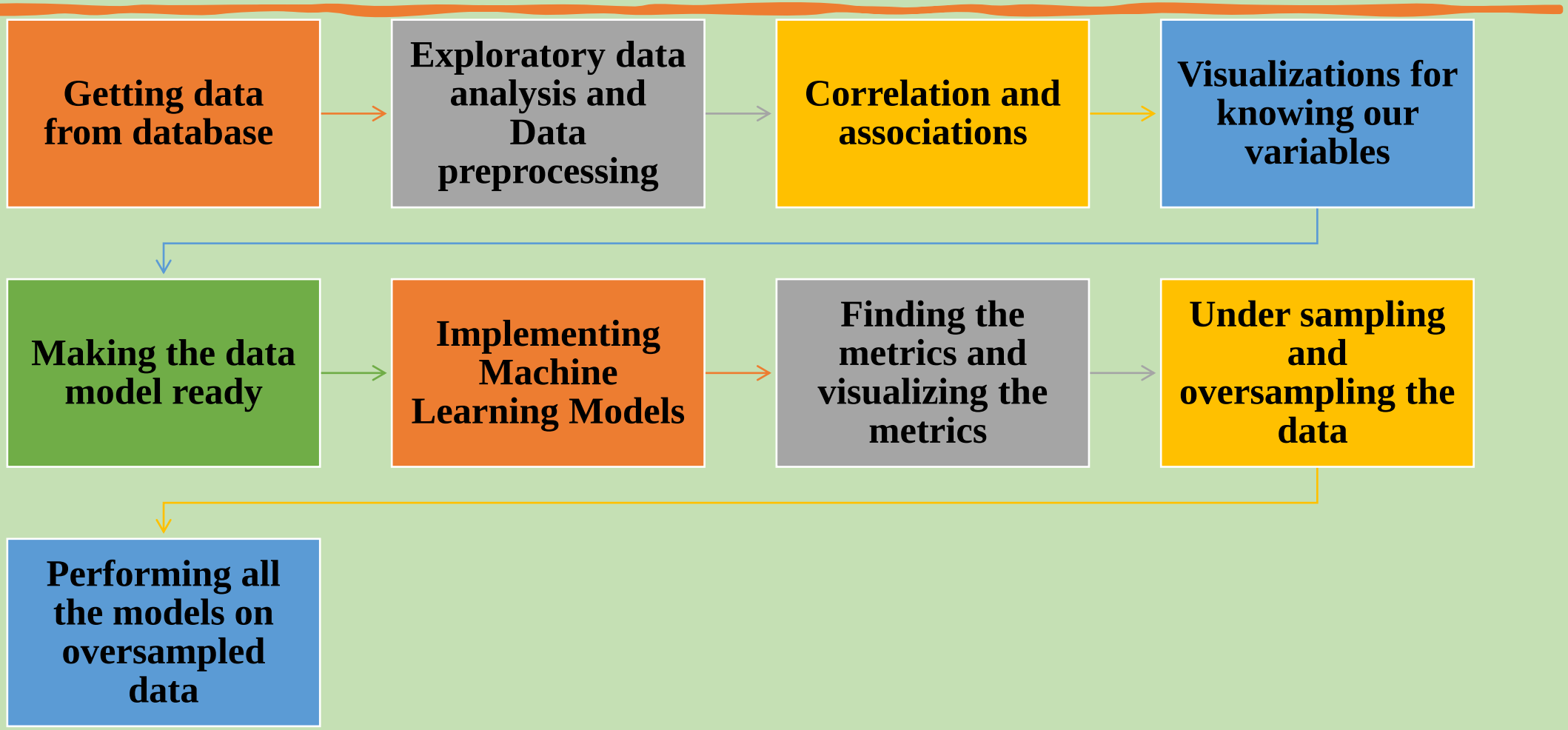
What is Classification model and why do we use it in Prediction modelling?

- Classification may be defined as the process of predicting a class or category from observed values or given data points.
- Classification is a type of supervised machine learning process in which data is predicted using previously labeled, predefined classes of data.
- There are primarily two categories of classification:

1. Binary Classification: There are typically two values when sorting data based on discrete or non-continuous values.
2. Multi-Class Classification: classifying data into three or more classes

- Here, We are trying to predict Season which is a Categorical variable and has 4 Sub-Classes,
- Hence it comes under Multiclass Classification.

APPLICATION OF RESEARCH METHOD



Since we have our data labeled and we have decided on predicting which Species of Mushrooms grow in what Season we can move forward with the Project.

The first Step as in any Data Modelling Project is to import the data from the dataset.

- We imported the Secondary dataset into SQL database.
- So that all the group members have a common connection to the data set, and everyone is working on the same dataset.

We imported the Dataset from SQL to python by connecting them using pymysql Library

SQL CONNECTION

MySQL has been used in this project to make connection and import the dataset from database and storing it in the data frame using library pymysql.

```
# importing libraries for sql connection
```

```
import pymysql
from sqlalchemy import create_engine
import MySQLdb
```

```
myvars = {}
with open("saswar-mysql-password1") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()


myvars.keys()

user, passwd, db =myvars['DB username'],myvars['DB password'],myvars['DB databasename']

engine = create_engine("mysql+pymysql://{user}:{pw}@localhost/{db}".format(user=user,pw=passwd,db=db))
```

```
query = "SELECT * FROM secondary_data_zip;"
```

```
#importing the dataset and storing it in DATA FRAME
sql_df = pd.read_sql(query, con = engine)
```

```
sql_df.drop(index=sql_df.index[0], axis=0, inplace = True)
```

```
df = pd.DataFrame()
df = sql_df.copy(deep=True)
```

- We need to Analyze, Investigate and summarize the key insights
- So, we perform Exploratory Data Analysis to get a basic understanding of how our data is distributed, and its null values to understand and study the relationship between different variables.
- It enables a thorough comprehension of the dataset, the definition or rejection of hypotheses, and the solid construction of predictive models.

DATA ANALYSIS

EXPLORATORY DATA ANALYSIS

Reading data from CSV file

In [79]:
```
df = pd.read_csv("secondary_data.csv", sep=';')
```

In [80]:
```
#Taking a look at the dataframe with head()
df.head()
```

Out[80]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	...	stem-root	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	spore-print-color	habitat	season
0	p	15.26	x	g	o	f	e	NaN	w	16.95	...	s	y	w	u	w	t	g	NaN	d	w
1	p	16.60	x	g	o	f	e	NaN	w	17.99	...	s	y	w	u	w	t	g	NaN	d	u
2	p	14.07	x	g	o	f	e	NaN	w	17.80	...	s	y	w	u	w	t	g	NaN	d	w
3	p	14.17	f	h	e	f	e	NaN	w	15.77	...	s	y	w	u	w	t	p	NaN	d	w
4	p	14.64	x	h	o	f	e	NaN	w	16.53	...	s	y	w	u	w	t	p	NaN	d	w

5 rows × 21 columns

In [81]:
```
df['cap-surface'].unique()
```

Out[81]:
```
array(['g', 'h', nan, 't', 'y', 'e', 's', 'l', 'd', 'w', 'i', 'k'],
      dtype=object)
```

After importing the data with the help of SQL engine, we are now going to read the csv file by using head function.

Cross-validating by checking the unique values for a variable named cap-surface.

Season is our TARGET(Dependent) variable and all other are 'potential' Predictors(Independent Variables).

Columns and 5 Rows can be observed

Everything is correctly imported in dataframe df

In [82]:
```
#Looking at the information about the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61069 entries, 0 to 61068
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   class                61069 non-null  object 
 1   cap-diameter         61069 non-null  float64
 2   cap-shape            61069 non-null  object 
 3   cap-surface          46949 non-null  object 
 4   cap-color            61069 non-null  object 
 5   does-bruise-or-bleed 61069 non-null  object 
 6   gill-attachment      51185 non-null  object 
 7   gill-spacing         36006 non-null  object 
 8   gill-color           61069 non-null  object 
 9   stem-height          61069 non-null  float64
 10  stem-width           61069 non-null  float64
 11  stem-root            9531 non-null   object 
 12  stem-surface         22945 non-null  object 
 13  stem-color           61069 non-null  object 
 14  veil-type            3177 non-null   object 
 15  veil-color           7413 non-null   object 
 16  has-ring             61069 non-null  object 
 17  ring-type            58598 non-null  object 
 18  spore-print-color    6354 non-null   object 
 19  habitat              61069 non-null  object 
 20  season               61069 non-null  object 
dtypes: float64(3), object(18)
memory usage: 9.8+ MB
```

Using info function, information related all the columns can be seen in the output.
It describes column names, range index, number of columns, object type of columns, Non-null values for each column.

- Descriptive Statistics is used to explain, demonstrate and summarize the essential element of the dataset which is then provided in a summary that describes the data sample and its measurements.
- It aids analysts in better understanding the data.

DESCRIPTIVE STATISTICS

In [83]:
```
# Describing all the Categorical Features
df.describe(include=['O'])
```

Out[83]:

	class	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-root	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	spore-print-color	habitat	season
count	61069	61069	46949	61069	61069	51185	36006	61069	9531	22945	61069	3177	7413	61069	58598	6354	61069	61069
unique	2	7	11	12	2	7	3	12	5	8	13	1	6	2	8	7	8	4
top	p	x	t	n	f	a	c	w	s	s	w	u	w	f	f	k	d	a
freq	33888	26934	8196	24218	50479	12698	24710	18521	3177	6025	22926	3177	5474	45890	48361	2118	44209	30177

In [155…
```
# Describing all the Numerical Variables
df.describe()
```

Out[155]:

	cap-diameter	stem-height	stem-width
count	61069.000000	61069.000000	61069.000000
mean	6.733854	6.581538	12.149410
std	5.264845	3.370017	10.035955
min	0.380000	0.000000	0.000000
25%	3.480000	4.640000	5.210000
50%	5.860000	5.950000	10.190000
75%	8.540000	7.740000	16.570000
max	62.340000	33.920000	103.910000

Statistics depicting both Nominal and Metric Variables

The next part of the project is very important as it affects the accuracy of the ML Algorithms.

DATA CLEANING:

Data cleaning is a very important step which is done to ensure that our data analysis is effective enough to achieve high accuracy in the ML algorithms, data cleaning entails filling null values, deleting rows with duplicate data, and dropping columns which create imbalance in the data.

DATA CLEANING

DELETION OF DUPLICATE VALUES

In [84]:
```
Dupes = df[df.duplicated()]
Dupes
```

Out[84]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	...	stem-root	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	spore-print-color	habitat	season
9863	p	1.14	x	g	w	f	a	d	w	3.13	...	NaN	NaN	e	NaN	NaN	f	f	NaN	d	u
12978	p	0.72	x	g	y	f	NaN	NaN	y	3.51	...	NaN	NaN	y	NaN	NaN	f	f	NaN	d	u
56526	p	4.27	o	s	n	f	NaN	c	w	0.00	...	f	f	f	NaN	NaN	f	f	n	d	u
56533	p	4.29	o	t	w	f	NaN	c	w	0.00	...	f	f	f	NaN	NaN	f	f	n	d	u
56576	p	4.59	o	s	w	f	NaN	c	w	0.00	...	f	f	f	NaN	NaN	f	f	n	d	u
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
58237	p	2.94	o	l	g	f	f	f	f	0.00	...	f	f	f	NaN	NaN	f	f	NaN	d	u
58239	p	3.30	o	l	g	f	f	f	f	0.00	...	f	f	f	NaN	NaN	f	f	NaN	d	u
58241	p	3.13	o	l	g	f	f	f	f	0.00	...	f	f	f	NaN	NaN	f	f	NaN	d	w
58242	p	2.83	o	l	g	f	f	f	f	0.00	...	f	f	f	NaN	NaN	f	f	NaN	d	u
58244	p	3.18	o	l	g	f	f	f	f	0.00	...	f	f	f	NaN	NaN	f	f	NaN	d	a

146 rows × 21 columns

In [85]:
```
df.drop_duplicates()
```

Out[85]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	...	stem-root	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	spore-print-color	habitat	season
0	p	15.26	x	g	o	f	e	NaN	w	16.95	...	s	y	w	u	w	t	g	NaN	d	w
1	p	16.60	x	g	o	f	e	NaN	w	17.99	...	s	y	w	u	w	t	g	NaN	d	u
2	p	14.07	x	g	o	f	e	NaN	w	17.80	...	s	y	w	u	w	t	g	NaN	d	w
3	p	14.17	f	h	e	f	e	NaN	w	15.77	...	s	y	w	u	w	t	p	NaN	d	w
4	p	14.64	x	h	o	f	e	NaN	w	16.53	...	s	y	w	u	w	t	p	NaN	d	w
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
61064	p	1.18	s	s	y	f	f	f	f	3.93	...	NaN	NaN	y	NaN	NaN	f	f	NaN	d	a
61065	p	1.27	f	s	y	f	f	f	f	3.18	...	NaN	NaN	y	NaN	NaN	f	f	NaN	d	a
61066	p	1.27	s	s	y	f	f	f	f	3.86	...	NaN	NaN	y	NaN	NaN	f	f	NaN	d	u
61067	p	1.24	f	s	y	f	f	f	f	3.56	...	NaN	NaN	y	NaN	NaN	f	f	NaN	d	u
61068	p	1.17	s	s	y	f	f	f	f	3.25	...	NaN	NaN	y	NaN	NaN	f	f	NaN	d	u

60923 rows × 21 columns

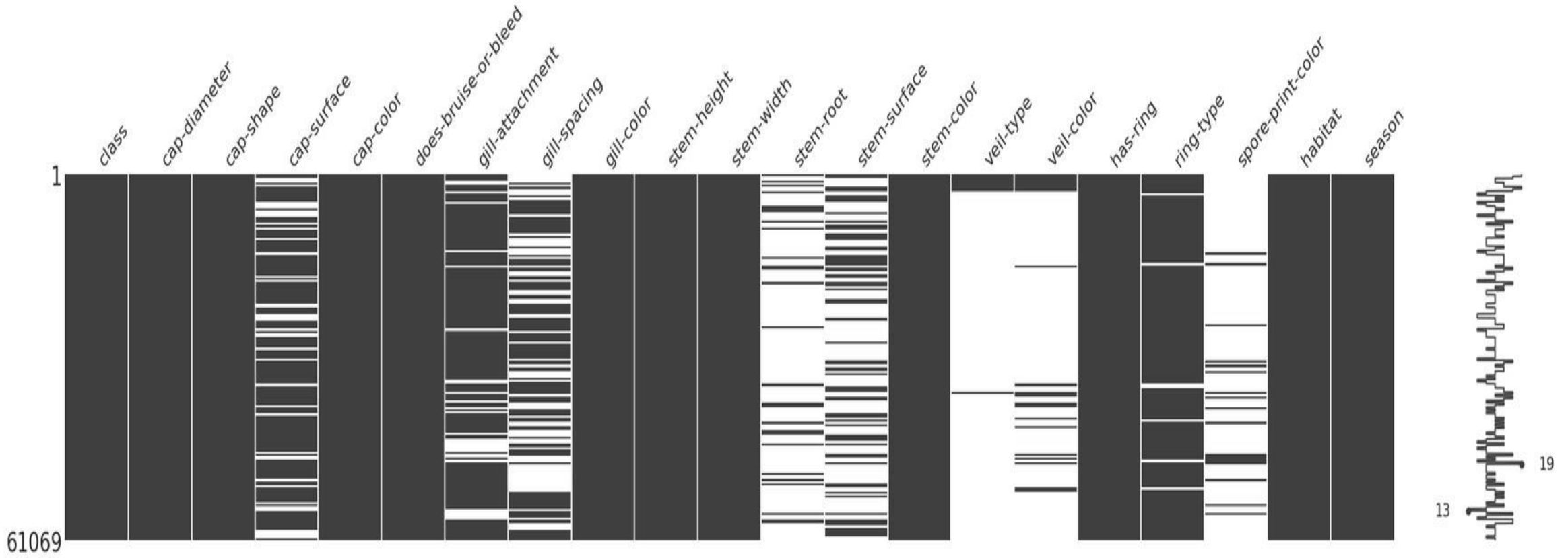
146 rows were found to be duplicate in the dataset

OBSERVATION OF NULL VALUES

Matrix depicting the missing values of the different columns along with variation seen in the sparkline showing distribution of the missing values in the data frame.

```
In [86]: # How many missing values are there in our dataset?
         msno.matrix(df, figsize = (30,5))
```

```
Out[86]: <AxesSubplot:>
```



```
In [88]: df.isnull().sum()
```

```
Out[88]: class                    0
         cap-diameter             0
         cap-shape                0
         cap-surface          14120
         cap-color                0
         does-bruise-or-bleed     0
         gill-attachment       9884
         gill-spacing         25063
         gill-color               0
         stem-height              0
         stem-width               0
         stem-root            51538
         stem-surface         38124
         stem-color               0
         veil-type            57892
         veil-color           53656
         has-ring                 0
         ring-type             2471
         spore-print-color    54715
         habitat                  0
         season                   0
         dtype: int64
```

Dropping off the columns which are greater than 20 % values

```
In [89]: df.drop(['stem-root','stem-surface','veil-type','veil-color','spore-print-color'], axis=1,inplace = True)
```

After deleting columns with too many missing values looking at null values

```
In [90]: df.isnull().sum()
```

```
Out[90]: class                    0
         cap-diameter             0
         cap-shape                0
         cap-surface          14120
         cap-color                0
         does-bruise-or-bleed     0
         gill-attachment       9884
         gill-spacing         25063
         gill-color               0
         stem-height              0
         stem-width               0
         stem-color               0
         has-ring                 0
         ring-type             2471
         habitat                  0
         season                   0
         dtype: int64
```

Imputing missing values in the remaining columns with their MODE, since all of them are categorical variables

```
In [91]: df['cap-surface'].fillna(df['cap-surface'].mode()[0], inplace=True)
         df['gill-attachment'].fillna(df['gill-attachment'].mode()[0], inplace=True)
         df['gill-spacing'].fillna(df['gill-spacing'].mode()[0], inplace=True)
         df['ring-type'].fillna(df['ring-type'].mode()[0], inplace=True)
```

```
In [92]: df.isnull().sum()
```

```
Out[92]: class                    0
         cap-diameter             0
         cap-shape                0
         cap-surface              0
         cap-color                0
         does-bruise-or-bleed     0
         gill-attachment          0
         gill-spacing             0
         gill-color               0
         stem-height              0
         stem-width               0
         stem-color               0
         has-ring                 0
         ring-type                0
         habitat                  0
         season                   0
         dtype: int64
```

Observing null values, deletion of the columns with greater than 20% values is done and then imputing the missing values with mode

Once Data cleaning is done, we then move to Correlation and Association

DATA CORRELATION AND ASSOCIATION:

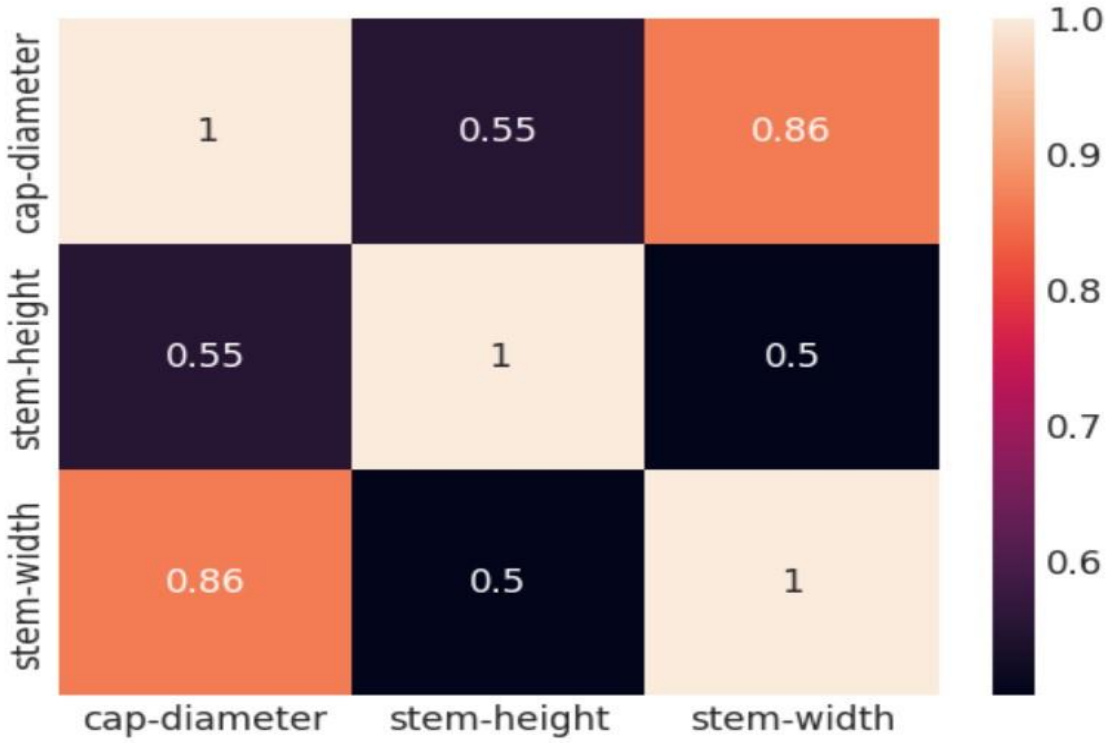
Here we Correlate all variables with each other to find out their strength and the linear relationship of the 2 variables with each other.

However, descriptive analysis reveals that we only have three numerical variables, and the remaining ones are all categorical data. As a result, we are unable to construct a correlation matrix using categorical data, so it is limited to numerical variables.

Plotting Correlation between Continuous Variables

In [110…

```
cor_matrix = df.corr(method='spearman')
sns.heatmap(cor_matrix, annot = True)
plt.show()
```



A correlation matrix has been plotted for continuous variables to determine the relationship between them

Why delete the highly correlated feature ? Because it might make our model give little importance to other variables and if two features are highly correlated for our model it's like repeated features.

Based on highly correlated features - we are dropping one from cap-diameter or stem-width

In [111…

```
df.drop(['stem-width'], axis=1,inplace = True)
```

BIVARIATE ANALYSIS AND COLUMN DESCRIPTION

For the variable named cap surface, we have added dented=d after researching on internet

In [107…

```
def replace_data_with_attributes(df) :
    with open('./names.txt') as file:
        for line in file.readlines():
            line = line.split(':')
            columnName = line[0].strip()
            values = line[1].split(',')
            for value in values:
                value = value.split('=')
                df[columnName].loc[df[columnName]==value[1].strip()] = value[0].strip()
            print('\n========== Description for \''+columnName+'\' column ==========\n')
            print(df[columnName].describe())
    return df

# Let's plot the distribution of each feature
def plot_distribution(dataset, cols=5, width=20, height=15, hspace=0.2, wspace=0.5):
    plt.style.use('seaborn-whitegrid')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, hspace=hspace)
    rows = math.ceil(float(dataset.shape[1]) / cols)
    for i, column in enumerate(dataset.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if dataset.dtypes[column] == object:
            g = sns.countplot(y=column, data=dataset)
            substrings = [s.get_text()[:18] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)
            plt.xticks(rotation=25)
        else:
            g = sns.distplot(dataset[column])
            plt.xticks(rotation=25)

df = replace_data_with_attributes(df)
plot_distribution(df, cols=3, width=20, height=30, hspace=0.45, wspace=0.5)
```

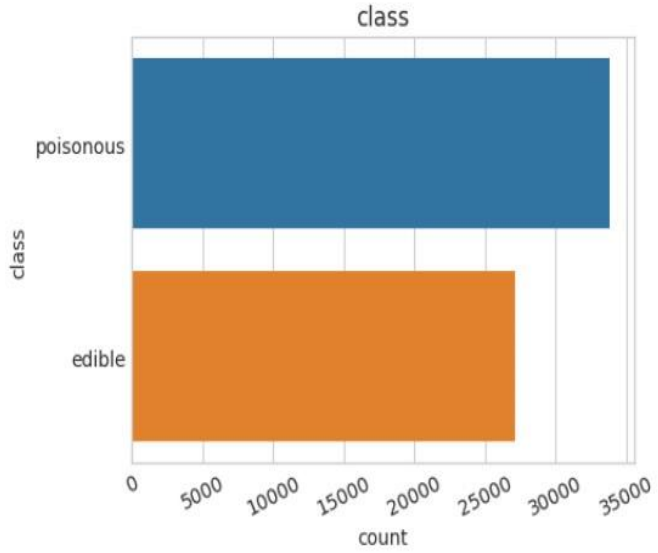
Plots will show frequency distribution of the columns with their attribute names for all the columns after imputing the data

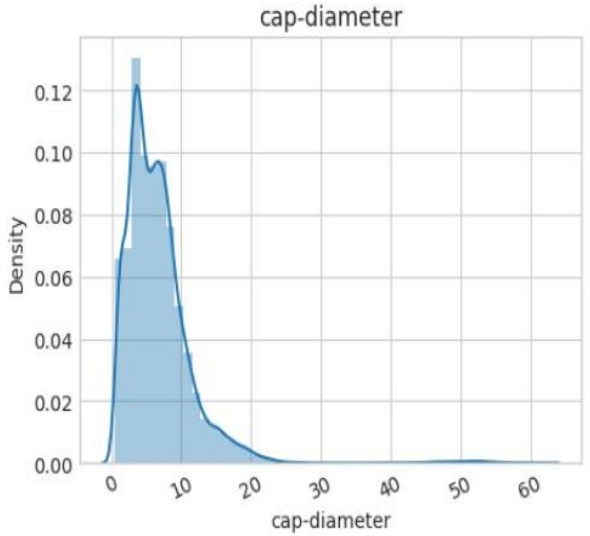
Description of each column after imputation and data cleaning

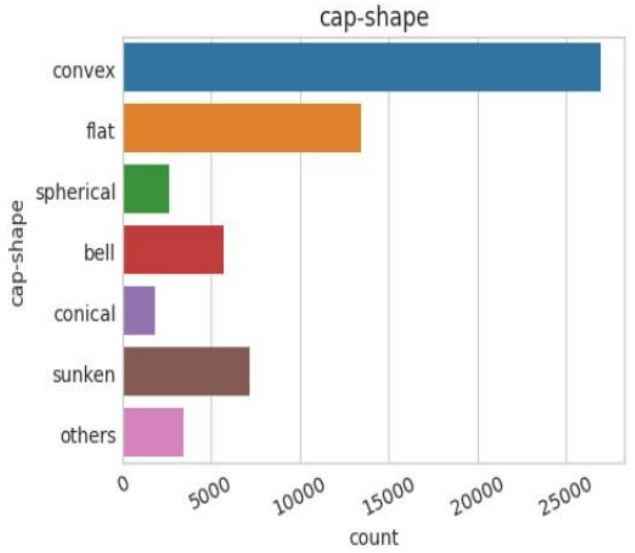
```
========== Description for 'class' column ==========

count         61069
unique            2
top       poisonous
freq          33888
Name: class, dtype: object

========== Description for 'cap-shape' column ==========

count      61069
unique         7
top       convex
freq       26934
Name: cap-shape, dtype: object

========== Description for 'cap-surface' column ==========

count      61069
unique        11
top       sticky
freq       22316
Name: cap-surface, dtype: object

========== Description for 'cap-color' column ==========

count      61069
unique        12
top       brown
freq       24218
Name: cap-color, dtype: object

========== Description for 'does-bruise-or-bleed' column ==

count      61069
unique         2
top           no
freq       50479
Name: does-bruise-or-bleed, dtype: object
```

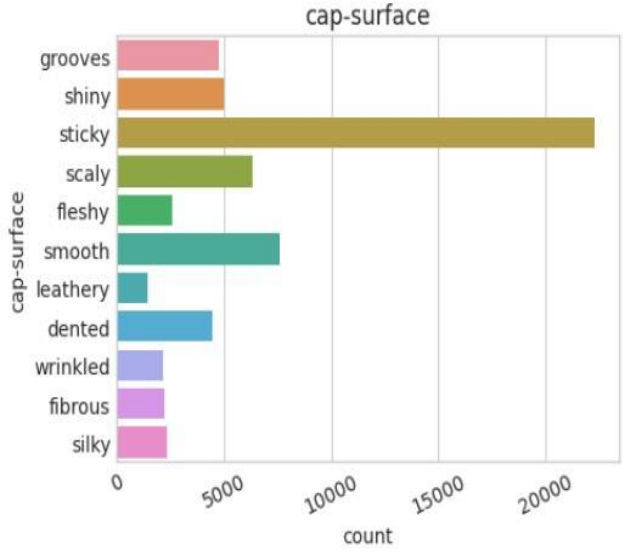
```
========== Description for 'gill-attachment' column ======

count      61069
unique         7
top       adnate
freq       22582
Name: gill-attachment, dtype: object

========== Description for 'gill-spacing' column =========

count      61069
unique         3
top       close
freq       49773
Name: gill-spacing, dtype: object

========== Description for 'gill-color' column ==========

count      61069
unique        12
top       white
freq       18521
Name: gill-color, dtype: object

========== Description for 'stem-color' column ==========

count      61069
unique        13
top       white
freq       22926
Name: stem-color, dtype: object
```

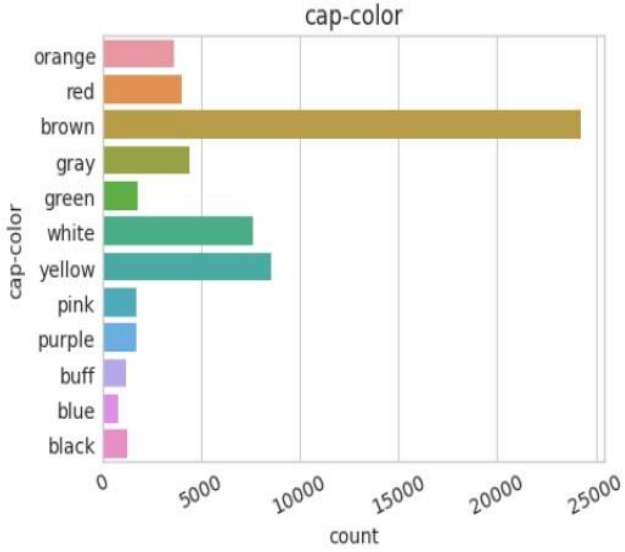
```
========== Description for 'has-ring' column ==========

count      61069
unique         2
top         none
freq       45890
Name: has-ring, dtype: object

========== Description for 'ring-type' column ==========

count      61069
unique         8
top         none
freq       50832
Name: ring-type, dtype: object

========== Description for 'habitat' column ==========

count      61069
unique         8
top        woods
freq       44209
Name: habitat, dtype: object

========== Description for 'season' column ==========

count      61069
unique         4
top       autumn
freq       30177
Name: season, dtype: object
```

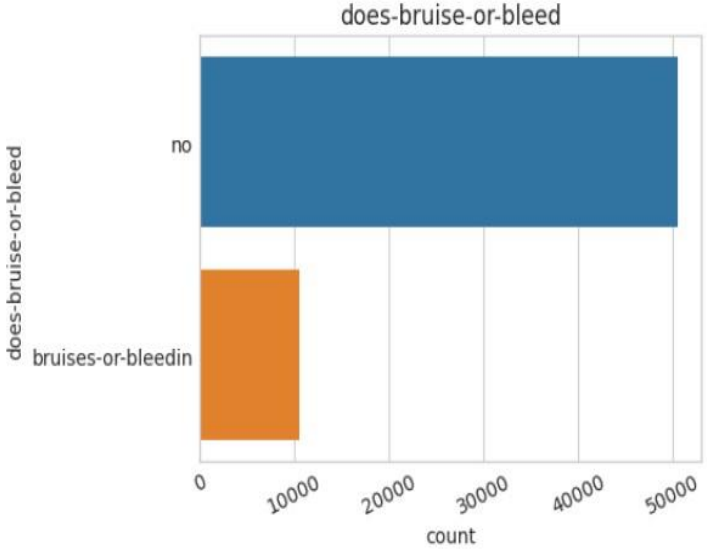

class
poisonous
edible
class
0
5000
10000
15000
20000
25000
30000
35000
count

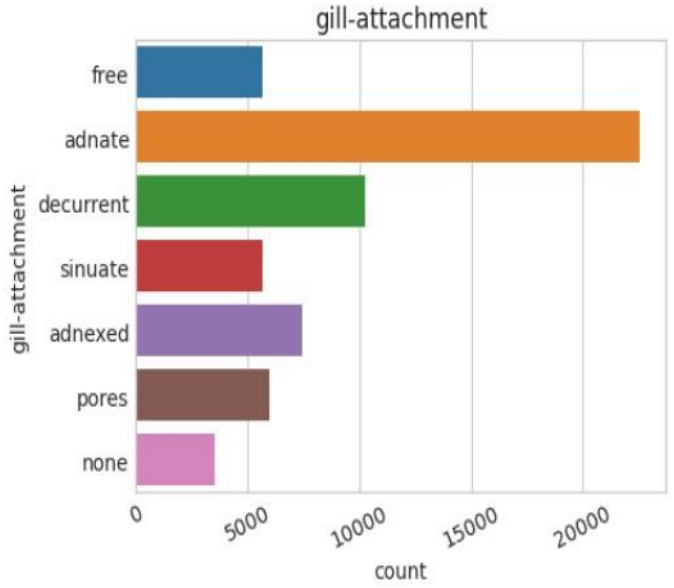

cap-diameter
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
0
10
20
30
40
50
60
cap-diameter

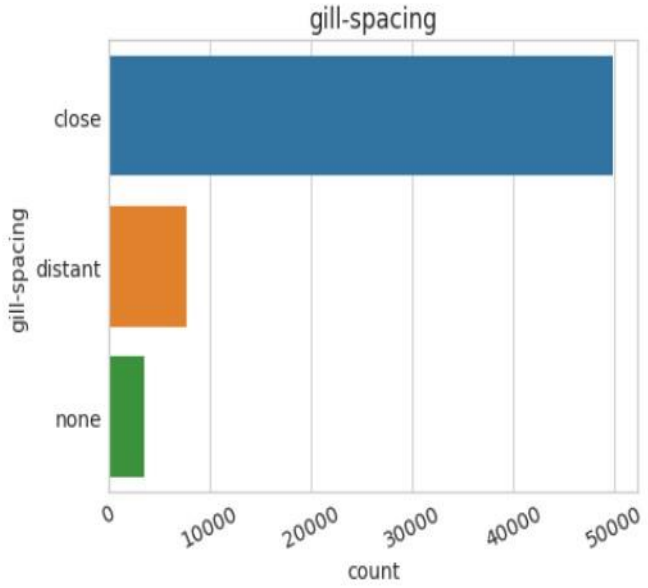

cap-shape
convex
flat
spherical
bell
conical
sunken
others
cap-shape
0
5000
10000
15000
20000
25000
count

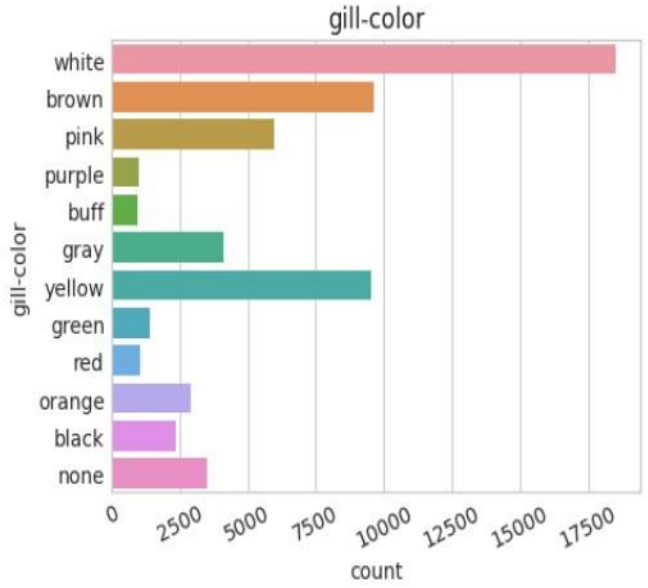

cap-surface
grooves
shiny
sticky
scaly
fleshy
smooth
leathery
dented
wrinkled
fibrous
silky
cap-surface
0
5000
10000
15000
20000
count

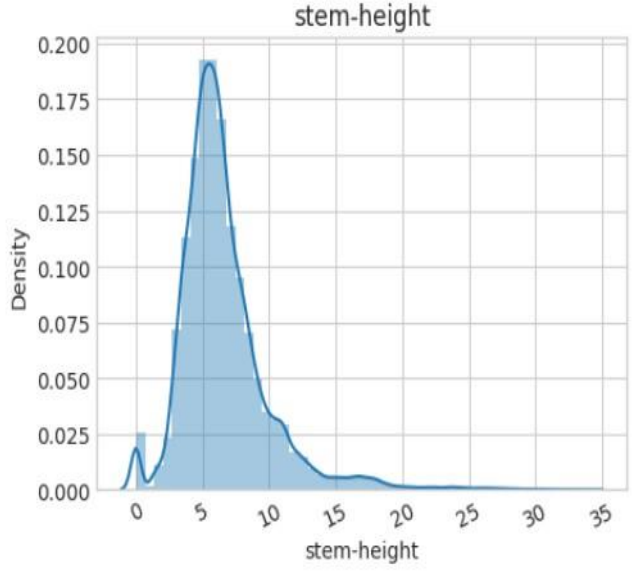

cap-color
orange
red
brown
gray
green
white
yellow
pink
purple
buff
blue
black
cap-color
0
5000
10000
15000
20000
25000
count

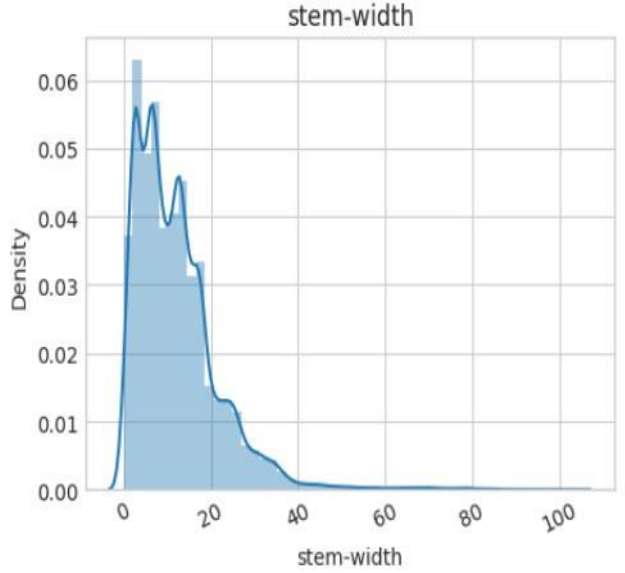

does-bruise-or-bleed
no
bruises-or-bleedin
does-bruise-or-bleed
0
10000
20000
30000
40000
50000
count

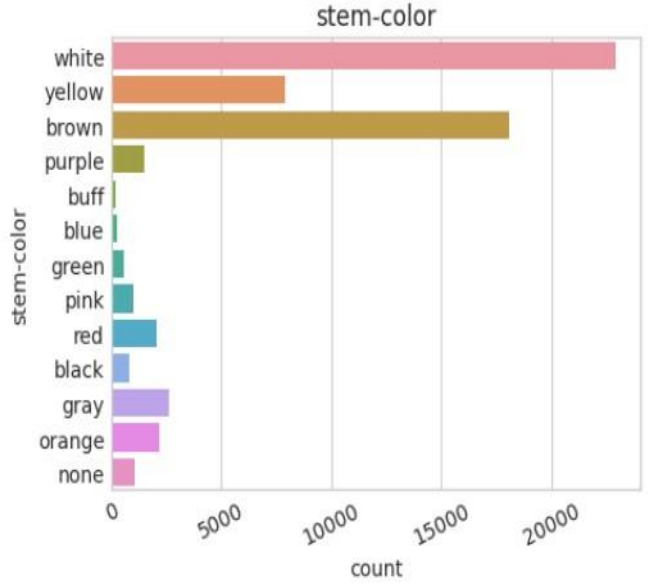

gill-attachment
free
adnate
decurrent
sinuate
adnexed
pores
none
gill-attachment
0
5000
10000
15000
20000
count

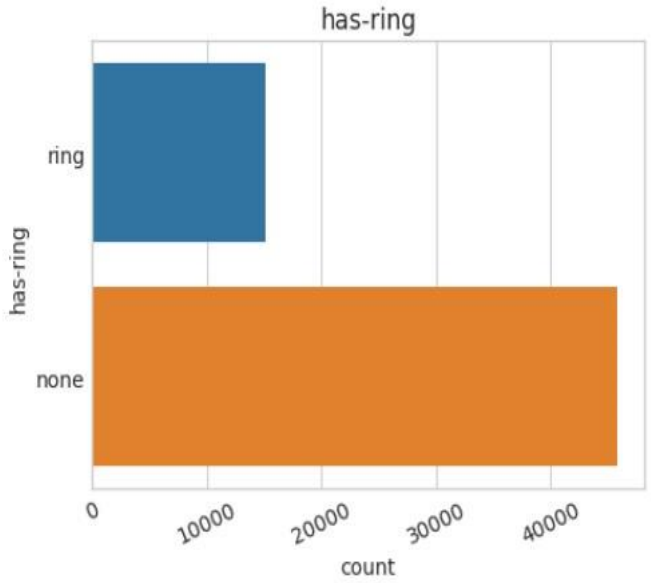

gill-spacing
close
distant
none
gill-spacing
0
10000
20000
30000
40000
50000
count

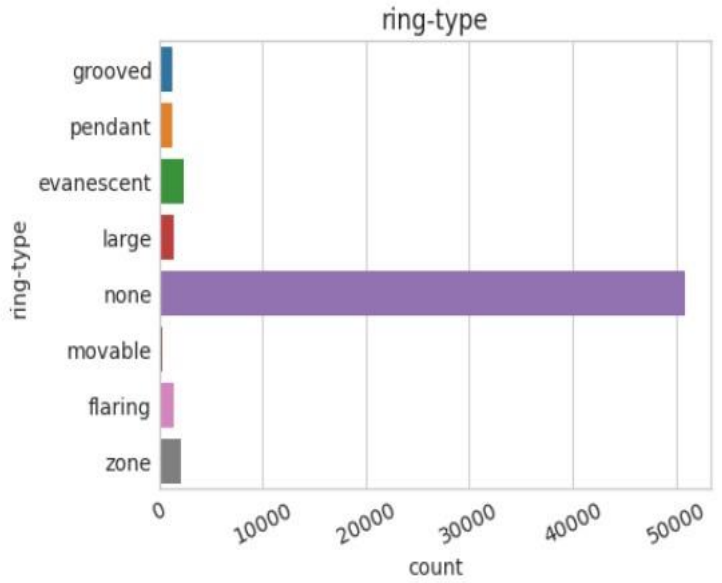

gill-color
white
brown
pink
purple
buff
gray
yellow
green
red
orange
black
none
gill-color
0
2500
5000
7500
10000
12500
15000
17500
count

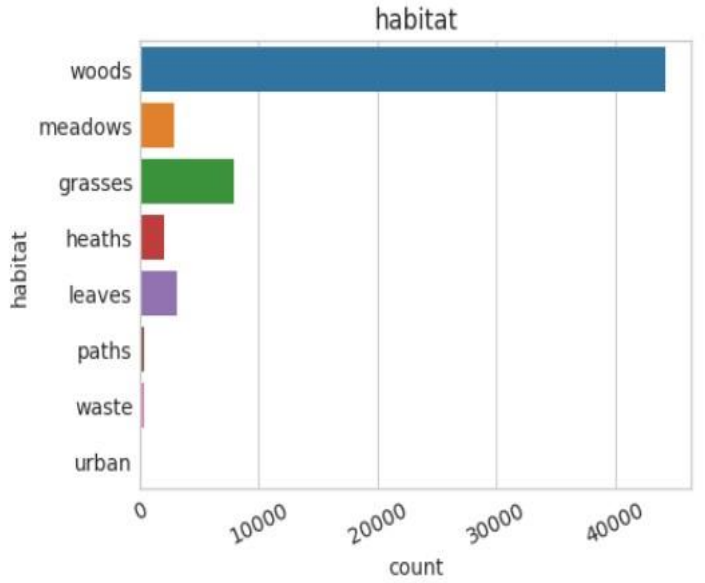

stem-height
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
0
5
10
15
20
25
30
35
stem-height

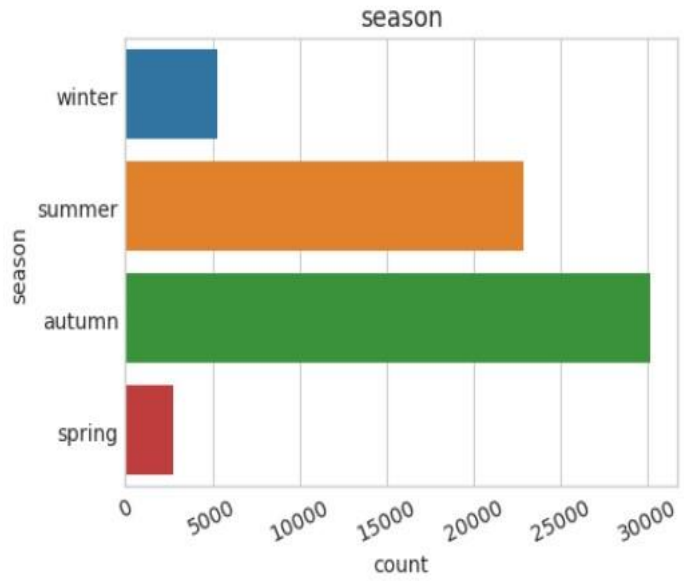

stem-width
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
0
20
40
60
80
100
stem-width


stem-color
white
yellow
brown
purple
buff
blue
green
pink
red
black
gray
orange
none
stem-color
0
5000
10000
15000
20000
count


has-ring
ring
none
has-ring
0
10000
20000
30000
40000
count


ring-type
grooved
pendant
evanescent
large
none
movable
flaring
zone
ring-type
0
10000
20000
30000
40000
50000
count


habitat
woods
meadows
grasses
heaths
leaves
paths
waste
urban
habitat
0
10000
20000
30000
40000
count


season
winter
summer
autumn
spring
season
0
5000
10000
15000
20000
25000
30000
count

CREATION OF NEW DATAFRAMES

To perform our data analysis, let's create new dataframes.

In [114…
```
dataset_bin = pd.DataFrame() # To contain our dataframe with our discretised continuous variables
dataset_bin = df[['class', 'cap-shape', 'cap-surface', 'cap-color',
       'does-bruise-or-bleed', 'gill-attachment', 'gill-spacing', 'gill-color', 'stem-color', 'has-ring', 'ring-type',
       'habitat', 'season']]
```

In [115…
```
dataset_bin
```

Out[115]:

	class	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-color	has-ring	ring-type	habitat	season
0	poisonous	convex	grooves	orange	no	free	close	white	white	ring	grooved	woods	winter
1	poisonous	convex	grooves	orange	no	free	close	white	white	ring	grooved	woods	summer
2	poisonous	convex	grooves	orange	no	free	close	white	white	ring	grooved	woods	winter
3	poisonous	flat	shiny	red	no	free	close	white	white	ring	pendant	woods	winter
4	poisonous	convex	shiny	orange	no	free	close	white	white	ring	pendant	woods	winter
...	...	...	...	...	...	...	...	...	...	...	...	...	...
61064	poisonous	sunken	smooth	yellow	no	none	none	none	yellow	none	none	woods	autumn
61065	poisonous	flat	smooth	yellow	no	none	none	none	yellow	none	none	woods	autumn
61066	poisonous	sunken	smooth	yellow	no	none	none	none	yellow	none	none	woods	summer
61067	poisonous	flat	smooth	yellow	no	none	none	none	yellow	none	none	woods	summer
61068	poisonous	sunken	smooth	yellow	no	none	none	none	yellow	none	none	woods	summer

61069 rows × 13 columns

HYPOTHESIS TESTING

Why Chi Square ?

If the rows and columns of a table are unordered (i.e., are nominal factors), then the most common approach for formally assessing independence is using Pearson’s \chi^2 statistic. It’s often useful to look at the cell-wise contributions to the \chi^2 statistic to see where the evidence for dependence is coming from.

Mapping numbers to season

'autumn' is '0', 'spring' is '1', 'summer' is '2','winter' is '3'

In [116]:
```
dataset_bin.loc[dataset_bin['season'] == 'autumn', 'season'] = '0'
dataset_bin.loc[dataset_bin['season'] == 'spring', 'season'] = '1'
dataset_bin.loc[dataset_bin['season'] == 'summer', 'season'] = '2'
dataset_bin.loc[dataset_bin['season'] == 'winter', 'season'] = '3'
```

After mapping numbers to Seasons the significance of predictors is tested which lists the rounded p values hence showing significant relationship of season variable with other variables.

Testing the significance of predictors with Target Variable.

If the rows and columns of a table are unordered (i.e. are nominal factors), then the most common approach for formally assessing independence is using Pearson's \chi^2 statistic. It's often useful to look at the cell-wise contributions to the \chi^2 statistic to see where the evidence for dependence is coming from.

In [660…
```
def Asses_variable(var1, var2):
    # Contigency Table
    table = sm.stats.Table.from_data(df[[var1, var2]])
    print("\n\nTable Original")
    print(table.table_orig)

    # p-value
    rslt = table.test_ordinal_association()
    print("P-value is: ", rslt.pvalue)

    # Grouped plots for visualization
    plt.figure(figsize = (3,5))
    sns.catplot(x=var2, col=var1,
                data=df, kind="count",
                height=8, aspect=.8);
    return rslt.pvalue
```

In [661…
```
# We will save all the p-values in a list
pvalues_list = []
```

In [238…

```
pvalues_list_rounded = [round(item, 15) for item in pvalues_list]

pvalues_list_rounded
```

Listed are the p-values for different variables

Out[238]:

```
[0.306001760949946,
 0.0,
 0.0,
 0.015122017659931,
 0.0,
 0.002636280376777,
 0.0,
 5.1727e-11,
 0.0,
 0.0,
 0.0,
 0.04351327793937]
```

pvalue_list has almost all very low values, so all the features have significant relationship with "season" variable.

Hence we are rejecting the null hypothesis.

REJECTING THE NULL HYPOTHESIS

For almost all the variables we found p value less than 0.05, hence we can reject null hypothesis.

Strong association has been found between the target variable – season and other variables hence it can be predicted that, in which season mushrooms have better growth condition.

FEATURE ENCODING

Feature Encoding

Label Encoding

Label Encoding is done for all the categorical variables.

In [132…
```
categorical_variables = identify_nominal_columns(df)
categorical_variables
```

Out[132]:
```
['class',
 'cap-shape',
 'cap-surface',
 'cap-color',
 'does-bruise-or-bleed',
 'gill-attachment',
 'gill-spacing',
 'gill-color',
 'stem-color',
 'has-ring',
 'ring-type',
 'habitat',
 'season']
```

In [133…
```
coulmns = df.columns
coulmns
```

Out[133]:
```
Index(['class', 'cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
       'does-bruise-or-bleed', 'gill-attachment', 'gill-spacing', 'gill-color',
       'stem-height', 'stem-color', 'has-ring', 'ring-type', 'habitat',
       'season'],
      dtype='object')
```

In [134…
```
LE = LabelEncoder()
dataset_con_enc = pd.DataFrame()
for col in coulmns:
    if col in categorical_variables:
        dataset_con_enc[col] = LE.fit_transform(df[col])
    else:
        dataset_con_enc[col] = df[col]
dataset_con_enc.head()
```

Out[134]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	stem-color	has-ring	ring-type	habitat	season
0	1	15.26	2	3	6	1	3	0	10	16.95	11	1	2	7	3
1	1	16.60	2	3	6	1	3	0	10	17.99	11	1	2	7	2
2	1	14.07	2	3	6	1	3	0	10	17.80	11	1	2	7	3
3	1	14.17	3	6	9	1	3	0	10	15.77	11	1	6	7	3
4	1	14.64	2	6	6	1	3	0	10	16.53	11	1	6	7	3

One Hot Encoding

One Hot Encoding is done for all the variables

In [135…
```
# One Hot Encodes all labels before Machine Learning
one_hot_cols = dataset_bin.columns.tolist()
one_hot_cols.remove('season')
dataset_bin_enc = pd.get_dummies(dataset_bin, columns=one_hot_cols)

dataset_bin_enc.head()
```

Out[135]:

	season	class_edible	class_poisonous	cap-shape_bell	cap-shape_conical	cap-shape_convex	cap-shape_flat	cap-shape_others	cap-shape_spherical	cap-shape_sunken	...	ring-type_pendant	ring-type_zone	habitat_grasses	habita
0	3	0	1	0	0	1	0	0	0	0	...	0	0	0	
1	2	0	1	0	0	1	0	0	0	0	...	0	0	0	
2	3	0	1	0	0	1	0	0	0	0	...	0	0	0	
3	3	0	1	0	0	0	1	0	0	0	...	1	0	0	
4	3	0	1	0	0	1	0	0	0	0	...	1	0	0	

5 rows × 88 columns

In [136…
```
dataset_con_enc.columns
```

Out[136]:
```
Index(['class', 'cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
       'does-bruise-or-bleed', 'gill-attachment', 'gill-spacing', 'gill-color',
       'stem-height', 'stem-color', 'has-ring', 'ring-type', 'habitat',
       'season'],
      dtype='object')
```

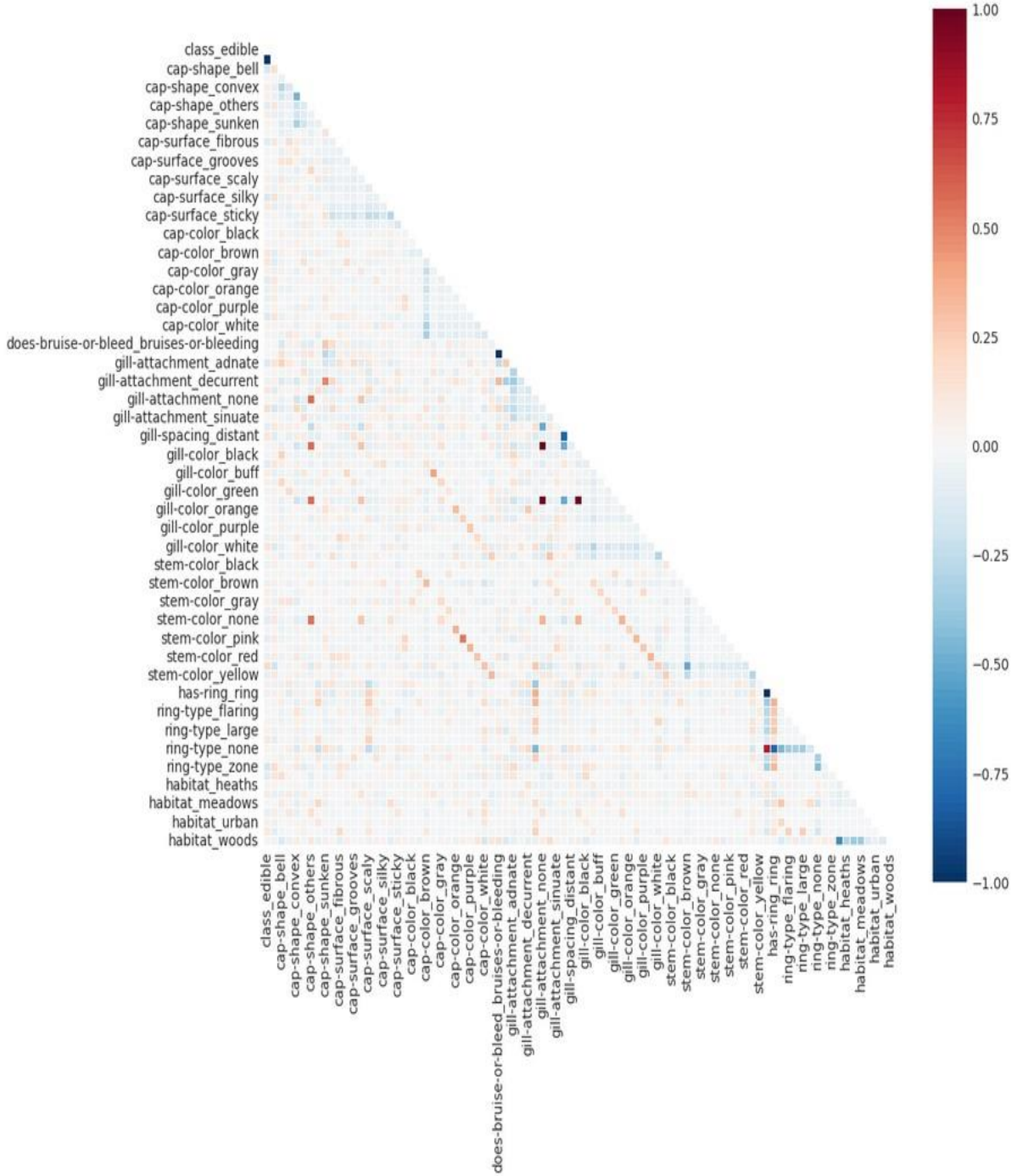
Feature Encoding has been done to improve machine learning

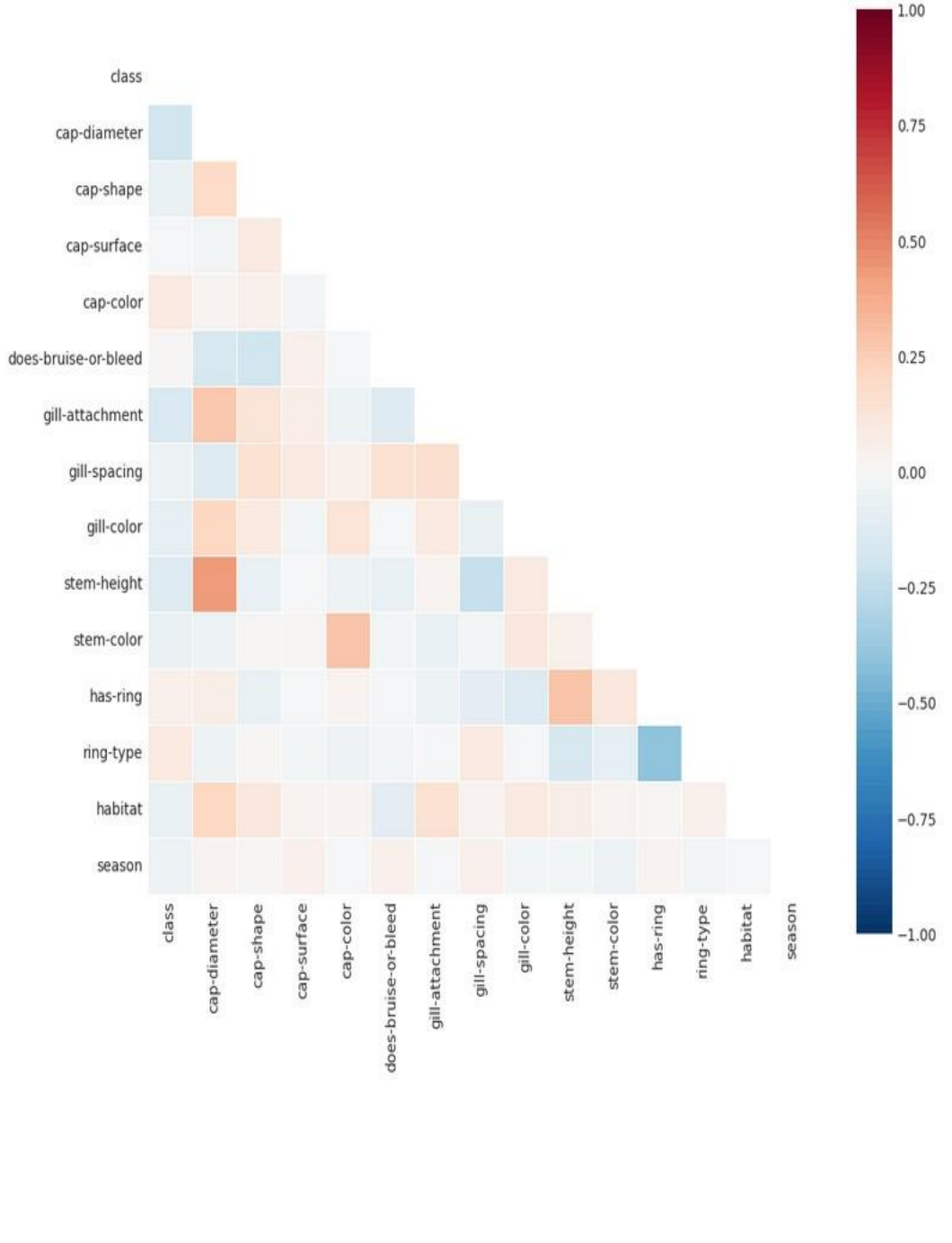
FEATURE CORRELATION

In [338…

```
# Create a correlation plot of both datasets.
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(25,10))

plt.subplot(1,2,1)
#Generate the mask for the upper triangle
mask = np.zeros_like(dataset_bin_enc.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)]=True
sns.heatmap(dataset_bin_enc.corr(),
            vmin=-1, vmax=1,
            square=True,
            cmap=sns.color_palette("RdBu_r", 100),
            mask=mask,
            linewidths=0.5);

plt.subplot(1, 2, 2)
mask = np.zeros_like(dataset_con_enc.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(dataset_con_enc.corr(),
             vmin=-1, vmax=1,
             square=True,
             cmap=sns.color_palette("RdBu_r", 100),
             mask=mask,
             linewidths=.5);
```

Feature correlation is plotted for both datasets to see association between them


class_edible
cap-shape_bell
cap-shape_convex
cap-shape_others
cap-shape_sunken
cap-surface_fibrous
cap-surface_grooves
cap-surface_scaly
cap-surface_silky
cap-surface_sticky
cap-color_black
cap-color_brown
cap-color_gray
cap-color_orange
cap-color_purple
cap-color_white
does-bruise-or-bleed_bruises-or-bleeding
gill-attachment_adnate
gill-attachment_decurrent
gill-attachment_none
gill-attachment_sinuate
gill-spacing_distant
gill-color_black
gill-color_buff
gill-color_green
gill-color_orange
gill-color_purple
gill-color_white
stem-color_black
stem-color_brown
stem-color_gray
stem-color_none
stem-color_pink
stem-color_red
stem-color_yellow
has-ring_ring
ring-type_flaring
ring-type_large
ring-type_none
ring-type_zone
habitat_heaths
habitat_meadows
habitat_urban
habitat_woods
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


class
cap-diameter
cap-shape
cap-surface
cap-color
does-bruise-or-bleed
gill-attachment
gill-spacing
gill-color
stem-height
stem-color
has-ring
ring-type
habitat
season
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

CramerV Correlation to see if there are any highly correlated columns

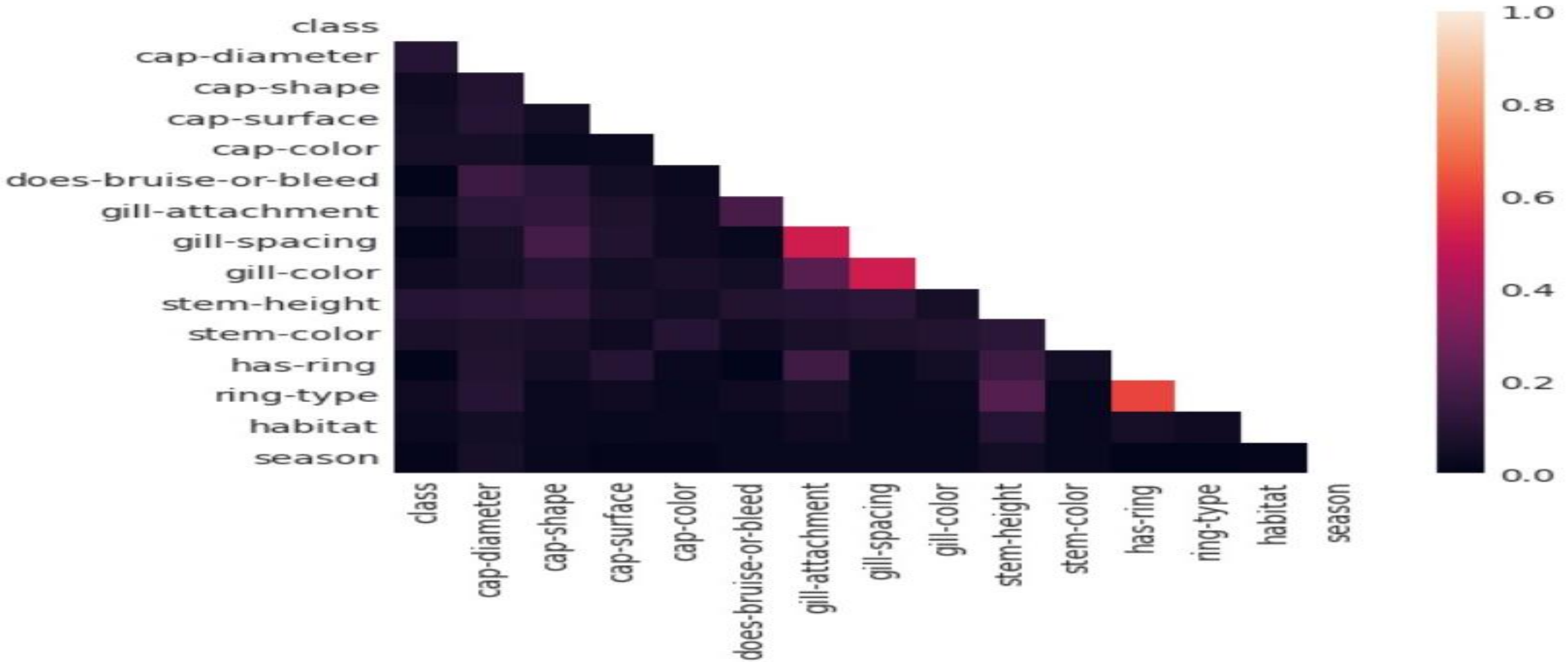
In [339]:
```
from scipy.stats import chi2_contingency


def cramers_V(var1,var2) :
  crosstab =np.array(pd.crosstab(var1,var2, rownames=None, colnames=None)) # Cross table building
  stat = chi2_contingency(crosstab)[0] # Keeping of the test statistic of the Chi2 test
  obs = np.sum(crosstab) # Number of observations
  mini = min(crosstab.shape)-1 # Take the minimum value between the columns and the rows of the cross table
  return (stat/(obs*mini))

rows= []

for var1 in dataset_con_enc:
  col = []
  for var2 in dataset_con_enc :
    cramers =cramers_V(dataset_con_enc[var1], dataset_con_enc[var2]) # Cramer's V test
    col.append(round(cramers,2)) # Keeping of the rounded value of the Cramer's V
  rows.append(col)

cramers_results = np.array(rows)
cramers = pd.DataFrame(cramers_results, columns = dataset_con_enc.columns, index =dataset_con_enc.columns)
cramers
```

CramerV Correlation matrix is done to check highly related columns

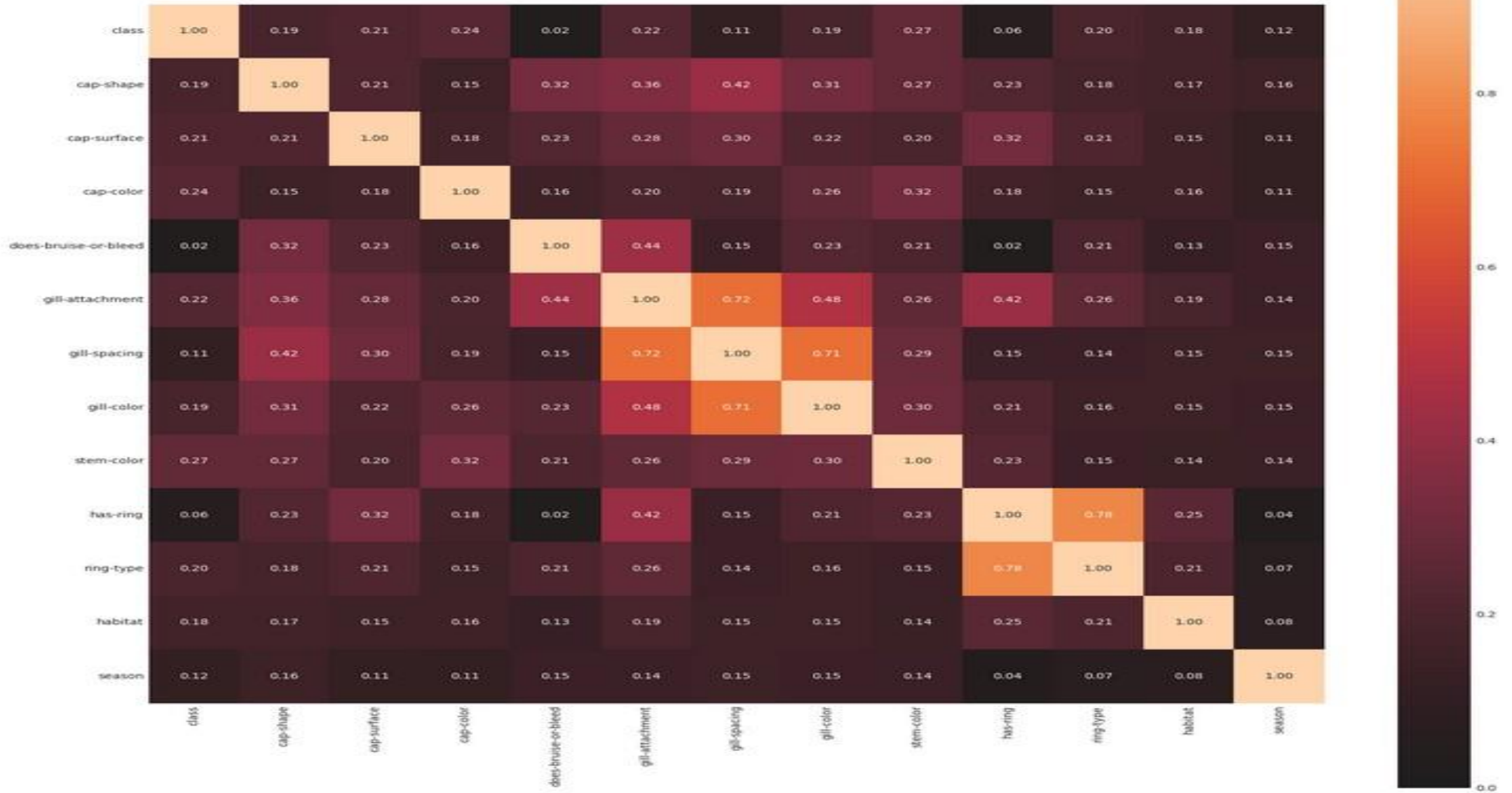
Out[339]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	stem-color	has-ring	ring-type	habitat	season
class	1.00	0.10	0.04	0.05	0.06	0.00	0.05	0.01	0.04	0.10	0.07	0.00	0.04	0.03	0.01
cap-diameter	0.10	1.00	0.09	0.10	0.06	0.16	0.11	0.07	0.06	0.11	0.08	0.09	0.10	0.05	0.06
cap-shape	0.04	0.09	1.00	0.05	0.02	0.11	0.13	0.18	0.10	0.13	0.07	0.05	0.03	0.03	0.02
cap-surface	0.05	0.10	0.05	1.00	0.03	0.05	0.08	0.09	0.05	0.07	0.04	0.10	0.04	0.02	0.01
cap-color	0.06	0.06	0.02	0.03	1.00	0.03	0.04	0.04	0.07	0.05	0.10	0.03	0.02	0.03	0.01
does-bruise-or-bleed	0.00	0.16	0.11	0.05	0.03	1.00	0.19	0.02	0.05	0.09	0.04	0.00	0.04	0.02	0.02
gill-attachment	0.05	0.11	0.13	0.08	0.04	0.19	1.00	0.51	0.23	0.10	0.07	0.17	0.07	0.04	0.02
gill-spacing	0.01	0.07	0.18	0.09	0.04	0.02	0.51	1.00	0.51	0.11	0.08	0.02	0.02	0.02	0.02
gill-color	0.04	0.06	0.10	0.05	0.07	0.05	0.23	0.51	1.00	0.06	0.09	0.05	0.03	0.02	0.02
stem-height	0.10	0.11	0.13	0.07	0.05	0.09	0.10	0.11	0.06	1.00	0.11	0.16	0.22	0.10	0.05
stem-color	0.07	0.08	0.07	0.04	0.10	0.04	0.07	0.08	0.09	0.11	1.00	0.05	0.02	0.02	0.02
has-ring	0.00	0.09	0.05	0.10	0.03	0.00	0.17	0.02	0.05	0.16	0.05	1.00	0.61	0.06	0.00
ring-type	0.04	0.10	0.03	0.04	0.02	0.04	0.07	0.02	0.03	0.22	0.02	0.61	1.00	0.04	0.00
habitat	0.03	0.05	0.03	0.02	0.03	0.02	0.04	0.02	0.02	0.10	0.02	0.06	0.04	1.00	0.01
season	0.01	0.06	0.02	0.01	0.01	0.02	0.02	0.02	0.02	0.05	0.02	0.00	0.00	0.01	1.00

```
import seaborn as sns
import matplotlib.pyplot as plt


mask = np.zeros_like(cramers, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True


with sns.axes_style("white"):
   ax = sns.heatmap(cramers, mask=mask,vmin=0., vmax=1, square=True)

plt.show()
```



Association between all the Variables



Deleting the highly correlated features 'gill-spacing' and 'has-ring'

In [343…
```
dataset_bin_enc.drop(['has-ring_none','has-ring_ring','gill-spacing_close','gill-spacing_distant','gill-spacing_none'], axis=1,inplace = True)
dataset_con_enc.drop(['has-ring','gill-spacing'], axis=1,inplace = True)
```

Standard Scaler for normalizing the cap-diameter and stem-height

In [344…
```
scaled_features = dataset_con_enc[['cap-diameter','stem-height']]
col_names = ['cap-diameter','stem-height']
features = scaled_features[col_names]
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
scaled_features[col_names] = features

dataset_con_enc[['cap-diameter','stem-height']] = scaled_features[['cap-diameter','stem-height']]
dataset_con_enc
```

Out[344]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-color	stem-height	stem-color	ring-type	habitat	season
0	1	1.619462	2	3	6	1	3	10	3.076705	11	2	7	3
1	1	1.873982	2	3	6	1	3	10	3.385311	11	2	7	2
2	1	1.393432	2	3	6	1	3	10	3.328931	11	2	7	3
3	1	1.412426	3	6	9	1	3	10	2.726555	11	6	7	3
4	1	1.501699	2	6	6	1	3	10	2.952075	11	6	7	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
61064	1	-1.054903	6	8	11	1	4	5	-0.786809	12	5	7	0
61065	1	-1.037808	3	8	11	1	4	5	-1.009362	12	5	7	0
61066	1	-1.037808	6	8	11	1	4	5	-0.807581	12	5	7	2
61067	1	-1.043506	3	8	11	1	4	5	-0.896602	12	5	7	2
61068	1	-1.056802	6	8	11	1	4	5	-0.988590	12	5	7	2

61069 rows × 13 columns

Why scaling ?
Since our numerical variables vary more than our label encoded variables, we need to perform feature scaling. We chose Standard Scaler.

Finding the correlation between 4 seasons and all the other one hot encoded features

```
one_hot_cols_temp = dataset_bin.columns.tolist()
dataset_bin_enc_temp = pd.get_dummies(dataset_bin, columns=one_hot_cols_temp)

dataset_bin_enc_temp[['cap-diameter', 'stem-height']] = scaled_features[['cap-diameter', 'stem-height']]


d1=dataset_bin_enc_temp.corrwith((dataset_bin_enc_temp.season_1))
d0=dataset_bin_enc_temp.corrwith((dataset_bin_enc_temp.season_0))
d2=dataset_bin_enc_temp.corrwith((dataset_bin_enc_temp.season_2))
d3=dataset_bin_enc_temp.corrwith((dataset_bin_enc_temp.season_3))


d0 = pd.DataFrame(d0)
d1 = pd.DataFrame(d1)
d2 = pd.DataFrame(d2)
d3 = pd.DataFrame(d3)

df_all_cols = pd.concat([d0,d1,d2,d3], axis = 1)
df_all_cols.columns=['autumn','spring','summer','winter']

print(df_all_cols.to_markdown())
```

"Gill-spacing" and "Has-ring" has been deleted as they were found to be highly related.
Standard scaler has been applied to normalize the "cap-diameter" and "Stem-height" and Correlation has been found between 4 seasons and other variables.

	autumn	spring	summer	winter
class_edible	-0.0425984	0.054119	-0.0370658	0.0999677
class_poisonous	0.0425984	-0.054119	0.0370658	-0.0999677
cap-shape_bell	-0.0372465	0.0724562	0.0245476	-0.0293066
cap-shape_conical	-0.0229273	0.074676	0.0229981	-0.0537695
cap-shape_convex	0.0468131	-0.0857011	-0.0102175	-0.00269837
cap-shape_flat	0.02433	-0.0577599	-0.0316974	0.0538273
cap-shape_others	-0.0970075	0.17815	-0.000341941	0.0422834
cap-shape_spherical	-0.00921618	0.0836667	-0.0139319	-0.0211182
cap-shape_sunken	0.0177036	-0.0788161	0.0312019	-0.0273482
cap-surface_dented	0.0238531	-0.0314476	0.0105878	-0.0376026
cap-surface_fibrous	0.0623209	-0.0420403	-0.0159351	-0.0525805
cap-surface_fleshy	0.0672163	-0.0226013	-0.0668555	0.0122103
cap-surface_grooves	-0.0370664	0.0483874	0.0216154	-0.00686228
cap-surface_leathery	-0.0404765	0.0121055	0.0154319	0.0365685
cap-surface_scaly	0.0183192	0.0132149	-0.00627347	-0.0315301
cap-surface_shiny	-0.0264488	0.0104024	0.0183002	0.00789233
cap-surface_silky	-0.0447047	0.0196273	0.0228135	0.0258352
cap-surface_smooth	-0.0424931	0.0312734	-0.0306905	0.105595
cap-surface_sticky	0.0262289	-0.0298828	0.0132441	-0.0475657
cap-surface_wrinkled	-0.0208657	-0.00387454	0.0104339	0.0220185
cap-color_black	-0.00800821	-0.0316208	0.0272674	-0.00949507
cap-color_blue	-0.00344173	-0.0253467	0.0133192	0.00180956
cap-color_brown	-0.0015551	0.0188904	0.00790952	-0.0247688
cap-color_buff	0.0413123	-0.0309965	-0.0689078	0.0680601
cap-color_gray	-0.0380627	0.0776027	-0.0358064	0.0724401
cap-color_green	-0.00205648	-0.0308878	0.0262931	-0.018954
cap-color_orange	0.016208	-0.0151238	-0.0397542	0.050817
cap-color_pink	-0.000901234	-0.0250612	-0.00812398	0.0340534
cap-color_purple	0.0126109	-0.0366839	0.00783543	-0.00898163
cap-color_red	0.0452396	-0.0185685	-0.0306284	-0.0140902
cap-color_white	-0.023737	0.0228949	0.0272186	-0.0215081
cap-color_yellow	-0.00703331	-0.00445289	0.0383009	-0.0502471
does-bruise-or-bleed_bruises-or-bleeding	0.0376305	-0.0747341	0.0634545	-0.121459
does-bruise-or-bleed_no	-0.0376305	0.0747341	-0.0634545	0.121459
gill-attachment_adnate	-0.0126738	0.0288409	-0.0211735	0.0378654
gill-attachment_adnexed	0.0201477	-0.00340429	0.00833498	-0.0477511
gill-attachment_decurrent	0.0220419	-0.0970788	-0.00535342	0.0414012
gill-attachment_free	-0.00960217	-0.0200337	0.0378593	-0.0334469
gill-attachment_none	-0.0633418	0.187311	-0.0153033	0.00138698
gill-attachment_pores	-0.0161046	-0.0298217	0.0665768	-0.0641878
gill-attachment_sinuate	0.0471495	-0.0192128	-0.0611479	0.0356127
gill-spacing_close	0.0617357	-0.103642	0.0280049	-0.0819888
gill-spacing_distant	-0.0275789	-0.0104196	-0.0219181	0.0945803
gill-spacing_none	-0.0633418	0.187311	-0.0153033	0.00138698
gill-color_black	-0.0258509	0.0344237	-0.00288907	0.0256946
gill-color_brown	-0.0154508	0.03072	0.00190874	0.00162356
gill-color_buff	0.0542982	-0.0272355	-0.0664744	0.0379694
gill-color_gray	-0.0215362	0.03829	-0.00027734	0.0106617
gill-color_green	-0.00225705	0.00875858	0.0190907	-0.0353459
gill-color_none	-0.0633418	0.187311	-0.0153033	0.00138698
gill-color_orange	0.00300306	-0.0483517	-0.015998	0.0578143
gill-color_pink	0.0112956	-0.0304473	0.0154353	-0.0243335
gill-color_purple	0.0182439	-0.0282194	-0.0193917	0.0217113
gill-color_red	0.0523287	-0.0288167	-0.0324657	-0.0160072
gill-color_white	-0.0143981	-0.065719	0.0487492	-0.0100723
gill-color_yellow	0.0491418	-0.0334608	-0.0219163	-0.0251069
stem-color_black	-0.0503113	0.0883889	0.010812	0.00592575
stem-color_blue	-0.00144393	-0.0131765	0.0179709	-0.0187243
stem-color_brown	-0.0417487	0.068513	0.00668689	0.0124134
stem-color_buff	0.00647826	-0.0115234	-0.041282	0.0681308
stem-color_gray	-0.00510632	-0.0258965	0.0307426	-0.0248677
stem-color_green	-0.00378093	-0.0204587	0.029492	-0.0290725
stem-color_none	-0.0502553	0.0824219	-0.00805209	0.0427471
stem-color_orange	0.0270205	-0.015214	-0.0493338	0.0481463
stem-color_pink	-0.00522647	-0.0282474	-0.0185174	0.0620239
stem-color_purple	0.029661	-0.0341899	-0.0429004	0.046311
stem-color_red	0.0590978	-0.0402933	-0.0222824	-0.0371796
stem-color_white	-0.0317755	-0.0261546	0.0418997	0.00357863
stem-color_yellow	0.084826	-0.0360247	-0.0276689	-0.0768522
has-ring_none	0.0384713	-0.000218438	-0.0308831	-0.0150978
has-ring_ring	-0.0384713	0.000218438	0.0308831	0.0150978
ring-type_evanescent	-0.0231412	0.0329287	-0.00397791	0.0238463
ring-type_flaring	-0.0348752	0.0103483	0.0154733	0.0278141
ring-type_grooved	-0.0103896	-0.0311248	0.0343085	-0.0177611
ring-type_large	0.00235315	-0.0334417	0.0125258	-0.00118696
ring-type_movable	0.003269	-0.016485	0.0172414	-0.0234257
ring-type_none	0.0243378	-0.00124647	-0.00566921	-0.0326511
ring-type_pendant	0.017915	-0.0314436	0.00633731	-0.0197001
ring-type_zone	-0.00565658	0.0417787	-0.0456886	0.058125
habitat_grasses	-0.0271657	-0.00746902	0.0483424	-0.0294893
habitat_heaths	0.0268998	-0.0193069	0.0132467	-0.0565463
habitat_leaves	0.00170506	0.00948568	-0.0406994	0.0601698
habitat_meadows	-0.0156422	0.040726	0.00636272	-0.0130781
habitat_paths	0.00689142	-0.0166486	0.0137049	-0.0236582
habitat_urban	-0.0225329	0.0399836	-0.0141369	0.0350913
habitat_waste	-0.00105137	-0.016485	0.0217032	-0.0234257
habitat_woods	0.0175315	-0.00906674	-0.0291516	0.0257193
season_0	1	-0.213682	-0.765503	-0.303649
season_1	-0.213682	1	-0.16745	-0.0664215
season_2	-0.765503	-0.16745	1	-0.237952
season_3	-0.303649	-0.0664215	-0.237952	1
cap-diameter	-0.033213	0.0274621	-0.00533931	0.0481536
stem-height	0.0249918	-0.040375	0.00392004	-0.0215638

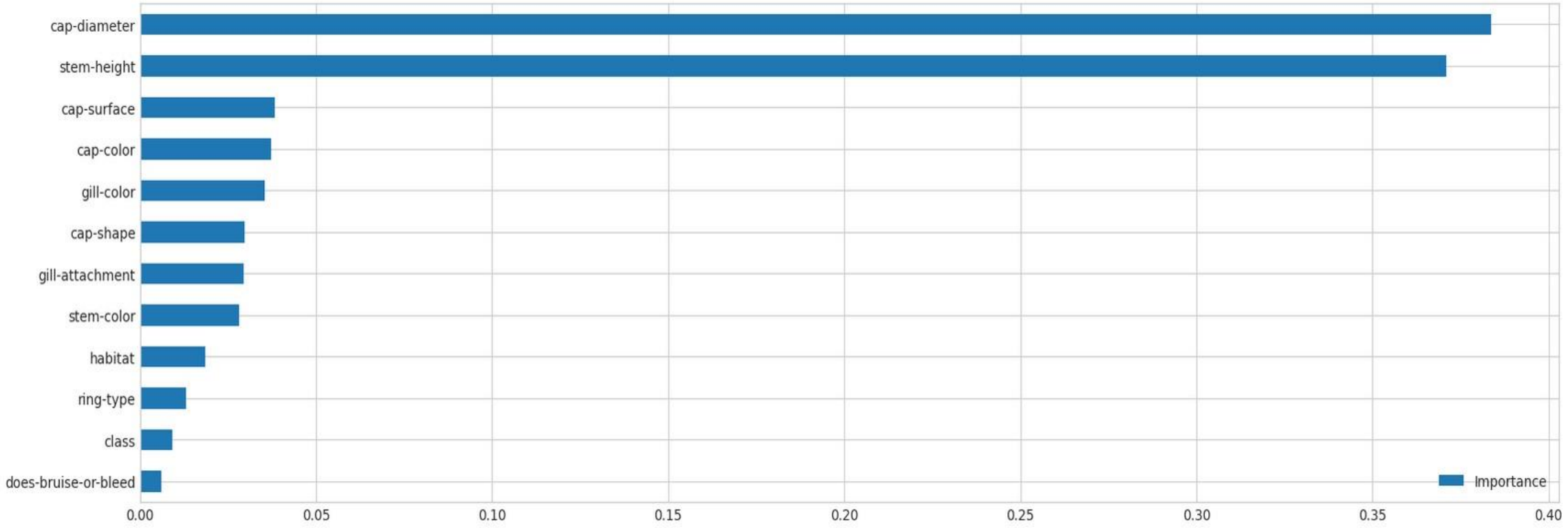
Correlation table of the features with all seasons

Feature Importance

Random Forest Classifier is done to understand Feature Importance

In [346…

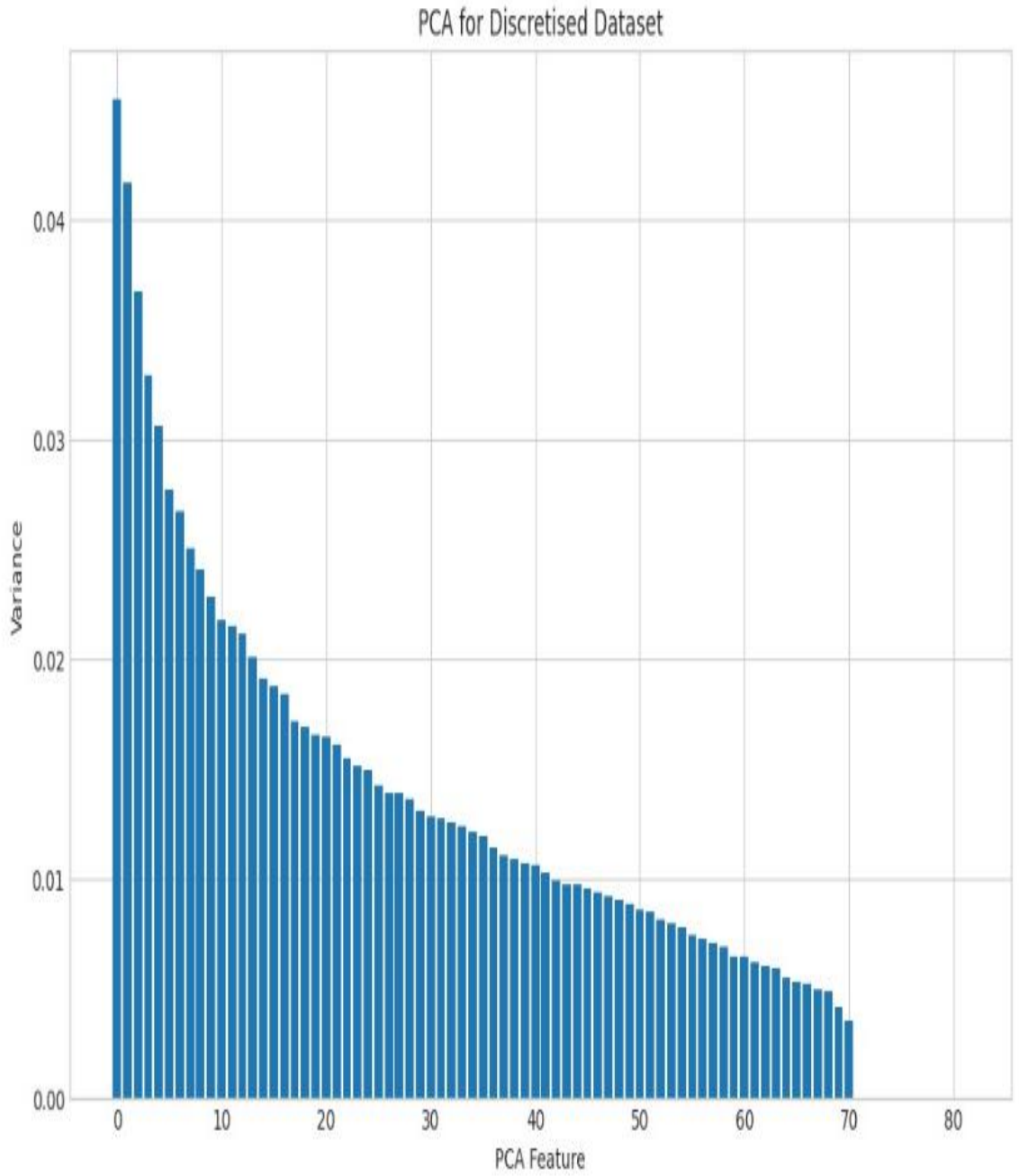
```
# Using Random Forest to gain an insight on Feature Importance
clf = RandomForestClassifier()
clf.fit(dataset_con_enc.drop('season', axis=1), dataset_con_enc['season'])

plt.style.use('seaborn-whitegrid')
importance = clf.feature_importances_
importance = pd.DataFrame(importance, index=dataset_con_enc.drop('season', axis=1).columns, columns=["Importance"])
importance.sort_values(by='Importance', ascending=True).plot(kind='barh', figsize=(20,len(importance)/2));
```

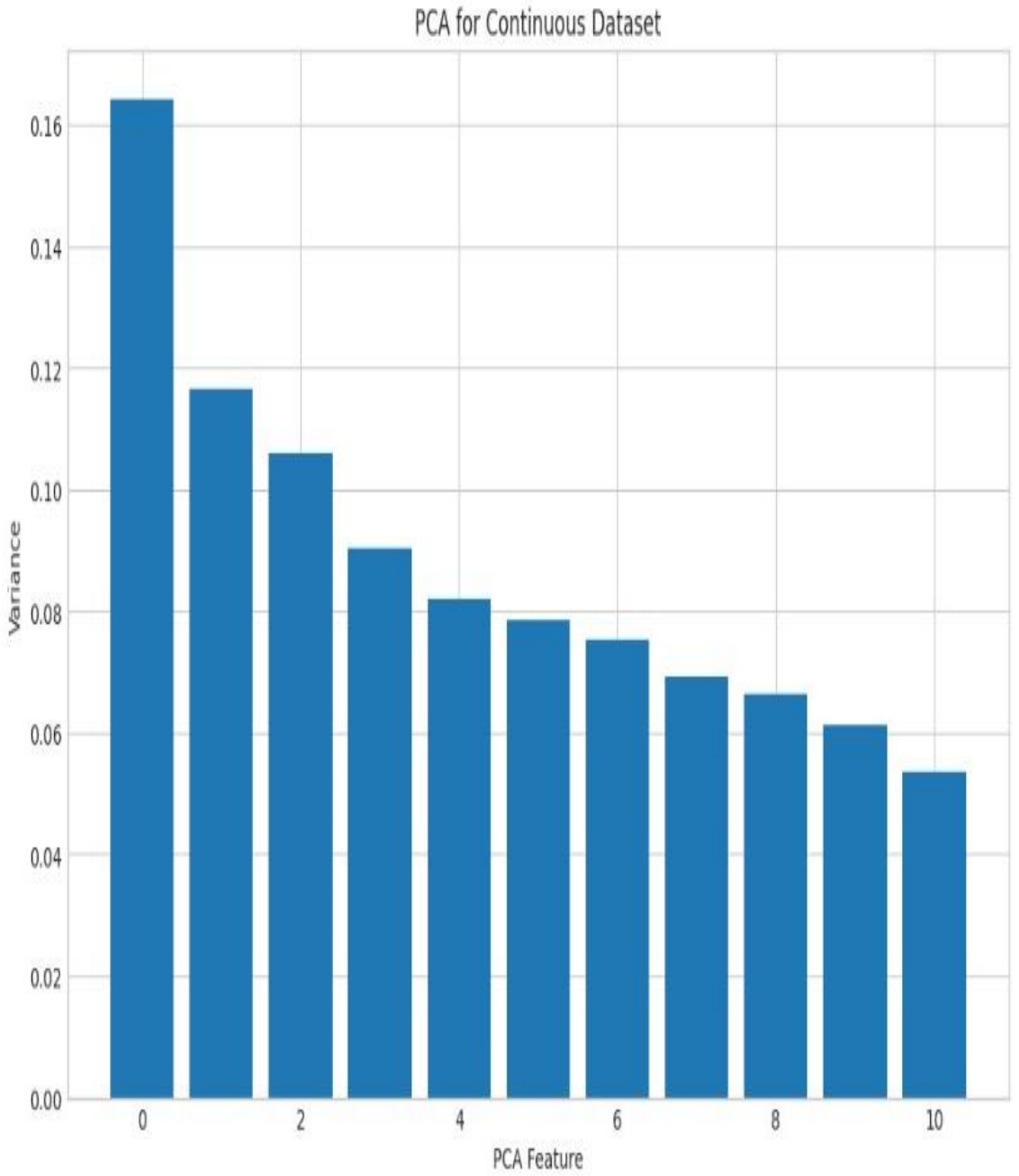


```
# Calculating PCA for both datasets, and graphing the Variance for each feature, per dataset
std_scale = preprocessing.StandardScaler().fit(dataset_bin_enc.drop('season', axis=1))
X = std_scale.transform(dataset_bin_enc.drop('season', axis=1))
pca1 = PCA(n_components=len(dataset_bin_enc.columns)-1)
fit1 = pca1.fit(X)

std_scale = preprocessing.StandardScaler().fit(dataset_con_enc.drop('season', axis=1))
X = std_scale.transform(dataset_con_enc.drop('season', axis=1))
pca2 = PCA(n_components=len(dataset_con_enc.columns)-2)
fit2 = pca2.fit(X)

# Graphing the variance per feature
plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(25,7))

plt.subplot(1, 2, 1)
plt.xlabel('PCA Feature')
plt.ylabel('Variance')
plt.title('PCA for Discretised Dataset')
plt.bar(range(0, fit1.explained_variance_ratio_.size), fit1.explained_variance_ratio_);

plt.subplot(1, 2, 2)
plt.xlabel('PCA Feature')
plt.ylabel('Variance')
plt.title('PCA for Continuous Dataset')
plt.bar(range(0, fit2.explained_variance_ratio_.size), fit2.explained_variance_ratio_);
```

Principal Component Analysis has been used to reduce the dimensionality of feature

From PCA, we decide to go with label encoded data frame as it explains more variance.


PCA for Discretised Dataset
Variance
PCA Feature
0.00
0.01
0.02
0.03
0.04
0
10
20
30
40
50
60
70
80


PCA for Continuous Dataset
Variance
PCA Feature
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0
2
4
6
8
10

From the explained variance ratio we are selecting the dataset with Label Encoded values instead of One Hot Encoded dataset. Since it is explaining more variance than the other.

In [539…
```
# OPTIONS:
# - dataset_bin_enc
# - dataset_con_enc

selected_dataset = dataset_con_enc
selected_dataset.head()
```

Out[539]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-color	stem-height	stem-color	ring-type	habitat	season
0	1	1.619462	2	3	6	1	3	10	3.076705	11	2	7	3
1	1	1.873982	2	3	6	1	3	10	3.385311	11	2	7	2
2	1	1.393432	2	3	6	1	3	10	3.328931	11	2	7	3
3	1	1.412426	3	6	9	1	3	10	2.726555	11	6	7	3
4	1	1.501699	2	6	6	1	3	10	2.952075	11	6	7	3

We have chosen Label Encoded Features. Further, we are splitting the dataset into train and test data.

In [540…
```
# Splitting dataset into train and test data

train = selected_dataset.sample(frac=0.7)
test = selected_dataset.loc[~selected_dataset.index.isin(train.index)]

X_train_w_label = train
X_train = train.drop(['season'], axis=1)
y_train = train['season'].astype('int64')
X_test = test.drop(['season'], axis=1)
y_test = test['season'].astype('int64')
```

In [541…
```
# calculate the fpr and tpr for all thresholds of the classification
def plot_roc_curve(y_test, preds):
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds, pos_label = 2)
    roc_auc = metrics.auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

```
# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, X_test, cv):
    # One Pass
    model = algo.fit(X_train, y_train)
    test_pred = model.predict(X_test)
    if (isinstance(algo, (LogisticRegression,
                          KNeighborsClassifier,
                          GaussianNB,
                          DecisionTreeClassifier,
                          RandomForestClassifier,
                          GradientBoostingClassifier))):
        probs = model.predict_proba(X_test)[:,1]
    else:
        probs = "Not Available"
    acc = round(model.score(X_test, y_test) * 100, 2)
    # CV
    train_pred = model_selection.cross_val_predict(algo,
                                                   X_train,
                                                   y_train,
                                                   cv=cv,
                                                   n_jobs = -1)
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)
    return train_pred, test_pred, acc, acc_cv, probs
```

IMPLEMENTATION OF MACHINE LEARNING MODELS TO TEST AND TRAIN THE DATASET

- LOGISTIC REGRESSION MODEL
- K – NEAREST NEIGHBORS MODEL
- GAUSSIAN NAÏVE BAYES MODEL
- LINEAR SVC MODEL
- STOCHASTIC GRADIENT DESCENT MODEL
- DECISION TREE CLASSIFIER MODEL
- RANDOM FOREST CLASSIFIER MODEL
- GRADIENT BOOSTING TREES MODEL

Logistic Regression

```
# Logistic Regression
start_time = time.time()
train_pred_log, test_pred_log, acc_log, acc_cv_log, probs_log = fit_ml_algo(LogisticRegression(n_jobs = -1),
                                                                            X_train,
                                                                            y_train,
                                                                            X_test,
                                                                            10)
log_time = (time.time() - start_time)
print("Accuracy: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))
```

```
Accuracy: 49.65
Accuracy CV 10-Fold: 49.0
Running Time: 0:00:04.885960
```

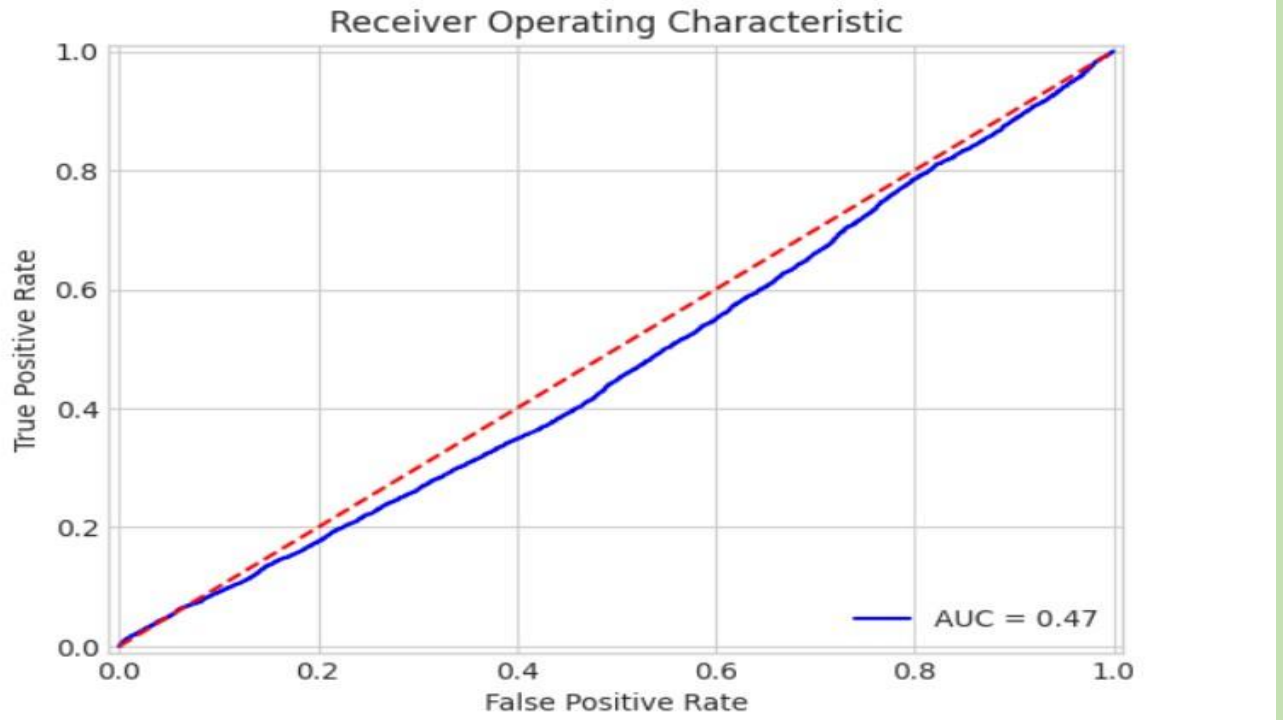
```
#from sklearn.metrics import classification_report
print(metrics.classification_report(y_train, train_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.50      0.94      0.65     21063
           1       0.00      0.00      0.00      1895
           2       0.42      0.06      0.11     16079
           3       0.14      0.02      0.03      3711

    accuracy                           0.49     42748
   macro avg       0.26      0.26      0.20     42748
weighted avg       0.42      0.49      0.36     42748
```

```
print(metrics.classification_report(y_test, test_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.50      0.95      0.66      9114
           1       0.00      0.00      0.00       832
           2       0.45      0.07      0.12      6819
           3       0.10      0.01      0.02      1556

    accuracy                           0.50     18321
   macro avg       0.26      0.26      0.20     18321
weighted avg       0.43      0.50      0.37     18321
```

```
plot_roc_curve(y_test, probs_log)
```



K-Nearest Neighbors

```
# k-Nearest Neighbors
start_time = time.time()
train_pred_knn, test_pred_knn, acc_knn, acc_cv_knn, probs_knn = fit_ml_algo(KNeighborsClassifier(n_neighbors = 3,
                                                                                                  n_jobs = -1),
                                                                            X_train,
                                                                            y_train,
                                                                            X_test,
                                                                            10)

knn_time = (time.time() - start_time)
print("Accuracy: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time: %s" % datetime.timedelta(seconds=knn_time))
```

```
Accuracy: 51.18
Accuracy CV 10-Fold: 51.55
Running Time: 0:00:04.252671
```

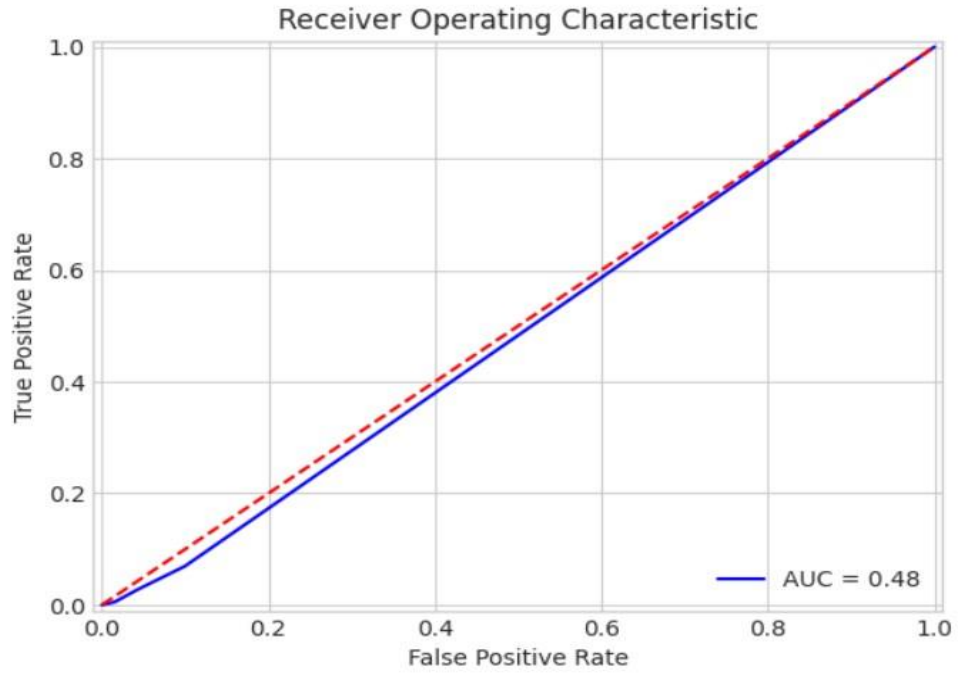
```
print(metrics.classification_report(y_train, train_pred_knn))
```

	precision	recall	f1-score	support
0	0.55	0.59	0.57	21063
1	0.43	0.39	0.41	1895
2	0.49	0.47	0.48	16079
3	0.41	0.35	0.38	3711
accuracy			0.52	42748
macro avg	0.47	0.45	0.46	42748
weighted avg	0.51	0.52	0.51	42748

```
print(metrics.classification_report(y_test, test_pred_knn))
```

	precision	recall	f1-score	support
0	0.56	0.58	0.57	9114
1	0.40	0.38	0.39	832
2	0.48	0.47	0.48	6819
3	0.40	0.34	0.37	1556
accuracy			0.51	18321
macro avg	0.46	0.44	0.45	18321
weighted avg	0.51	0.51	0.51	18321

```
plot_roc_curve(y_test, probs_knn)
```



Gaussian Naïve-Bayes

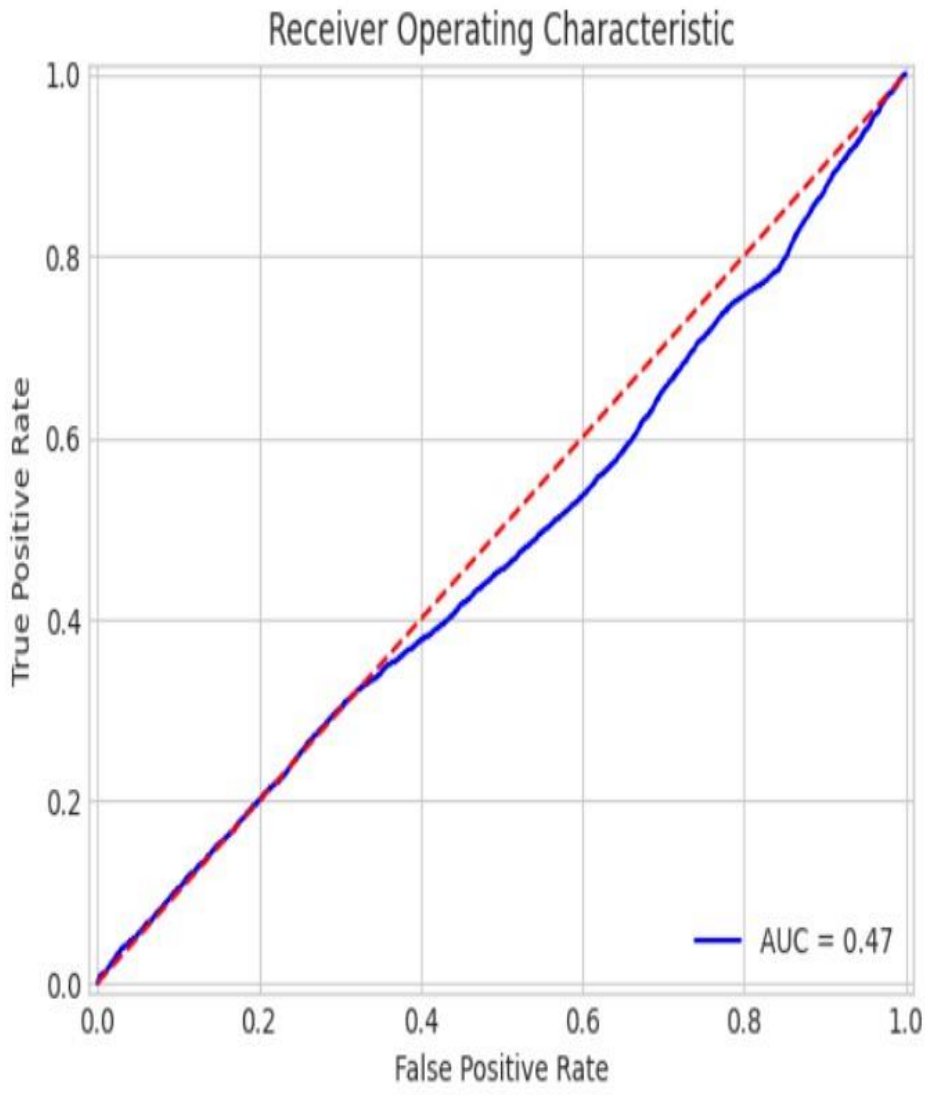
```
# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, test_pred_gaussian, acc_gaussian, acc_cv_gaussian, probs_gau = fit_ml_algo(GaussianNB(),
                                                                                                X_train,
                                                                                                y_train,
                                                                                                X_test,
                                                                                                10)
gaussian_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
print("Running Time: %s" % datetime.timedelta(seconds=gaussian_time))
```

```
Accuracy: 46.89
Accuracy CV 10-Fold: 46.86
Running Time: 0:00:00.699360
```

```
print(metrics.classification_report(y_train, train_pred_gaussian))
print(metrics.classification_report(y_test, test_pred_gaussian))
```

```
              precision    recall  f1-score   support

           0       0.51      0.84      0.63     21063
           1       0.24      0.06      0.09      1895
           2       0.42      0.10      0.17     16079
           3       0.17      0.16      0.17      3711

    accuracy                           0.47     42748
   macro avg       0.33      0.29      0.26     42748
weighted avg       0.43      0.47      0.39     42748

              precision    recall  f1-score   support

           0       0.51      0.84      0.63      9114
           1       0.27      0.06      0.10       832
           2       0.43      0.10      0.17      6819
           3       0.13      0.13      0.13      1556

    accuracy                           0.47     18321
   macro avg       0.34      0.28      0.26     18321
weighted avg       0.44      0.47      0.39     18321
```

```
plot_roc_curve(y_test, probs_gau)
```



Linear SVC

```
# Linear SVC
start_time = time.time()
train_pred_svc, test_pred_svc, acc_linear_svc, acc_cv_linear_svc, _ = fit_ml_algo(LinearSVC(),
                                                                                  X_train,
                                                                                  y_train,
                                                                                  X_test,
                                                                                  10)

linear_svc_time = (time.time() - start_time)
print("Accuracy: %s" % acc_linear_svc)
print("Accuracy CV 10-Fold: %s" % acc_cv_linear_svc)
print("Running Time: %s" % datetime.timedelta(seconds=linear_svc_time))
```

```
Accuracy: 49.52
Accuracy CV 10-Fold: 47.76
Running Time: 0:00:39.804990
```

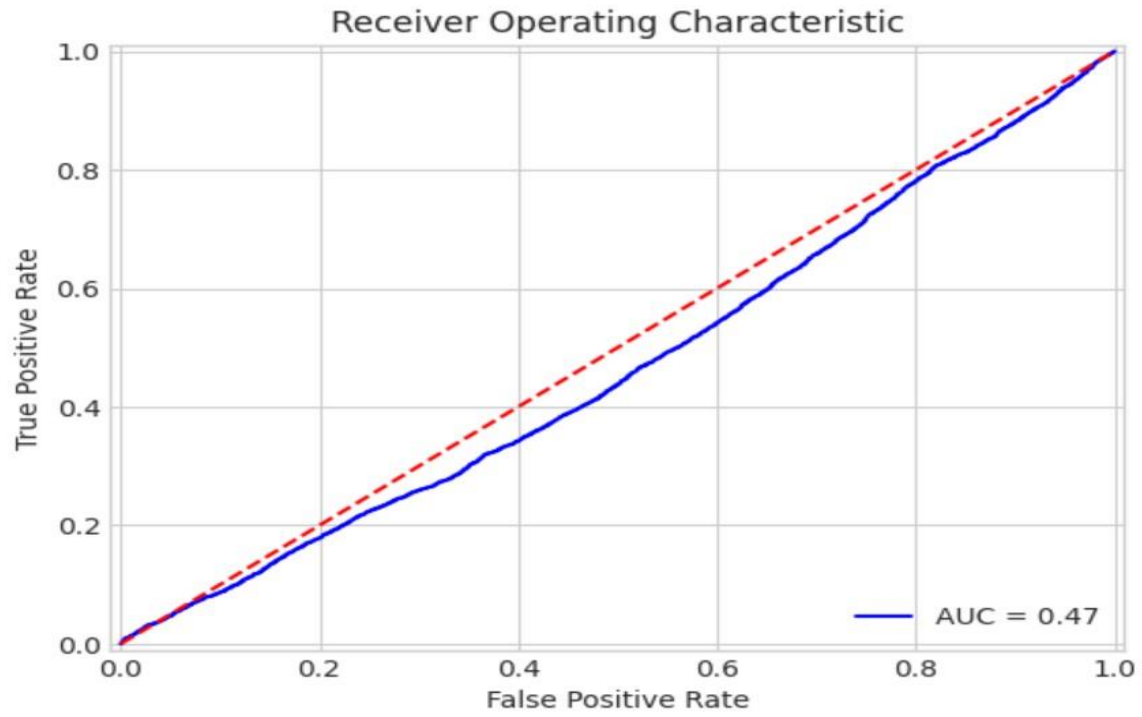
```
print(metrics.classification_report(y_train, train_pred_svc))
```

	precision	recall	f1-score	support
0	0.50	0.83	0.62	21063
1	0.00	0.00	0.00	1895
2	0.39	0.18	0.24	16079
3	0.01	0.00	0.00	3711
accuracy			0.48	42748
macro avg	0.22	0.25	0.22	42748
weighted avg	0.39	0.48	0.40	42748

```
print(metrics.classification_report(y_test, test_pred_svc))
```

	precision	recall	f1-score	support
0	0.50	0.95	0.66	9114
1	0.00	0.00	0.00	832
2	0.41	0.06	0.11	6819
3	0.00	0.00	0.00	1556
accuracy			0.50	18321
macro avg	0.23	0.25	0.19	18321
weighted avg	0.40	0.50	0.37	18321

```
clf = LinearSVC(random_state=0)
y_score1 = clf.fit(X_train, y_train).decision_function(X_test)
y_score = y_score1[:,1]
plot_roc_curve(y_test,y_score)
```



```
# Stochastic Gradient Descent
start_time = time.time()
train_pred_sgd, test_pred_sgd, acc_sgd, acc_cv_sgd, _ = fit_ml_algo(SGDClassifier(n_jobs = -1),
                                                                    X_train,
                                                                    y_train,
                                                                    X_test,
                                                                    10)
sgd_time = (time.time() - start_time)
print("Accuracy: %s" % acc_sgd)
print("Accuracy CV 10-Fold: %s" % acc_cv_sgd)
print("Running Time: %s" % datetime.timedelta(seconds=sgd_time))
```

```
Accuracy: 49.74
Accuracy CV 10-Fold: 46.06
Running Time: 0:00:03.114447
```

Stochastic Gradient Descent

```
print(metrics.classification_report(y_train, train_pred_sgd))
```

```
              precision    recall  f1-score   support

           0       0.50      0.69      0.58     21063
           1       0.23      0.00      0.01      1895
           2       0.38      0.32      0.35     16079
           3       0.20      0.02      0.04      3711

    accuracy                           0.46     42748
   macro avg       0.33      0.26      0.24     42748
weighted avg       0.42      0.46      0.42     42748
```

```
print(metrics.classification_report(y_test, test_pred_sgd))
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67      9114
           1       0.00      0.00      0.00       832
           2       0.50      0.00      0.00      6819
           3       0.01      0.00      0.00      1556

    accuracy                           0.50     18321
   macro avg       0.25      0.25      0.17     18321
weighted avg       0.44      0.50      0.33     18321
```

Decision Tree Classifier

```
# Decision Tree Classifier
start_time = time.time()
train_pred_dt, test_pred_dt, acc_dt, acc_cv_dt, probs_dt = fit_ml_algo(DecisionTreeClassifier(),
                                                                       X_train,
                                                                       y_train,
                                                                       X_test,
                                                                       10)
dt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))
```

```
Accuracy: 51.47
Accuracy CV 10-Fold: 50.67
Running Time: 0:00:01.283718
```

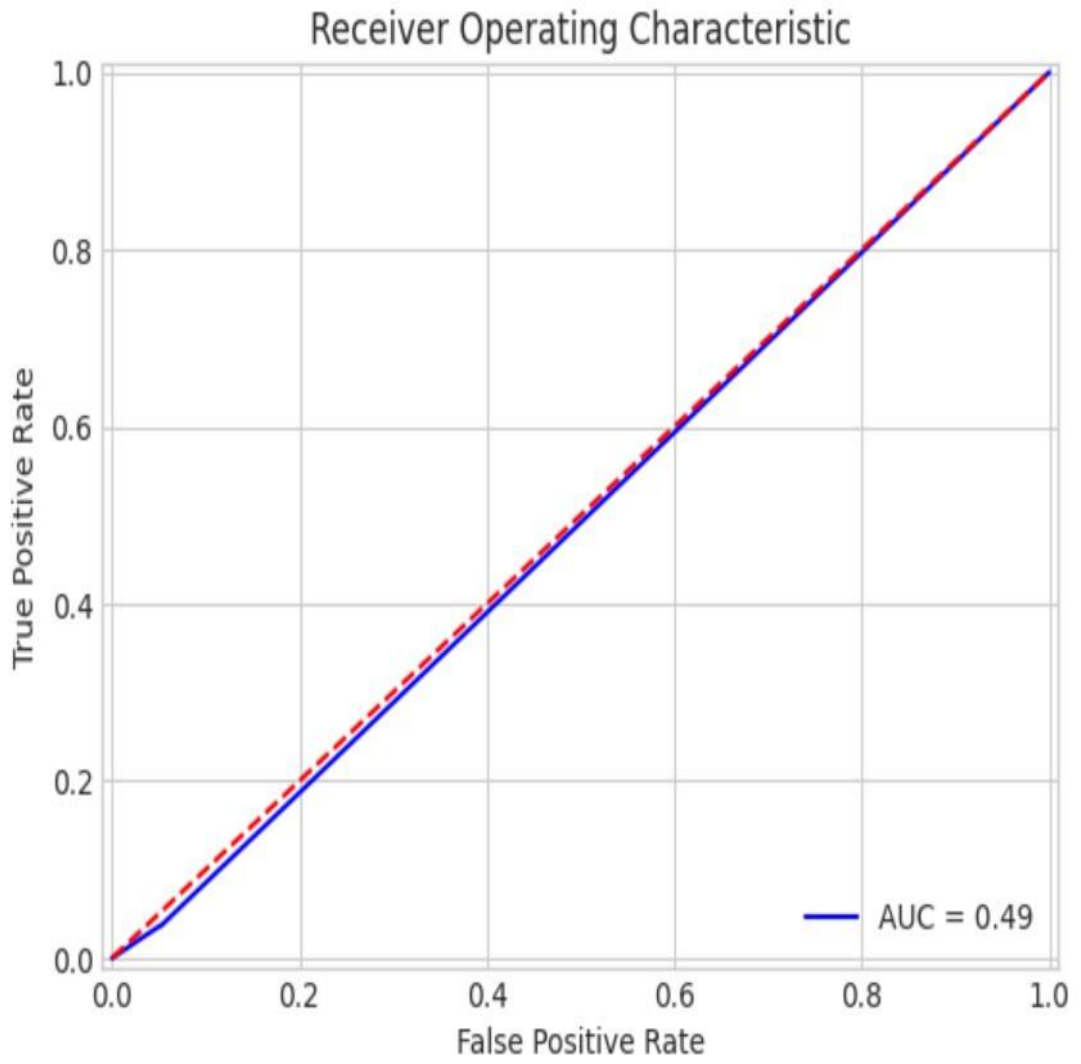
```
print(metrics.classification_report(y_train, train_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.56      0.56      0.56     21063
           1       0.40      0.40      0.40      1895
           2       0.48      0.48      0.48     16079
           3       0.38      0.39      0.39      3711

    accuracy                           0.51     42748
   macro avg       0.46      0.46      0.46     42748
weighted avg       0.51      0.51      0.51     42748
```

```
print(metrics.classification_report(y_test, test_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.57      0.57      0.57      9114
           1       0.40      0.40      0.40       832
           2       0.48      0.49      0.49      6819
           3       0.39      0.38      0.38      1556

    accuracy                           0.51     18321
   macro avg       0.46      0.46      0.46     18321
weighted avg       0.52      0.51      0.51     18321
```

```
plot_roc_curve(y_test, probs_dt)
```



```python
#Random Forest Classifier - Random Search for Hyperparameters

# Utility function to report best scores
def report(results, n_top=5):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")

# Specify parameters and distributions to sample from
param_dist = {"max_depth": [10, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 20),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# Run Randomized Search
n_iter_search = 10
rfc = RandomForestClassifier(n_estimators=10)
random_search = RandomizedSearchCV(rfc,
                                   n_jobs = -1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(random_search.cv_results_)
```

Random Forest Classifier

```python
# Random Forest Classifier
start_time = time.time()
rfc = RandomForestClassifier(n_estimators=10,
                             min_samples_leaf=2,
                             min_samples_split=17,
                             criterion='gini',
                             max_features=8)
train_pred_rf, test_pred_rf, acc_rf, acc_cv_rf, probs_rf = fit_ml_algo(rfc,
                                                                       X_train,
                                                                       y_train,
                                                                       X_test,
                                                                       10)
rf_time = (time.time() - start_time)
print("Accuracy: %s" % acc_rf)
print("Accuracy CV 10-Fold: %s" % acc_cv_rf)
print("Running Time: %s" % datetime.timedelta(seconds=rf_time))
```

```
Accuracy: 51.81
Accuracy CV 10-Fold: 51.54
Running Time: 0:00:02.008115
```

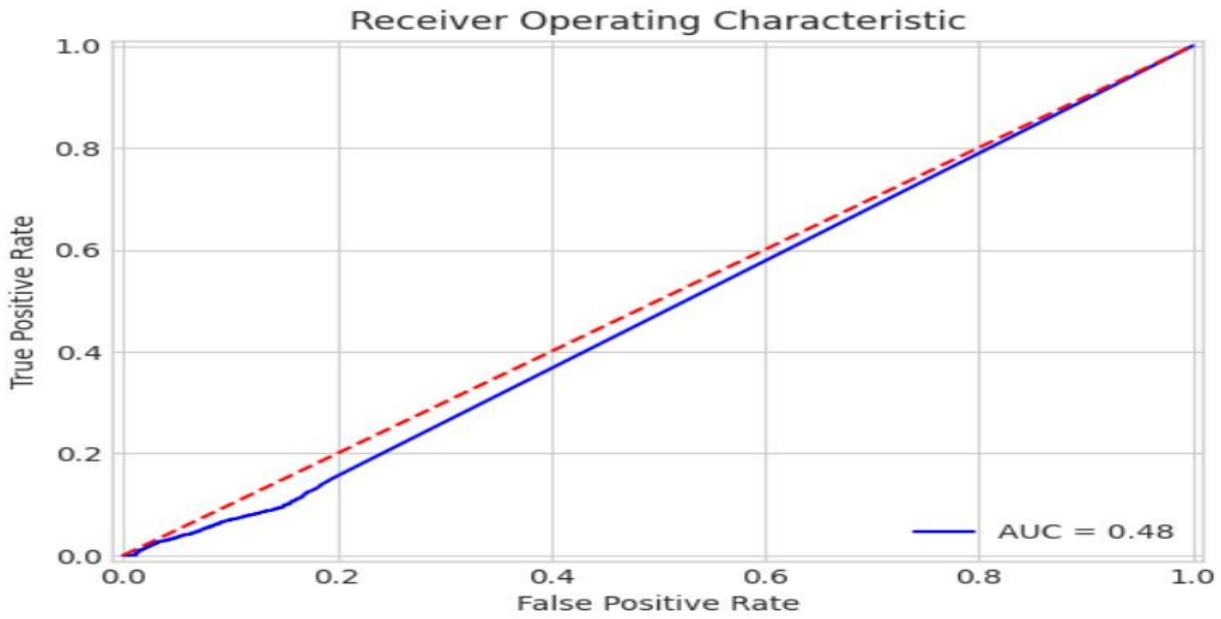
```python
print(metrics.classification_report(y_train, train_pred_rf))
```

	precision	recall	f1-score	support
0	0.57	0.57	0.57	21063
1	0.41	0.39	0.40	1895
2	0.48	0.49	0.49	16079
3	0.39	0.36	0.38	3711
accuracy			0.52	42748
macro avg	0.46	0.45	0.46	42748
weighted avg	0.51	0.52	0.51	42748

```python
print(metrics.classification_report(y_test, test_pred_rf))
```

	precision	recall	f1-score	support
0	0.58	0.57	0.58	9114
1	0.42	0.41	0.41	832
2	0.48	0.49	0.49	6819
3	0.39	0.36	0.38	1556
accuracy			0.52	18321
macro avg	0.47	0.46	0.46	18321
weighted avg	0.52	0.52	0.52	18321

```python
plot_roc_curve(y_test, probs_rf)
```



Gradient Boosting Trees

```
# Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, test_pred_gbt, acc_gbt, acc_cv_gbt, probs_gbt = fit_ml_algo(GradientBoostingClassifier(),
                                                                            X_train,
                                                                            y_train,
                                                                            X_test,
                                                                            10)
gbt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gbt)
print("Accuracy CV 10-Fold: %s" % acc_cv_gbt)
print("Running Time: %s" % datetime.timedelta(seconds=gbt_time))
```

```
Accuracy: 51.6
Accuracy CV 10-Fold: 51.24
Running Time: 0:00:30.540926
```

```
print(metrics.classification_report(y_train, train_pred_gbt))
```

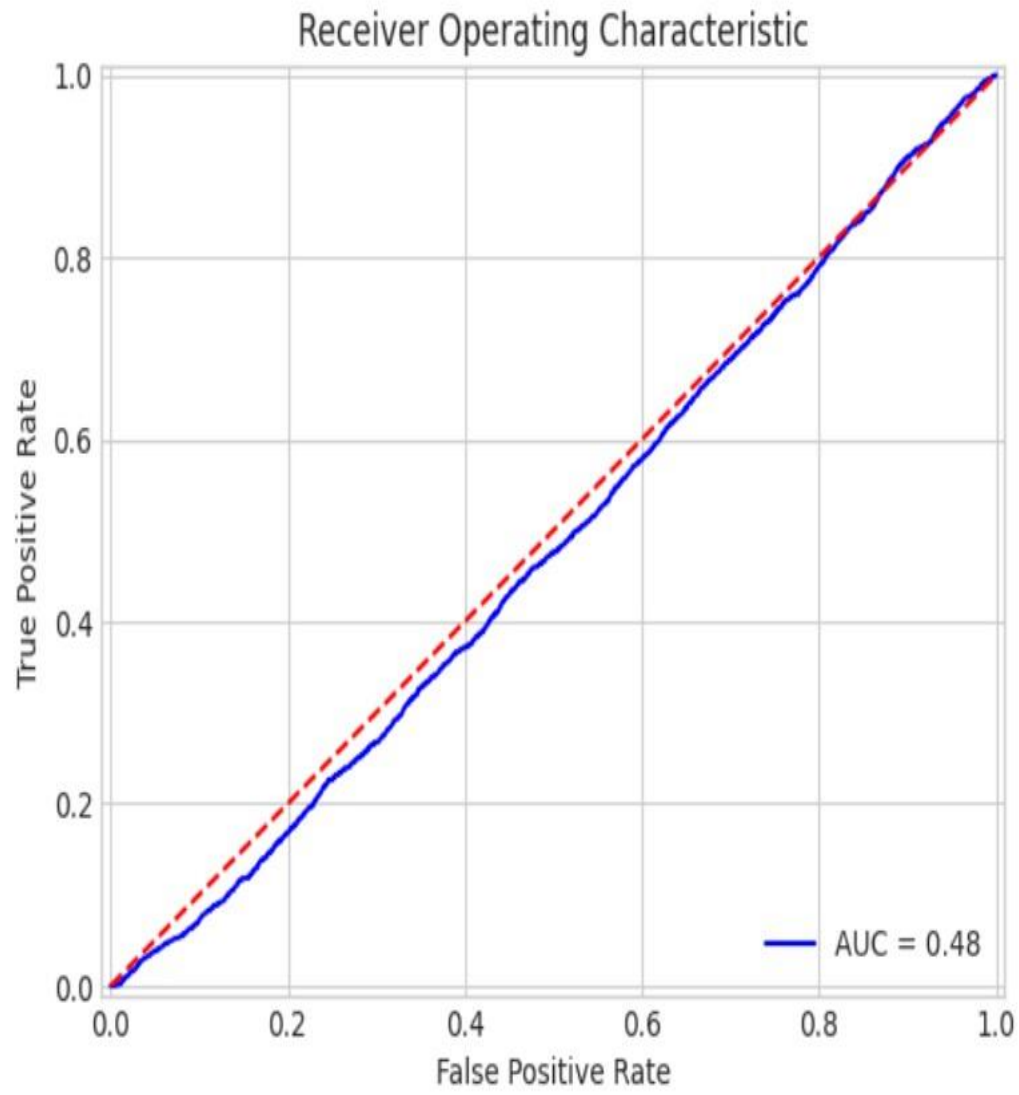
```
              precision    recall  f1-score   support

           0       0.53      0.76      0.63     21063
           1       0.66      0.15      0.24      1895
           2       0.47      0.31      0.37     16079
           3       0.42      0.15      0.22      3711

    accuracy                           0.51     42748
   macro avg       0.52      0.34      0.36     42748
weighted avg       0.50      0.51      0.48     42748
```

```
print(metrics.classification_report(y_test, test_pred_gbt))
```

```
              precision    recall  f1-score   support

           0       0.54      0.75      0.63      9114
           1       0.70      0.19      0.29       832
           2       0.46      0.33      0.39      6819
           3       0.39      0.13      0.19      1556

    accuracy                           0.52     18321
   macro avg       0.52      0.35      0.38     18321
weighted avg       0.51      0.52      0.49     18321
```

Gradient Boosting Trees

```
plot_roc_curve(y_test, probs_gbt)
```



Ranking Results

```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree', 'Gradient Boosting Trees'],
    'Score': [
        acc_knn,
        acc_log,
        acc_rf,
        acc_gaussian,
        acc_sgd,
        acc_linear_svc,
        acc_dt,
        acc_gbt
    ]})
models.sort_values(by='Score', ascending=False)
```

	Model	Score
2	Random Forest	51.81
7	Gradient Boosting Trees	51.60
6	Decision Tree	51.47
0	KNN	51.18
4	Stochastic Gradient Decent	49.74
1	Logistic Regression	49.65
5	Linear SVC	49.52
3	Naive Bayes	46.89

```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree', 'Gradient Boosting Trees'],
    'Score': [
        acc_cv_knn,
        acc_cv_log,
        acc_cv_rf,
        acc_cv_gaussian,
        acc_cv_sgd,
        acc_cv_linear_svc,
        acc_cv_dt,
        acc_cv_gbt    ]})
models.sort_values(by='Score', ascending=False)
```

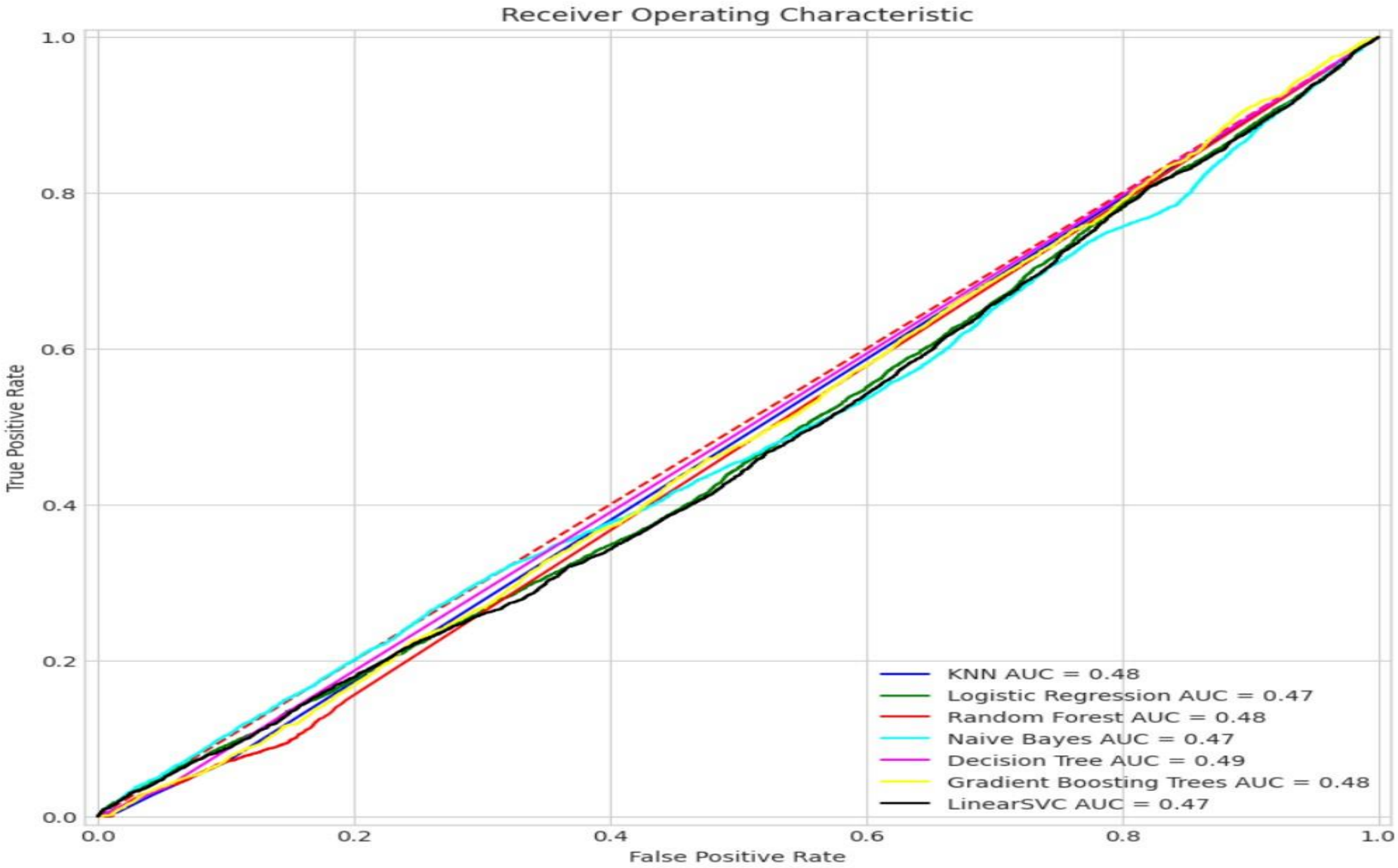
	Model	Score
0	KNN	51.55
2	Random Forest	51.54
7	Gradient Boosting Trees	51.24
6	Decision Tree	50.67
1	Logistic Regression	49.00
5	Linear SVC	47.76
3	Naive Bayes	46.86
4	Stochastic Gradient Decent	46.06

Accuracy for 10-fold cross validation

Ranking Results for the Machine Learning Models

```
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(10,10))

models = [
    'KNN',
    'Logistic Regression',
    'Random Forest',
    'Naive Bayes',
    'Decision Tree',
    'Gradient Boosting Trees',
'LinearSVC']
probs = [
    probs_knn,
    probs_log,
    probs_rf,
    probs_gau,
    probs_dt,
    probs_gbt,
y_score]
colors = [
    'blue',
    'green',
    'red',
    'cyan',
    'magenta',
    'yellow',
'black']

plt.title('Receiver Operating Characteristic')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.01])
plt.ylim([-0.01, 1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

def plot_roc_curves(y_test, prob, model):
    fpr, tpr, threshold = metrics.roc_curve(y_test, prob, pos_label = 2)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, 'b', label = model + ' AUC = %0.2f' % roc_auc, color=color
    plt.legend(loc = 'lower right')

for i, model in list(enumerate(models)):
    plot_roc_curves(y_test, probs[i], models[i])

plt.show()
```

ROC CURVE FOR ALL THE MACHINE LEARNING MODELS



OVERCOMING DIFFICULTIES WITH IMBALANCED DATA

- The AUC graphs and prediction models failed to produce good accuracy scores and better F1 scores.
- This led to an interpretation that the prediction models were not as accurate because the data was imbalanced and not highly co-related.
- For dealing with this problem, Oversampling and Under sampling on our datasets and Prediction Models were performed to increase the level of accuracy.

UNDERSAMPLING

UNDERSAMPLING

[741…]
```
from imblearn.under_sampling import RandomUnderSampler
```

[742…]
```
rus = RandomUnderSampler(random_state=0)
rus.fit(X, y)
X_resampled, y_resampled = rus.fit_resample(X, y)
```

[743…]
```
X_train_us, X_test_us, y_train_us, y_test_us = train_test_split(X_resampled, y_resampled, test_size=0.30,
                                                    random_state=15,
                                                    stratify = None)
```

[744…]
```
X_train_us.shape
```

t[744]:
```
(7635, 12)
```

[745…]
```
# Use the random grid to search for best hyperparameters
clf3_us = RandomForestClassifier()
rf_random_us = RandomizedSearchCV(estimator = clf3_us, param_distributions = random_grid, n_iter = 10, cv = 3, verbose=2, random_state=42, n_jobs = -1)
rf_random_us.fit(X_train_us, y_train_us)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

t[745]:

RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier

[746…]
```
rf_random_us.best_params_
```

t[746]:
```
{'n_estimators': 200,
 'min_samples_split': 2,
 'min_samples_leaf': 4,
 'max_depth': 22,
 'bootstrap': True}
```

[747…]
```
best_random2 = rf_random_us.best_estimator_
```

Under sampling was done on the data using imblearn RandomUnderSampler

In [747…]
```
best_random2 = rf_random_us.best_estimator_
```

In [748…]
```
best_random2.score(X_train,y_train)
```

Out[748]:
```
0.4912510526808272
```

In [749…]
```
predictions = best_random2.predict(X_test)
```

In [750…]
```
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, predictions))
```

```
ACCURACY OF THE MODEL:  0.4950057311282135
```

In [751…]
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.50      0.99      0.66      9114
           1       0.00      0.00      0.00       832
           2       0.40      0.01      0.02      6819
           3       0.18      0.02      0.03      1556

    accuracy                           0.50     18321
   macro avg       0.27      0.25      0.18     18321
weighted avg       0.41      0.50      0.34     18321
```

- Under sampling shows lower accuracy.
- Even combining the two minority classes and then running the models will not make any significant difference as we have highly imbalanced data.
- To tackle this, we tried Over sampling the minority classes.
- We chose SMOTE which is **Synthetic Minority Oversampling Technique.**
- **Why smote over others ?**
- **Because it uses k nearest approach to add synthetic data rather than just replicating the existing data.**

SMOTE/OVERSAMPLING

SMOTE / Oversampling

In [752…
```
from sklearn.model_selection import train_test_split
selected_dataset.columns
```

Out[752]:
```
Index(['class', 'cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
       'does-bruise-or-bleed', 'gill-attachment', 'gill-color', 'stem-height',
       'stem-color', 'ring-type', 'habitat', 'season'],
      dtype='object')
```

In [753…
```
PredictorCol=['class', 'cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
       'does-bruise-or-bleed', 'gill-attachment', 'gill-color',
       'stem-height', 'stem-color', 'ring-type',
       'habitat']
TargetCol='season'

X = selected_dataset[PredictorCol].values
y = selected_dataset[TargetCol].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=12)
```

In [754…
```
from imblearn.over_sampling import SMOTE

smote =SMOTE(random_state=10)
X_Smote,Y_smote=smote.fit_resample(X,y)
```

In [755…
```
print("X smote",X_Smote.shape)
print("Y smote",Y_smote.shape)
```

```
X smote (120708, 12)
Y smote (120708,)
```

In [756…
```
X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_Smote, Y_smote, test_size=0.30,
                                                    random_state=15,
                                                    stratify = None)
```

Oversampling/ SMOTE is Method for balancing class distribution using synthetic minority oversampling by randomly increasing minority class examples by replicating them

RANDOM FOREST ON OVERSAMPLED DATA

Random Forest on Oversampled data

In [757…

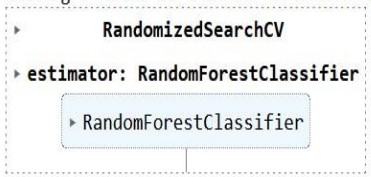
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 600, num = 10)]

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 50, num = 11)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]

# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [758…

```
# Use the random grid to search for best hyperparameters

clf3_sm = RandomForestClassifier()
rf_random_sm = RandomizedSearchCV(estimator = clf3_sm, param_distributions = random_grid, n_iter = 10, cv = 3, verbose=2, random_state=42, n_jobs = -1)
rf_random_sm.fit(X_train_sm, y_train_sm)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Out[758]:



In [594…

```
rf_random_sm.best_params_
```

Out[594]:

```
{'n_estimators': 466,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_depth': 30,
 'bootstrap': True}
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.3s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   4.2s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.6s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.8s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.2s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.3s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.4s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.4s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.5s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.6s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.6s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.7s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.7s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.5s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.6s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.6s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   2.8s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   2.9s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   2.9s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   3.0s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   3.0s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.0s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   3.1s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.2s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   3.3s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.4s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.5s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.6s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   3.7s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   3.8s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   3.8s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.3s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.4s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   5.2s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.8s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  11.9s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.7s
```

```
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.4s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.8s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.5s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.5s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.6s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.6s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   3.1s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=  12.5s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   2.9s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=  12.6s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.2s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=  12.9s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.8s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=  23.3s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.7s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=  23.5s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.8s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=  23.9s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.3s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=  26.1s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.2s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=  26.4s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.2s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=  26.9s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.4s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=  29.6s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.4s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=  30.0s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.6s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=  30.4s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.5s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=  30.6s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=  30.8s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.3s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=  31.1s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   3.5s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=  34.8s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   3.4s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=  34.8s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   3.4s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=  35.0s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.7s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=  36.7s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   2.9s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=  37.2s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   2.9s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=  37.5s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.8s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=  50.4s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.4s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=  50.6s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.3s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=  51.4s
```

In [759]:
```
best_random = rf_random_sm.best_estimator_
best_random
```

Out[759]:
```
RandomForestClassifier(max_depth=30, min_samples_split=5, n_estimators=466)
```

In [760]:
```
best_random.score(X_train,y_train)
```

Out[760]:
```
0.8276410592308412
```

In [761]:
```
predictions = best_random.predict(X_test)
```

In [762]:
```
from sklearn import metrics
from sklearn.metrics import(accuracy_score,
                            classification_report,
                            roc_auc_score, roc_curve, auc, precision_recall_curve,
                            confusion_matrix)

print("ACCURACY OF THE MODEL RANDOM Forest: ", metrics.accuracy_score(y_test, predictions))
```

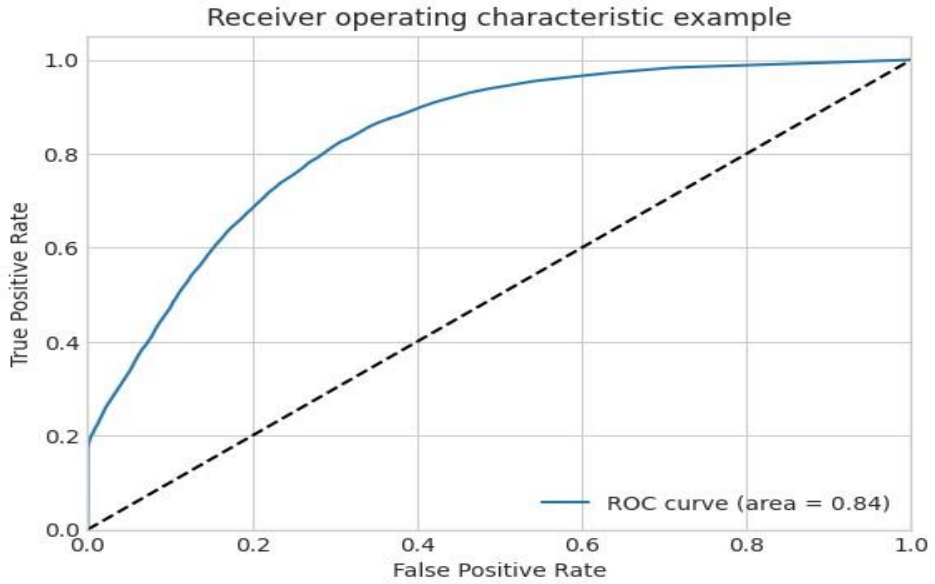
```
ACCURACY OF THE MODEL RANDOM Forest:  0.8234812510234157
```

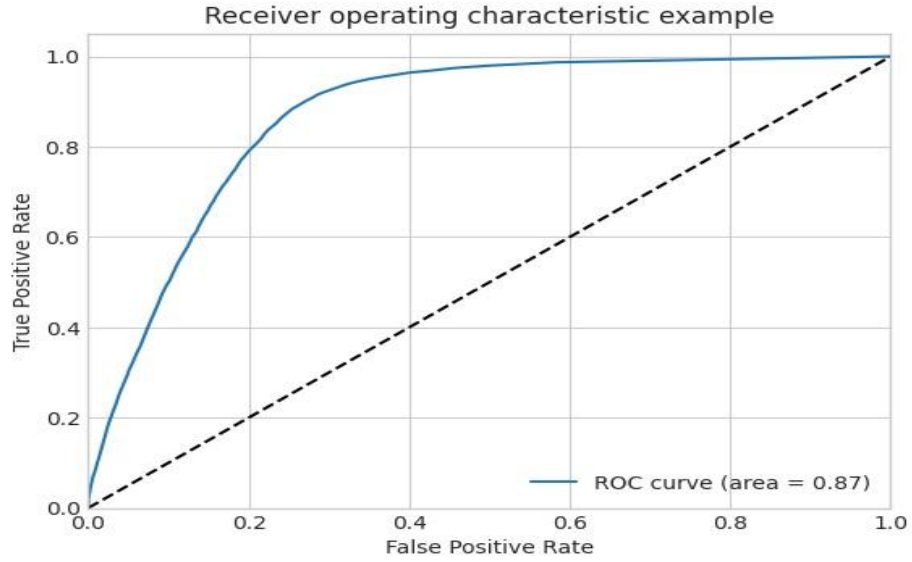
In [763]:
```
print(classification_report(y_test, predictions))
```

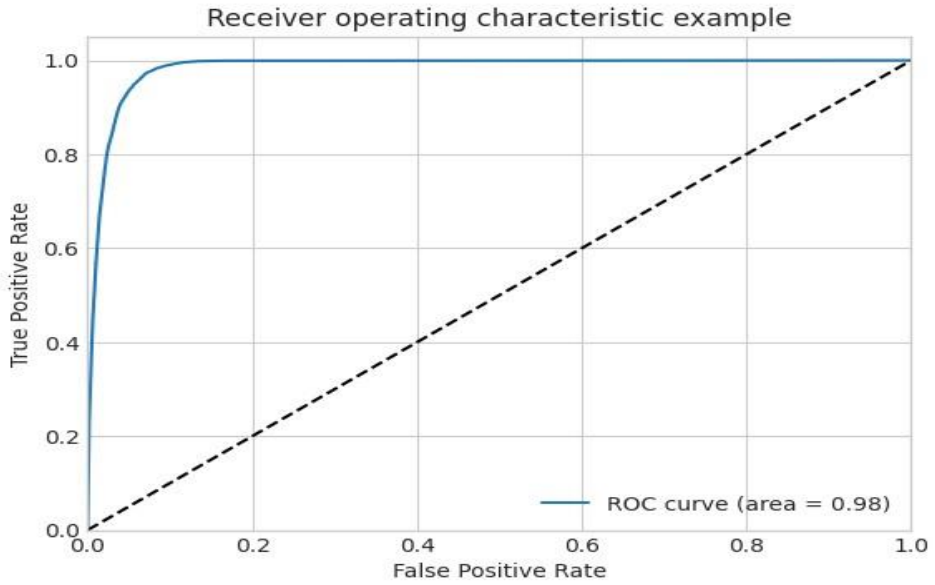
```
              precision    recall  f1-score   support

           0       0.90      0.78      0.84      8951
           1       0.63      0.98      0.77       840
           2       0.82      0.83      0.83      6924
           3       0.67      0.94      0.79      1606

    accuracy                           0.82     18321
   macro avg       0.76      0.88      0.80     18321
weighted avg       0.84      0.82      0.83     18321
```

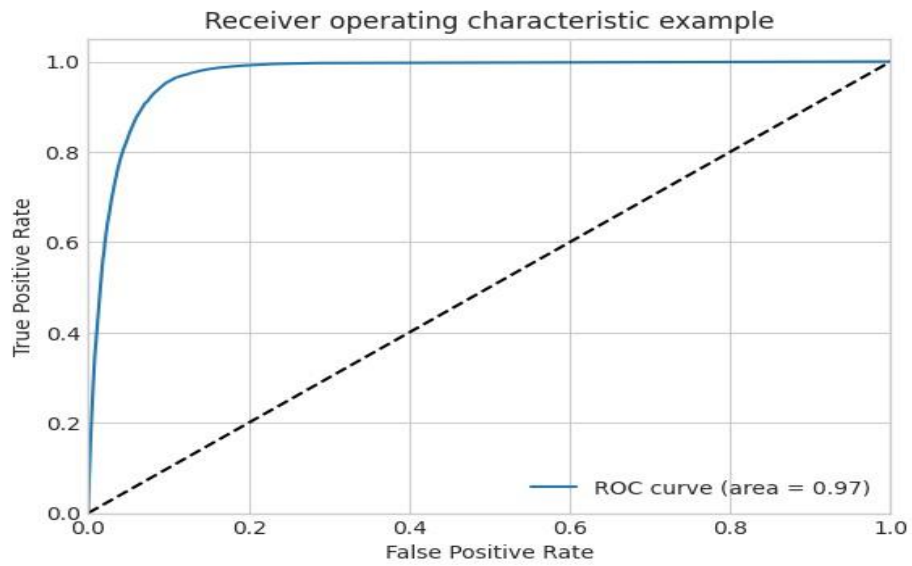
In [765]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize

X, y = X_Smote,Y_smote

y = label_binarize(y, classes=[0,1,2,3])
n_classes = 4

# shuffle and split training and test sets
X_train, X_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.33, random_state=0)

# # classifier
clf = OneVsRestClassifier(RandomForestClassifier(random_state=0))
y_score = clf.fit(X_train, y_train).predict_proba(X_test)


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

ROC CURVE FOR RANDOM FOREST ON OVERSAMPLED DATA









autumn' is '0' - top left, spring is '1' - bottom left, summer is '2' - top right, winter is '3' -bottom right

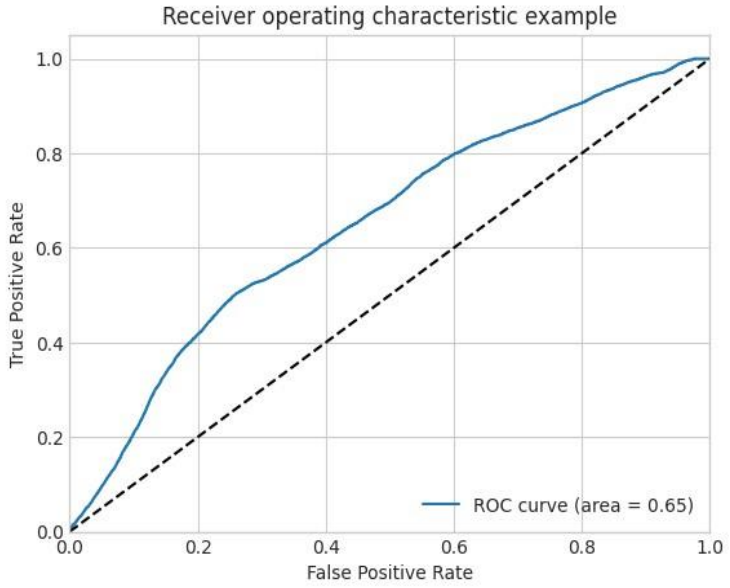
LOGISTIC REGRESSION ON OVERSAMPLED DATA

Logistic Regression on oversampled data

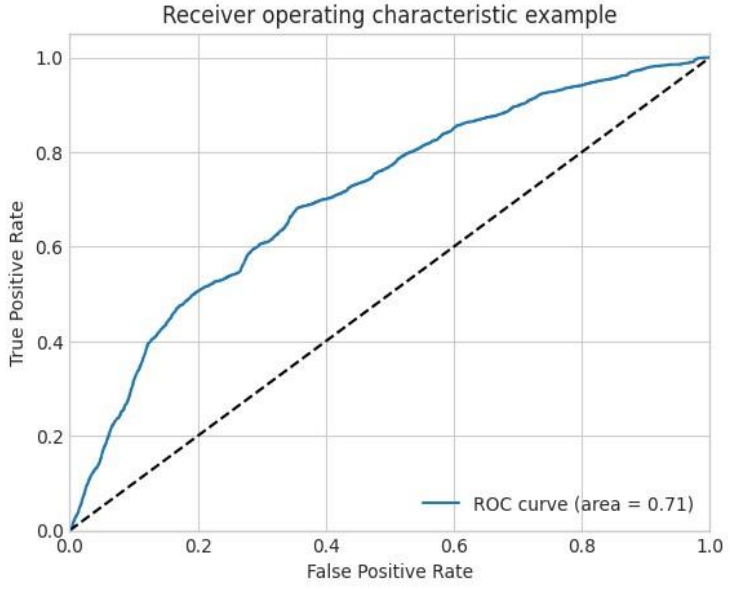
In [766…

```
# # classifier
clf = OneVsRestClassifier(LogisticRegression(random_state=0))
y_score = clf.fit(X_train, y_train).predict_proba(X_test)

predictions = clf.predict(X_test)
predictionstrain = clf.predict(X_train)
print("ACCURACY OF THE LOGISTIC REGRESSION MODEL:", metrics.accuracy_score(y_test, predictions))
print('\n\nTraining data metrics\n',metrics.classification_report(y_train, predictionstrain))
print('Testing data metrics\n',metrics.classification_report(y_test, predictions))


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

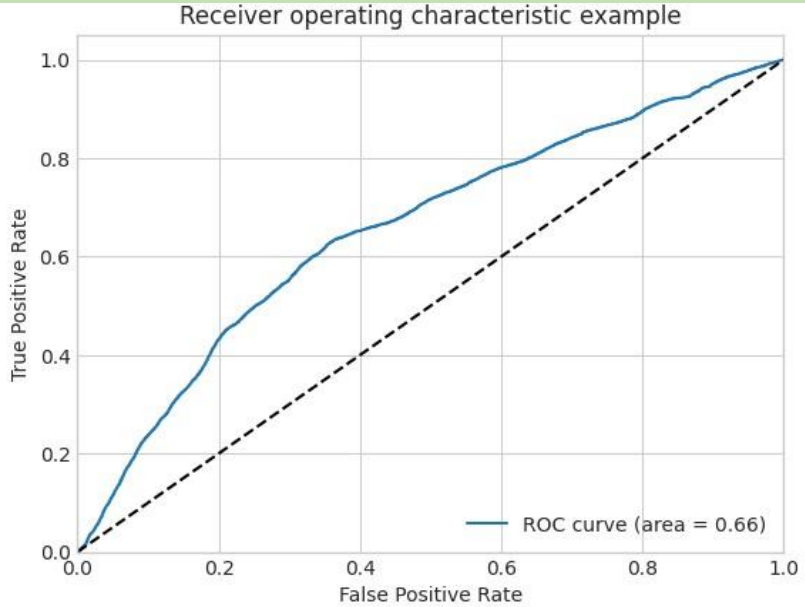
```
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.2s
ACCURACY OF THE LOGISTIC REGRESSION MODEL: 0.03770648189988452


Training data metrics
              precision    recall  f1-score   support

           0       0.41      0.04      0.06     20255
           1       0.52      0.11      0.18     20197
           2       0.38      0.04      0.08     20179
           3       0.71      0.01      0.02     20243

   micro avg       0.47      0.05      0.09     80874
   macro avg       0.51      0.05      0.09     80874
weighted avg       0.51      0.05      0.09     80874
 samples avg       0.04      0.05      0.05     80874

[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.2s
Testing data metrics
              precision    recall  f1-score   support

           0       0.39      0.03      0.06      9922
           1       0.54      0.12      0.19      9980
           2       0.39      0.04      0.08      9998
           3       0.67      0.01      0.02      9934

   micro avg       0.47      0.05      0.09     39834
   macro avg       0.50      0.05      0.09     39834
weighted avg       0.50      0.05      0.09     39834
 samples avg       0.04      0.05      0.05     39834
```



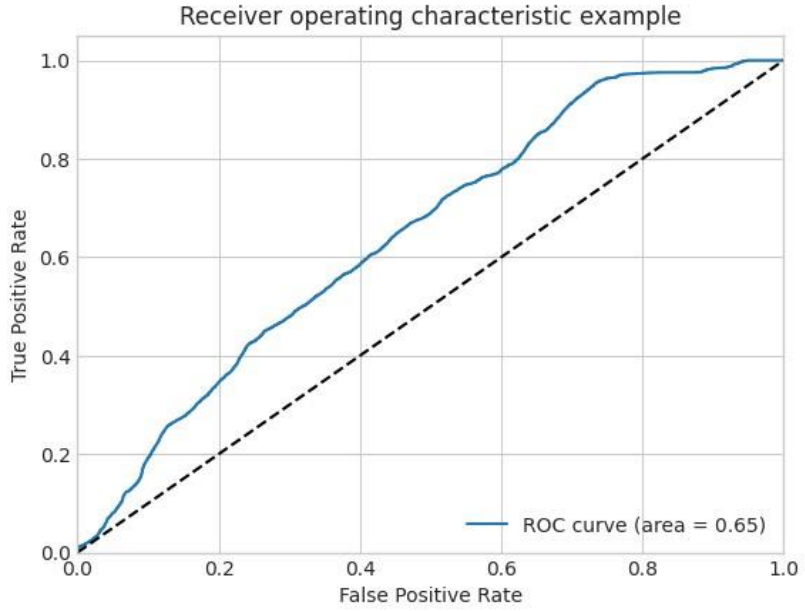

```
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.3s
```




```
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.4s
```




```
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.1s
```



autumn' is '0’ - top left, spring is ‘1’ - bottom left, summer is ‘2’ - top right, winter is ‘3’ -bottom right

```
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.7s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.2s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.3s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.5s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.3s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.5s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.2s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   3.1s
[CV] END bootstrap=False, max_depth=26, min_samples_leaf=2, min_samples_split=5, n_estimators=288; total time=   2.6s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.2s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.8s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=1, min_samples_split=5, n_estimators=422; total time=   2.8s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   2.9s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.5s
[CV] END bootstrap=False, max_depth=18, min_samples_leaf=2, min_samples_split=2, n_estimators=377; total time=   3.5s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.0s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   3.0s
[CV] END bootstrap=False, max_depth=None, min_samples_leaf=4, min_samples_split=2, n_estimators=422; total time=   3.2s
[CV] END bootstrap=True, max_depth=30, min_samples_leaf=1, min_samples_split=5, n_estimators=466; total time=   3.2s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   3.8s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   4.0s
[CV] END bootstrap=False, max_depth=14, min_samples_leaf=4, min_samples_split=2, n_estimators=555; total time=   4.0s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.4s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.7s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=   4.9s
```

KNN ON OVERSAMPLED DATA

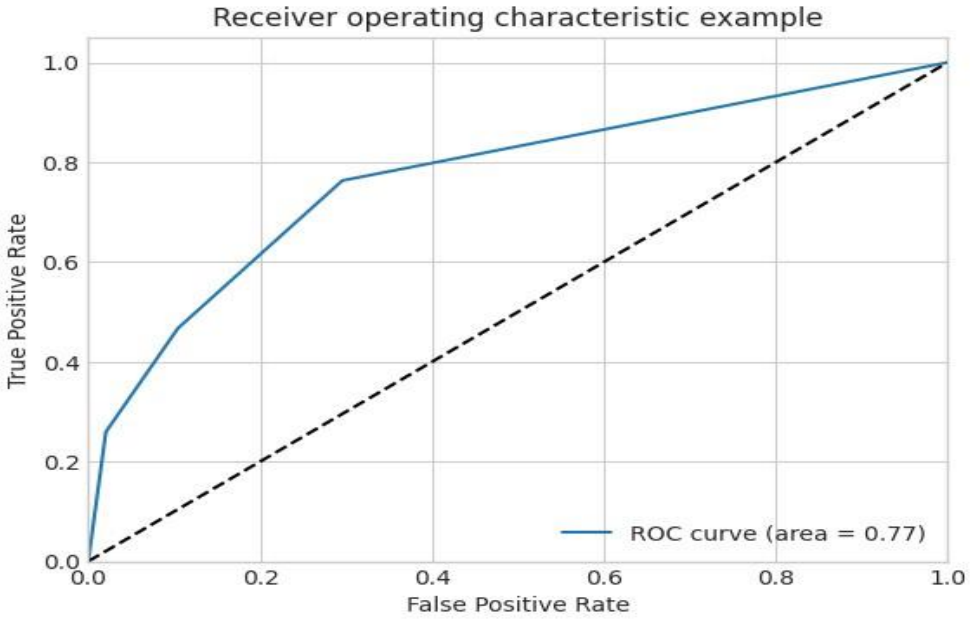
KNN on oversampled data

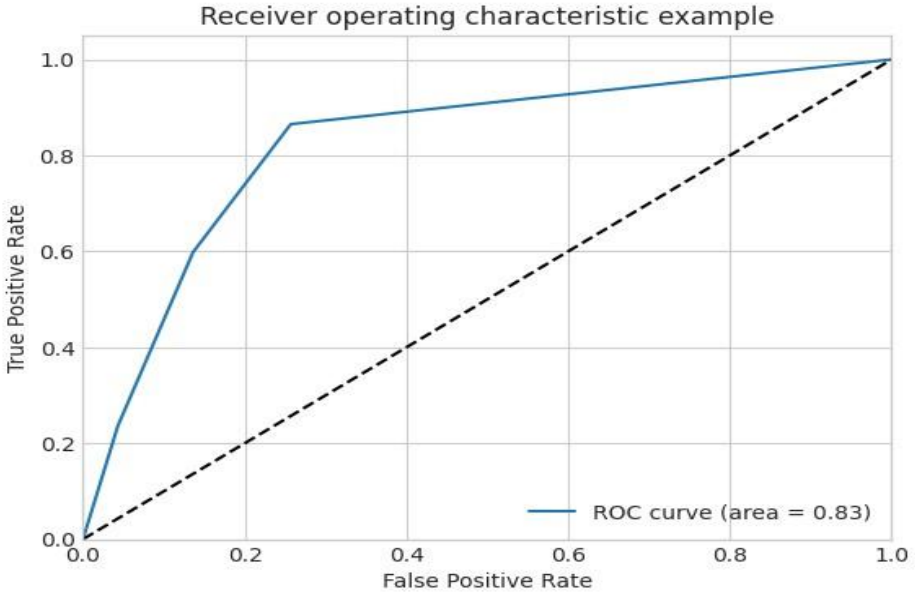
In [767…

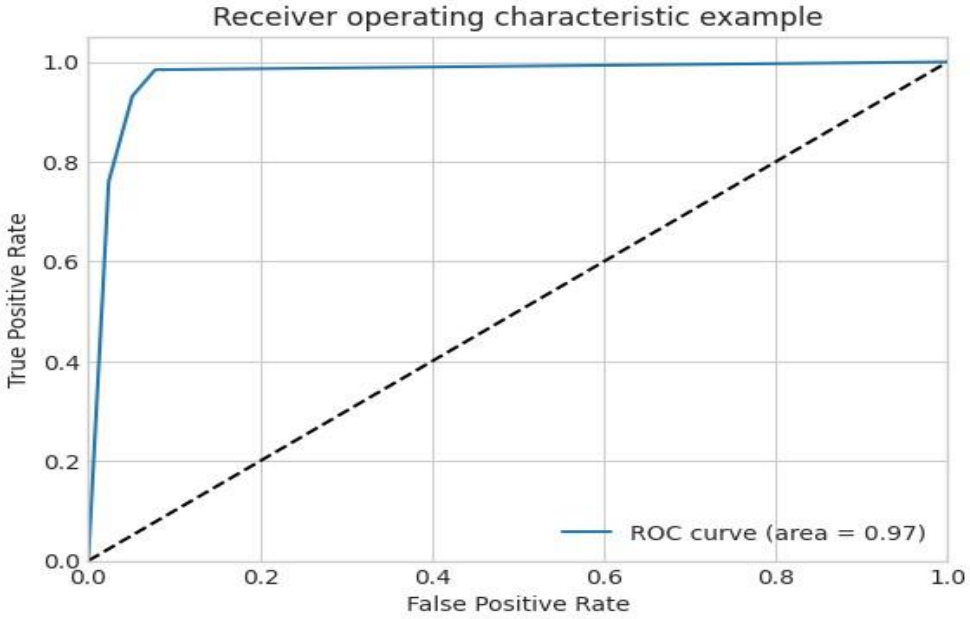
```
# # classifier
clf = OneVsRestClassifier(KNeighborsClassifier(n_neighbors = 3))
y_score = clf.fit(X_train, y_train).predict_proba(X_test)

predictions = clf.predict(X_test)
predictionstrain = clf.predict(X_train)
print("ACCURACY OF THE KNN MODEL:", metrics.accuracy_score(y_test, predictions))
print('\n\nTraining data metrics\n',metrics.classification_report(y_train, predictionstrain))
print('Testing data metrics\n',metrics.classification_report(y_test, predictions))


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

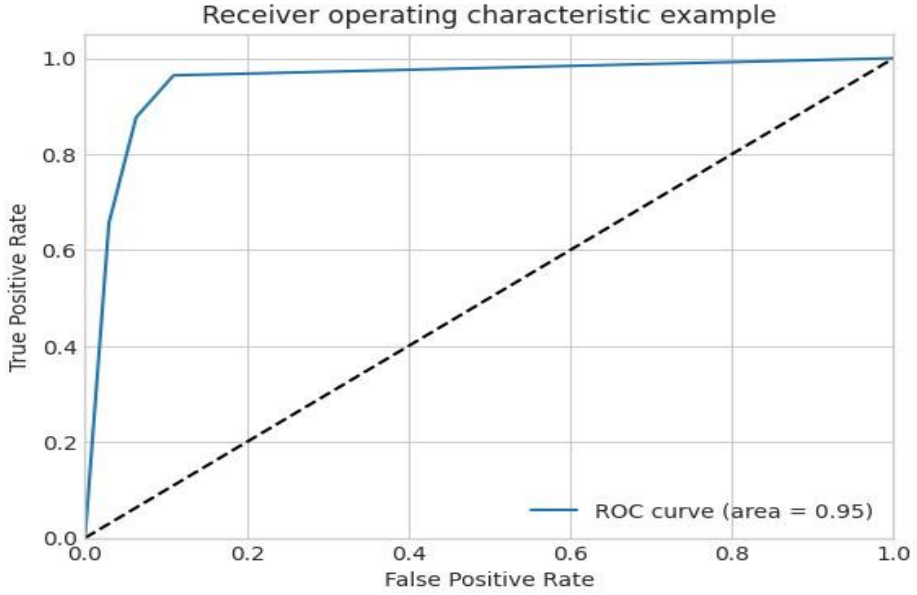
```
ACCURACY OF THE KNN MODEL: 0.7185570115981322
```

```
ACCURACY OF THE KNN MODEL: 0.7185570115981322


Training data metrics
              precision    recall  f1-score   support

           0       0.83      0.67      0.74     20255
           1       0.91      0.97      0.94     20197
           2       0.79      0.79      0.79     20179
           3       0.88      0.94      0.91     20243

   micro avg       0.85      0.84      0.85     80874
   macro avg       0.85      0.84      0.84     80874
weighted avg       0.85      0.84      0.84     80874
 samples avg       0.84      0.84      0.84     80874

Testing data metrics
              precision    recall  f1-score   support

           0       0.60      0.47      0.52      9922
           1       0.86      0.93      0.89      9980
           2       0.60      0.60      0.60      9998
           3       0.82      0.88      0.85      9934

   micro avg       0.73      0.72      0.72     39834
   macro avg       0.72      0.72      0.72     39834
weighted avg       0.72      0.72      0.72     39834
 samples avg       0.72      0.72      0.72     39834
```









autumn' is '0’ - top left, spring is ‘1’ - bottom left, summer is ‘2’ - top right, winter is ‘3’ -bottom right

GAUSSIAN NAÏVE BAYES ON OVERSAMPLED DATA

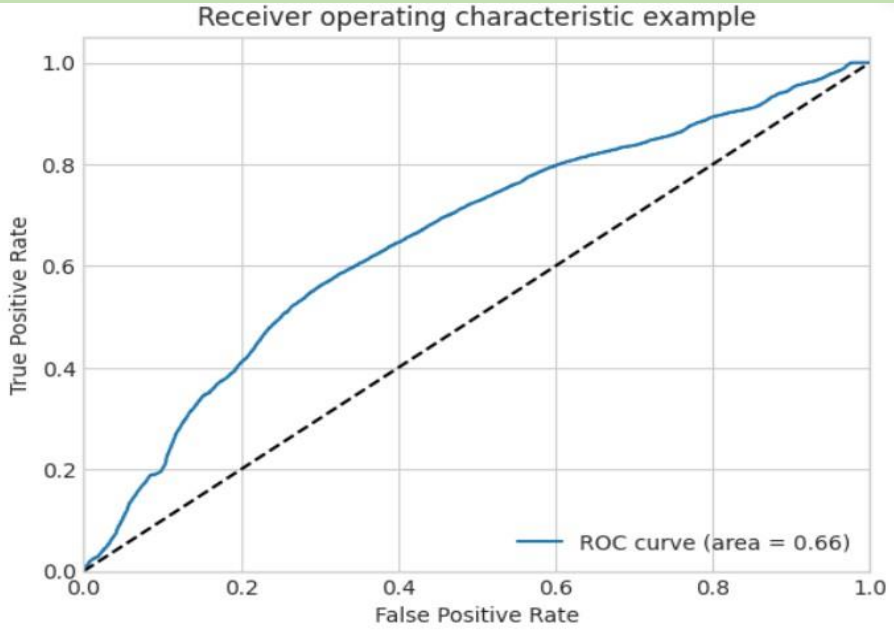
GAUSSIAN NAIVE BAYES ON OVERSAMPLED DATA

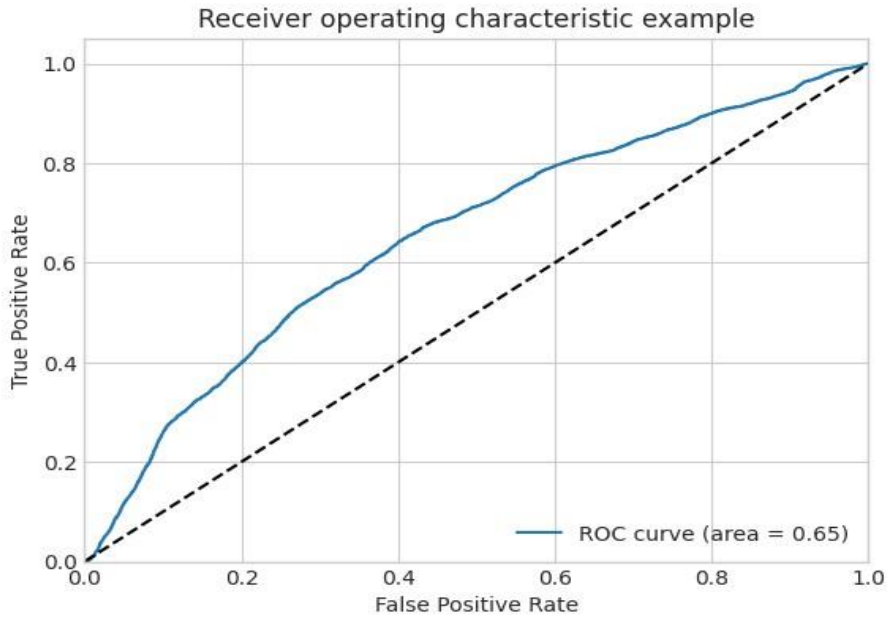
In [768…

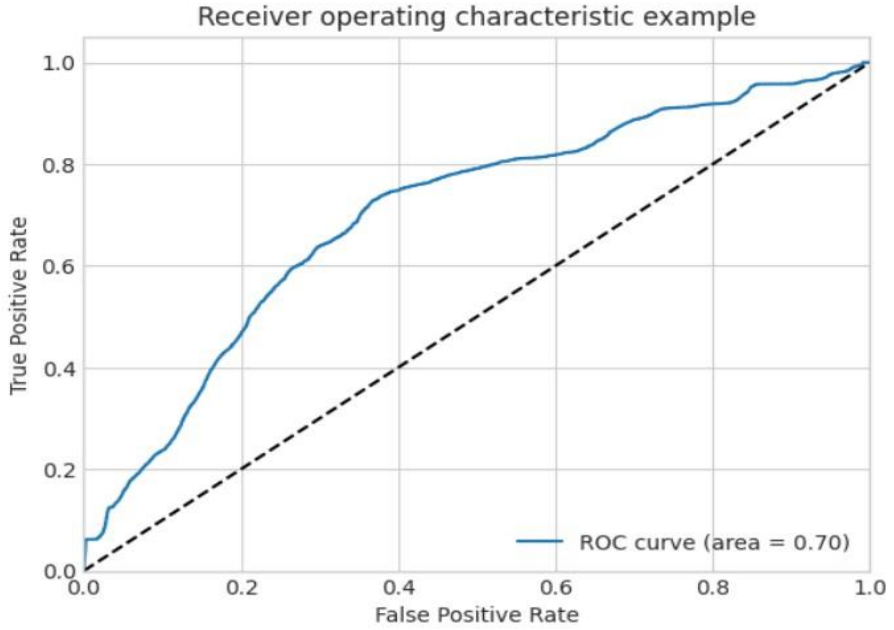
```
# # classifier
clf = OneVsRestClassifier(GaussianNB())
y_score = clf.fit(X_train, y_train).predict_proba(X_test)

predictions = clf.predict(X_test)
predictionstrain = clf.predict(X_train)
print("ACCURACY OF THE THE Naive Bayes MODEL:", metrics.accuracy_score(y_test, predictions))
print('\n\nTraining data metrics\n',metrics.classification_report(y_train, predictionstrain))
print('Testing data metrics\n',metrics.classification_report(y_test, predictions))


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

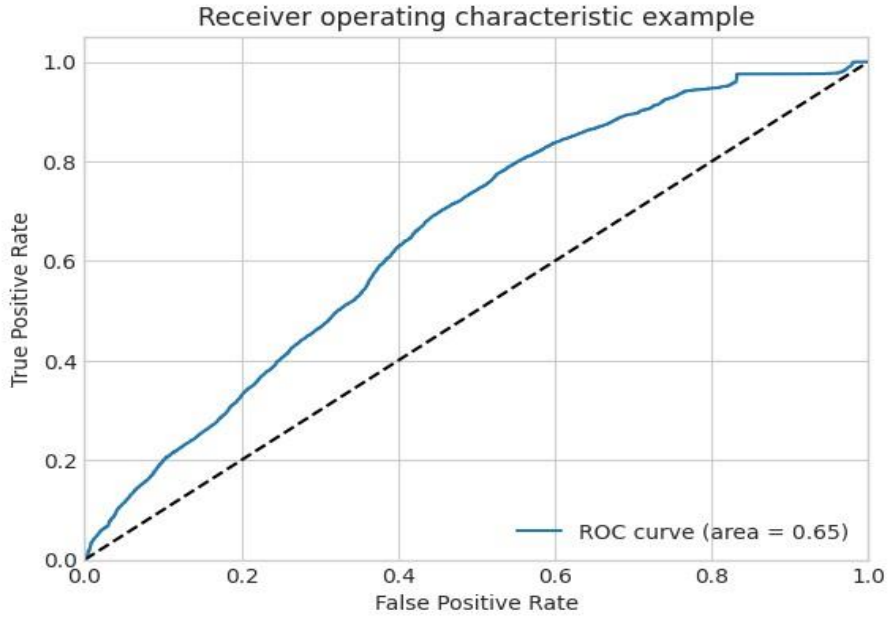
```
ACCURACY OF THE THE Naive Bayes MODEL: 0.2078626299141437
```

```
ACCURACY OF THE THE Naive Bayes MODEL: 0.2078626299141437


Training data metrics
              precision    recall  f1-score   support

           0       0.43      0.25      0.31     20255
           1       0.47      0.20      0.28     20197
           2       0.45      0.21      0.29     20179
           3       0.34      0.63      0.44     20243

   micro avg       0.39      0.32      0.35     80874
   macro avg       0.42      0.32      0.33     80874
weighted avg       0.42      0.32      0.33     80874
 samples avg       0.26      0.32      0.28     80874

Testing data metrics
              precision    recall  f1-score   support

           0       0.42      0.25      0.31      9922
           1       0.47      0.20      0.28      9980
           2       0.45      0.21      0.29      9998
           3       0.34      0.62      0.44      9934

   micro avg       0.39      0.32      0.35     39834
   macro avg       0.42      0.32      0.33     39834
weighted avg       0.42      0.32      0.33     39834
 samples avg       0.26      0.32      0.28     39834
```









autumn' is '0’ - top left, spring is ‘1’ - bottom left, summer is ‘2’ - top right, winter is ‘3’ -bottom right

DECISION TREE CLASSIFIER ON OVERSAMPLED DATA

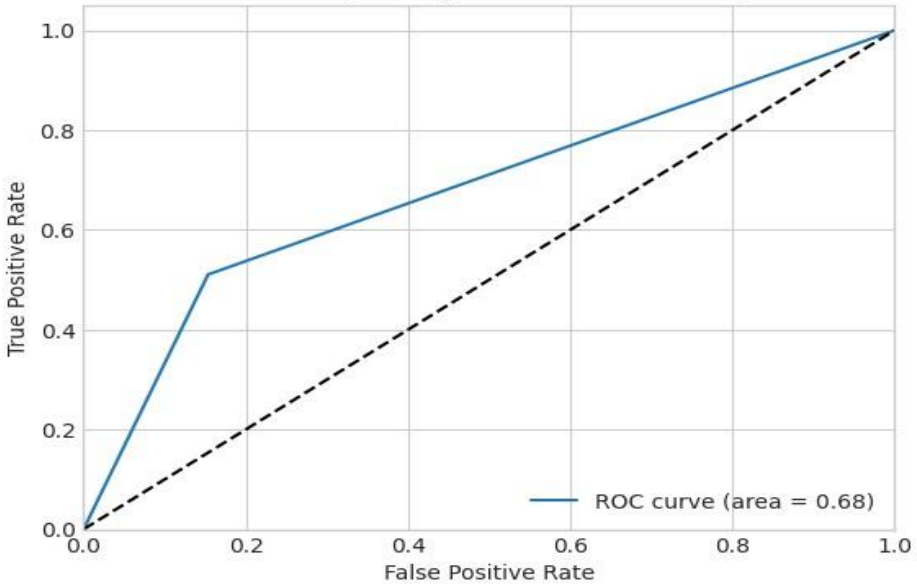
Decision Tree Classifier on oversampled data

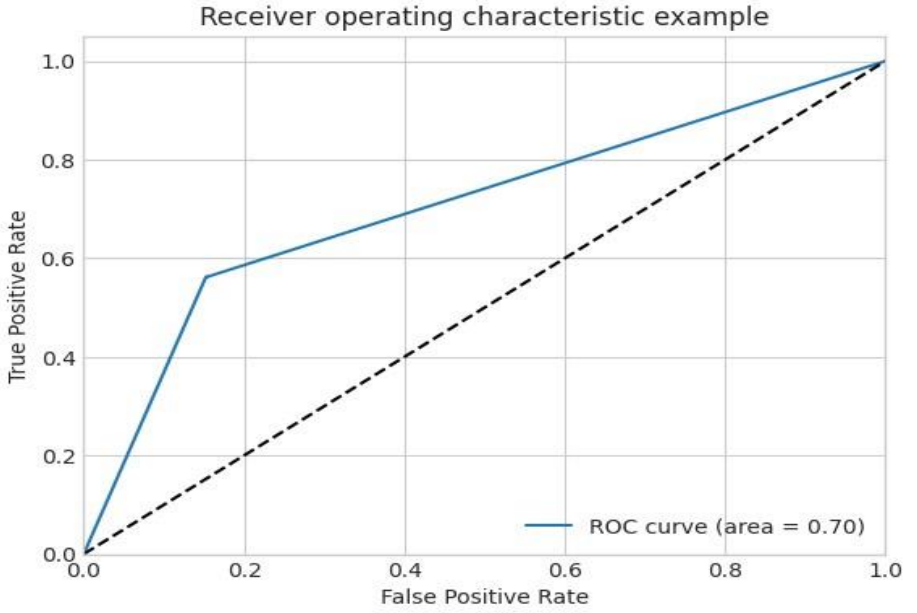
In [769…

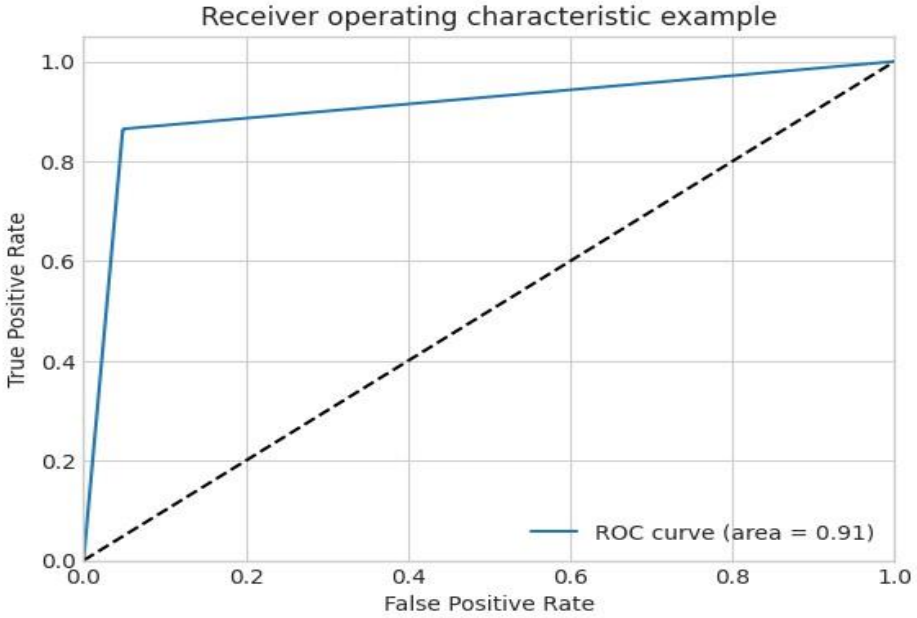
```
# # classifier
clf = OneVsRestClassifier(DecisionTreeClassifier(random_state=0))
y_score = clf.fit(X_train, y_train).predict_proba(X_test)


predictions = clf.predict(X_test)
predictionstrain = clf.predict(X_train)
print("ACCURACY OF THE Decision Tree Classifier MODEL:", metrics.accuracy_score(y_test, predictions))
print('\n\nTraining data metrics\n',metrics.classification_report(y_train, predictionstrain))
print('Testing data metrics\n',metrics.classification_report(y_test, predictions))


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

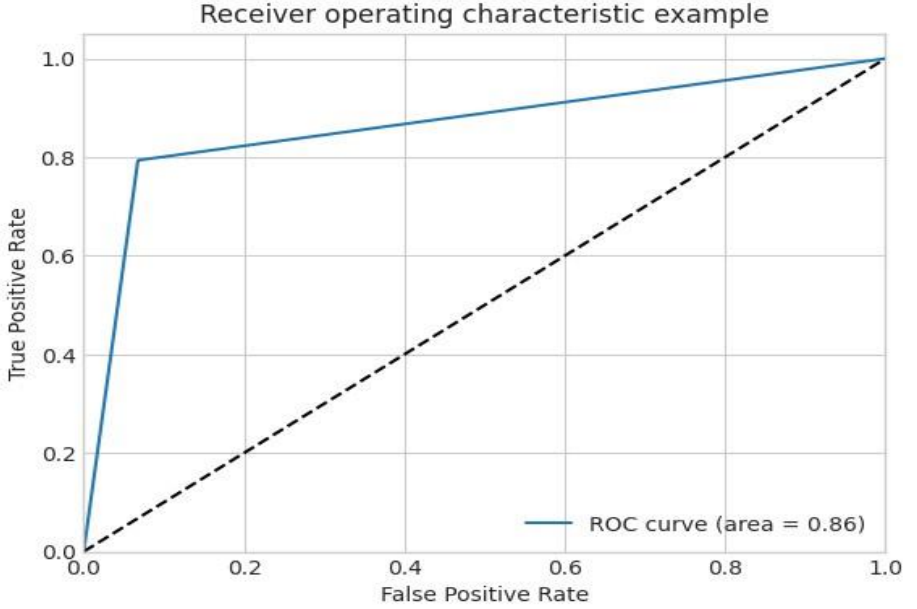
```
ACCURACY OF THE Decision Tree Classifier MODEL: 0.5757393181704072
```

```
ACCURACY OF THE Decision Tree Classifier MODEL: 0.5757393181704072


Training data metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     20255
           1       1.00      1.00      1.00     20197
           2       1.00      1.00      1.00     20179
           3       1.00      1.00      1.00     20243

   micro avg       1.00      1.00      1.00     80874
   macro avg       1.00      1.00      1.00     80874
weighted avg       1.00      1.00      1.00     80874
 samples avg       1.00      1.00      1.00     80874

Testing data metrics
              precision    recall  f1-score   support

           0       0.52      0.51      0.52      9922
           1       0.86      0.86      0.86      9980
           2       0.55      0.56      0.56      9998
           3       0.80      0.79      0.79      9934

   micro avg       0.68      0.68      0.68     39834
   macro avg       0.68      0.68      0.68     39834
weighted avg       0.68      0.68      0.68     39834
 samples avg       0.63      0.68      0.65     39834
```









autumn' is '0’ - top left, spring is ‘1’ - bottom left, summer is ‘2’ - top right, winter is ‘3’ -bottom right

GRADIENT BOOST ON OVERSAMPLED DATA

Gradient boost on oversampled data

In [770…

```
# # classifier
clf = OneVsRestClassifier(GradientBoostingClassifier(random_state=0))
y_score = clf.fit(X_train, y_train).predict_proba(X_test)


predictions = clf.predict(X_test)
predictionstrain = clf.predict(X_train)
print("ACCURACY OF THE Gradient Boosting Classifier MODEL:", metrics.accuracy_score(y_test, predictions))
print('\n\nTraining data metrics\n',metrics.classification_report(y_train, predictionstrain))
print('Testing data metrics\n',metrics.classification_report(y_test, predictions))


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

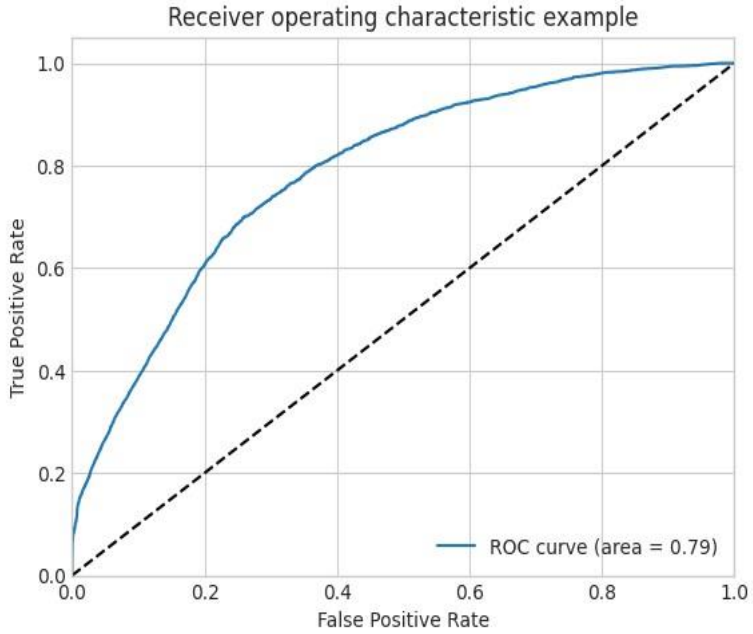
```
ACCURACY OF THE Gradient Boosting Classifier MODEL: 0.42072099211728675
```

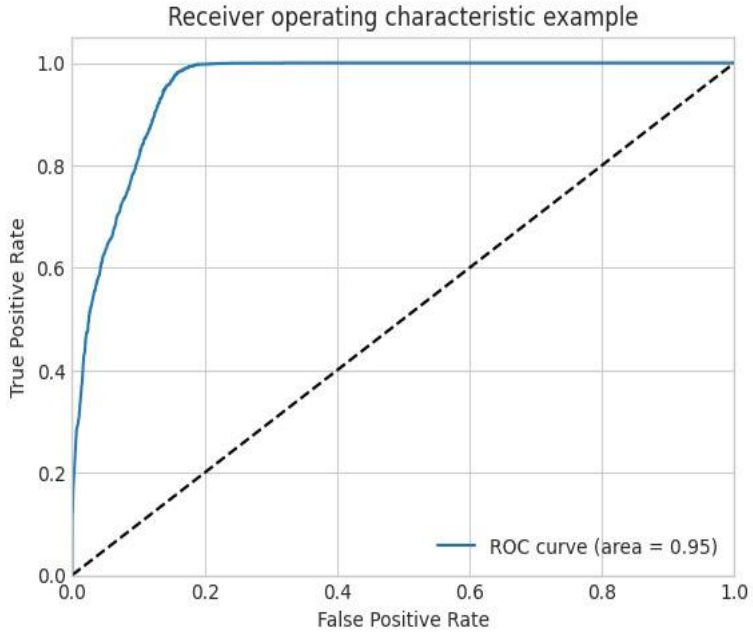
Training data metrics

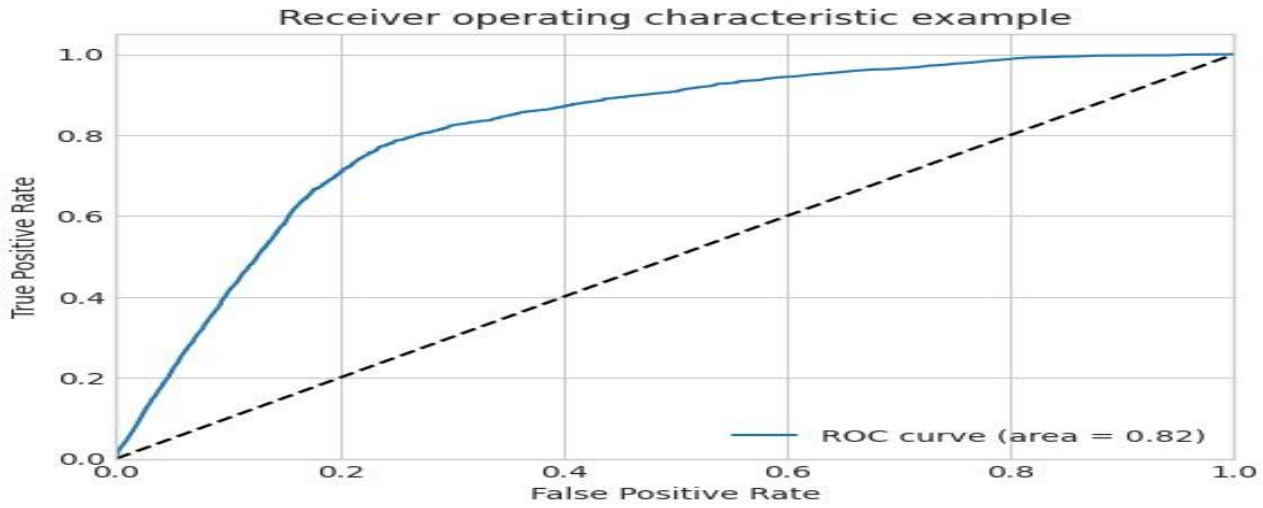
	precision	recall	f1-score	support
0	0.71	0.23	0.35	20255
1	0.77	0.73	0.75	20197
2	0.60	0.23	0.33	20179
3	0.81	0.57	0.67	20243
micro avg	0.74	0.44	0.55	80874
macro avg	0.72	0.44	0.53	80874
weighted avg	0.72	0.44	0.53	80874
samples avg	0.43	0.44	0.44	80874

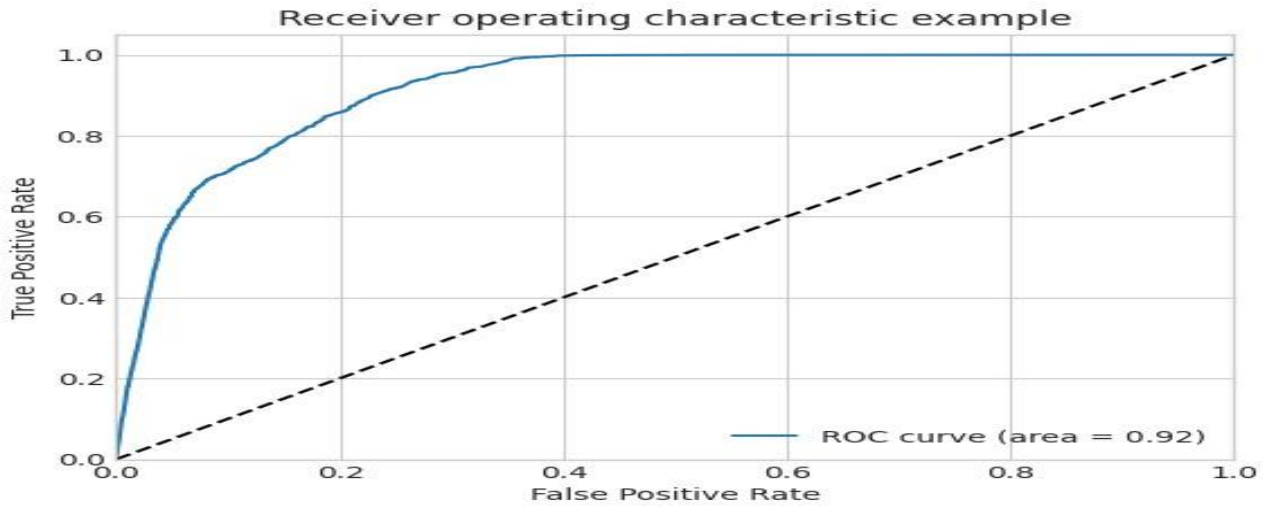
Testing data metrics

	precision	recall	f1-score	support
0	0.68	0.23	0.34	9922
1	0.76	0.73	0.75	9980
2	0.59	0.22	0.33	9998
3	0.81	0.56	0.66	9934
micro avg	0.74	0.44	0.55	39834
macro avg	0.71	0.44	0.52	39834
weighted avg	0.71	0.44	0.52	39834
samples avg	0.43	0.44	0.43	39834










```
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  11.9s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.2s
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.3s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=  12.3s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=  12.4s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.7s
[CV] END bootstrap=True, max_depth=22, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=  12.6s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  12.9s
[CV] END bootstrap=True, max_depth=46, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=  13.5s
```

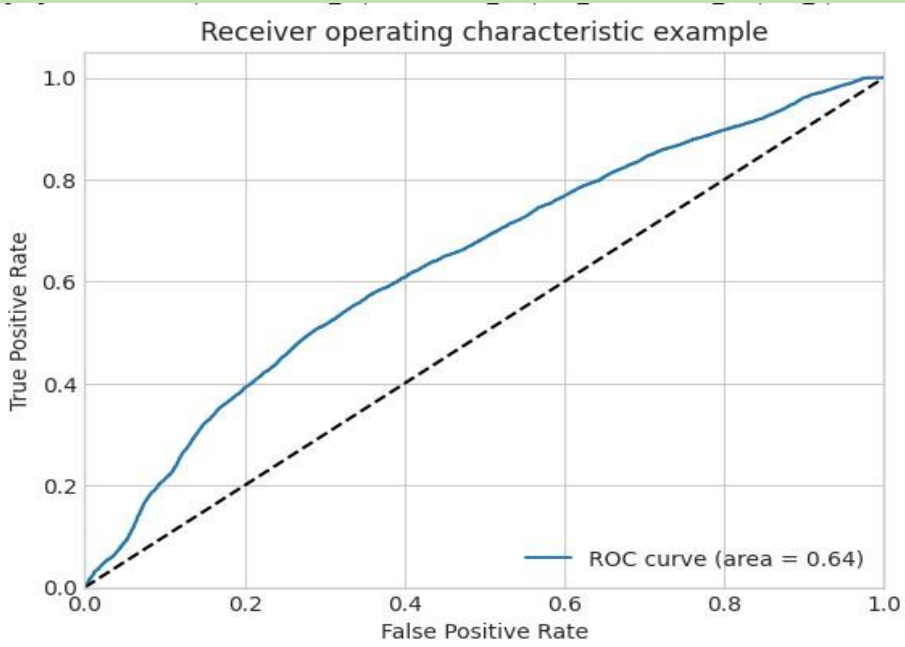
autumn' is '0’ - top left, spring is ‘1’ - bottom left, summer is ‘2’ - top right, winter is ‘3’ -bottom right

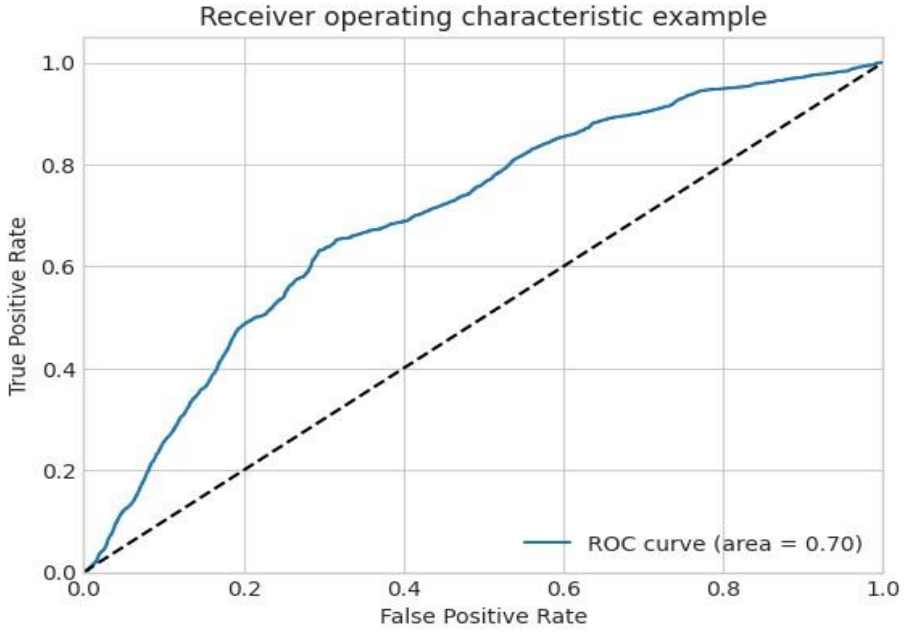
LINEAR SVC ON OVERSAMPLED DATA

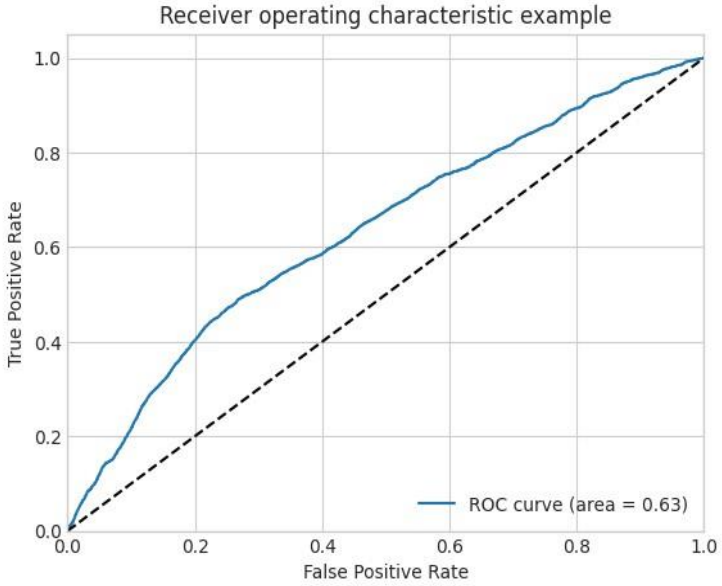
LINEAR SVC (SVM)

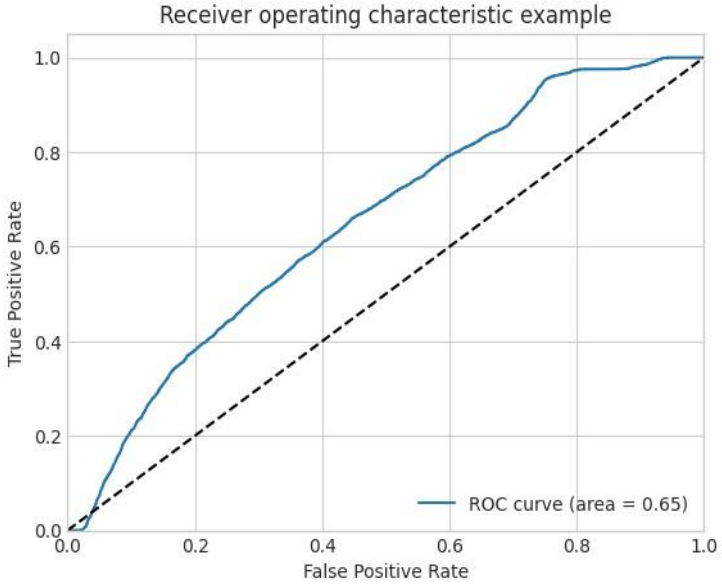
In [450]:
```
# # classifier
clf = OneVsRestClassifier(LinearSVC(random_state=0))
y_score = clf.fit(X_train, y_train).decision_function(X_test)

preds = clf.predict(X_test)


print("Accuracy of the Linear SVC with SMOTE data is: ", accuracy_score(y_test,preds)*100)
cr = classification_report(y_test, preds)
print(cr)


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

```
Accuracy of the Linear SVC with SMOTE data is:  9.730218235372874
              precision    recall  f1-score   support

           0       0.51      0.07      0.12      9914
           1       0.54      0.23      0.32     10074
           2       0.43      0.05      0.09      9888
           3       0.00      0.00      0.00         0

   micro avg       0.52      0.12      0.19     29876
   macro avg       0.37      0.09      0.13     29876
weighted avg       0.50      0.12      0.18     29876
 samples avg       0.11      0.12      0.11     29876
```










```
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=  51.0s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=  51.5s
[CV] END bootstrap=False, max_depth=34, min_samples_leaf=1, min_samples_split=5, n_estimators=555; total time=  51.9s
```

autumn' is '0’ - top left, spring is ‘1’ - bottom left, summer is ‘2’ - top right, winter is ‘3’ -bottom right

ML ALGORITHMS	ACCURACY in %age		AUC		
	Raw Data	Oversampled Data	Raw Data	Oversampled Data	
LOGISTIC REGRESSION MODEL	49.65 %	37 %	0.47	0.65	0.66
				0.71	0.65
K – NEAREST NEIGHBORS MODEL	51.18 %	71 %	0.48	0.77	0.83
				0.97	0.95
GAUSSIAN NAÏVE BAYES MODEL	46.89 %	20 %	0.47	0.66	0.65
				0.70	0.65
LINEAR SVC MODEL	49.52 %	9.73 %	0.47	0.64	0.63
				0.70	0.65
STOCHASTIC GRADIENT DESCENT MODEL	49.74 %	---	---	---	---
				---	---
DECISION TREE CLASSIFIER MODEL	51.47 %	57 %	0.49	0.68	0.70
				0.91	0.86
RANDOM FOREST CLASSIFIER MODEL	51.81 %	82 %	0.48	0.84	0.87
				0.98	0.97
GRADIENT BOOSTING TREES MODEL	51.6 %	42 %	0.48	0.79	0.82
				0.95	0.92

PERFORMANCE ANALYSIS OF THE MODELS

- For the Raw Data (not over/under sampled) almost all algorithms performed fairly similar, however the maximum accuracy is achieved by Random Forest Classifier and the lowest with Naïve Bayes.
- The AUC score for the Raw Data follows the same trend and varies between 0.47 to 0.49.
- After the Oversampling using the SMOTE, we saw a significant increase in accuracy and AUC score for Random Forest Classifier and K nearest algorithm with accuracy 82% and 71% respectively.
- For the other algorithms, the accuracy got reduced which accounts to the fact that oversampling reduces the overall accuracy and increases the chance of predicting the right class in the minority classes.
- Surprisingly, the AUC score for all the algorithms for each classes is significantly high.
- From "Learning from Imbalanced Datasets" (Page 55, 2018) we came to conclusion that for Multiclass Classification problem accuracy is not always the best measure.

CHALLENGES

- We found anomalies in dataset, some entries with label 'd’ for a categorical variable has no meaning provided in the data dictionary.
- We found that our data for the target variable, season, is highly imbalanced.
- ROC Curve and AUC are high even if the accuracy is low.
- Even after binarizing the target classes and plotting the ROC curve with One Vs Rest classifier we are observing a significant difference in AUC and accuracy.

LEARNINGS

- If the data is imbalanced, then the accuracy and other metrics will be biased towards the majority classes.
- To tackle the imbalanced data in the target variable, we can do oversampling, under sampling.
- If there is a mix of categorical variables and continuous variables in predictors we should normalize or standardize the continuous variables so that they are scaled when categorical variables are encoded.

BIBLIOGRAPHY

- Wagner, D., Heider, D. & Hattab, G. Mushroom data creation, curation, and simulation to support classification tasks. Sci Rep 11, 8134 (2021). https://doi.org/10.1038/s41598-021-87602-3
- https://archive.ics.uci.edu/ml/datasets/Secondary+Mushroom+Dataset
- https://www.hsph.harvard.edu/nutritionsource/food-features/mushrooms/
- https://www.grandviewresearch.com/industry-analysis/mushroom-market
- https://www.data-to-viz.com/
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/

- https://link.springer.com/book/10.1007/978-3-319-98074-4
- https://www.tutorialspoint.com/machine_learning_with_python/classification_introduction.htm
- https://www.activestate.com/resources/quick-reads/how-to-classify-data-in-python/#:~:text=Classification%20in%20supervised%20Machine%20Learning,has%20not%20yet%20been%20labeled
- https://www.digitalocean.com/community/tutorials/exploratory-data-analysis-python
- https://towardsdatascience.com/exploratory-data-analysis-in-python-a-step-by-step-process-d0dfa6bf94ee

THANK YOU

APPENDIX

IMPORTING LIBRARIES

In [83]:
```
#Basic Python standard
import io, os, sys, types, time, datetime, math, random, requests, subprocess, tempfile
from io import StringIO
```

In [84]:
```
# Data Manipulation
import numpy as np
import pandas as pd
import statsmodels.api as sm
```

In [85]:
```
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.mosaicplot import mosaic
import missingno as msno
```

In [86]:
```
# Feature Selection and Encoding
from sklearn.feature_selection import RFE, RFECV
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
```

In [87]:
```
# Machine learning
import sklearn.ensemble as ske
from sklearn import datasets, model_selection, tree, preprocessing, metrics, linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge, Lasso, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

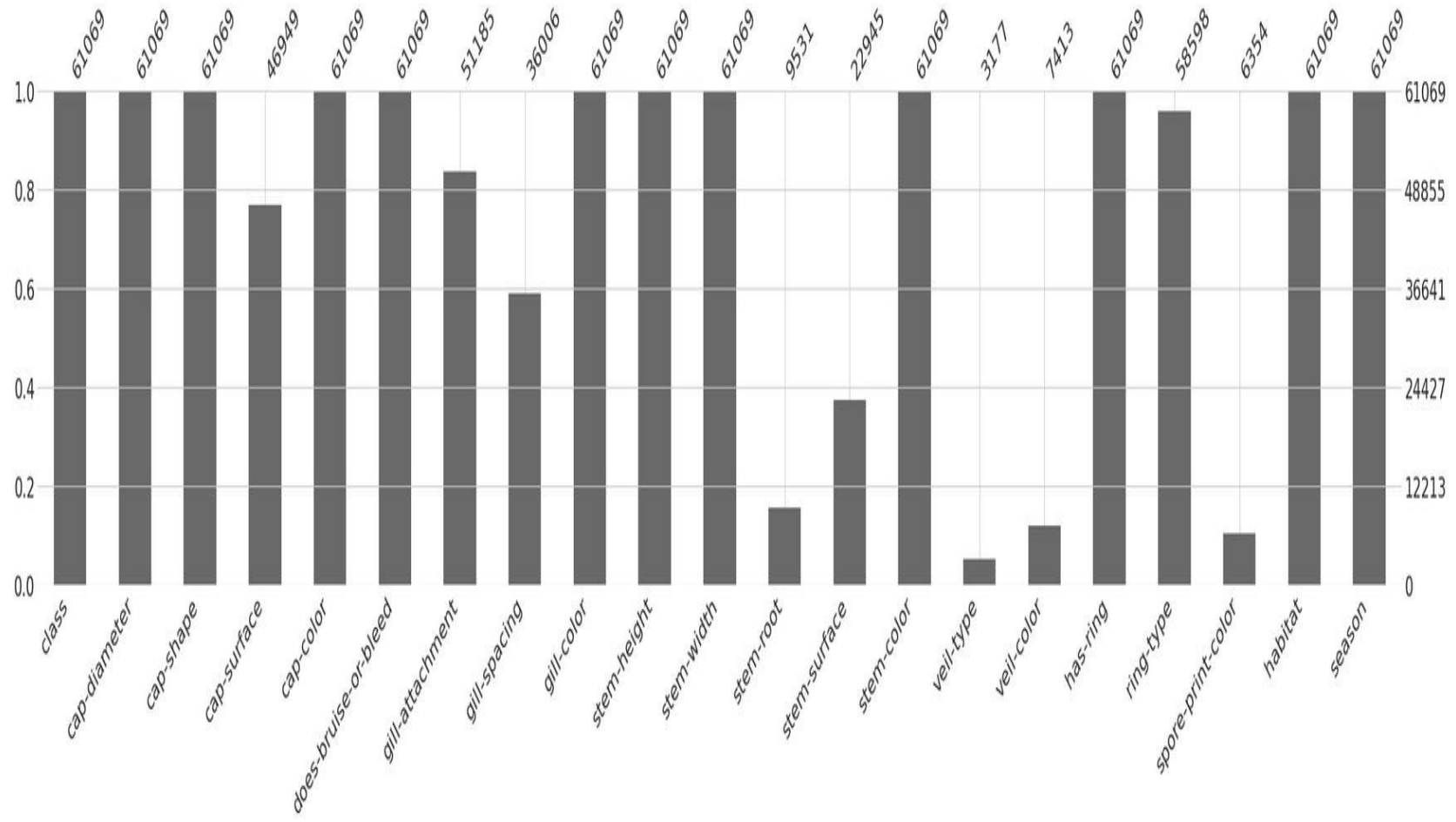
In [88]:
```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import classification_report
```

In [89]:
```
import dython
from dython.nominal import associations
from dython.nominal import identify_nominal_columns
```

Association between all the variables

```
In [87]: msno.bar(df, figsize = (30,5))
```

Out[87]: <AxesSubplot:>




```
df_categorical_variables = df[['class',
 'cap-shape',
 'cap-surface',
 'cap-color',
 'does-bruise-or-bleed',
 'gill-attachment',
 'gill-spacing',
 'gill-color',
 'stem-color',
 'has-ring',
 'ring-type',
 'habitat',
 'season']].copy()

#df_categorical_variables
```

```
associations(df_categorical_variables, nominal_columns = 'auto', numerical_columns = None, mark_columns = False ,nom_nom_assoc = 'cramer', num_num_assoc = 'pearson', figsize = (20
```

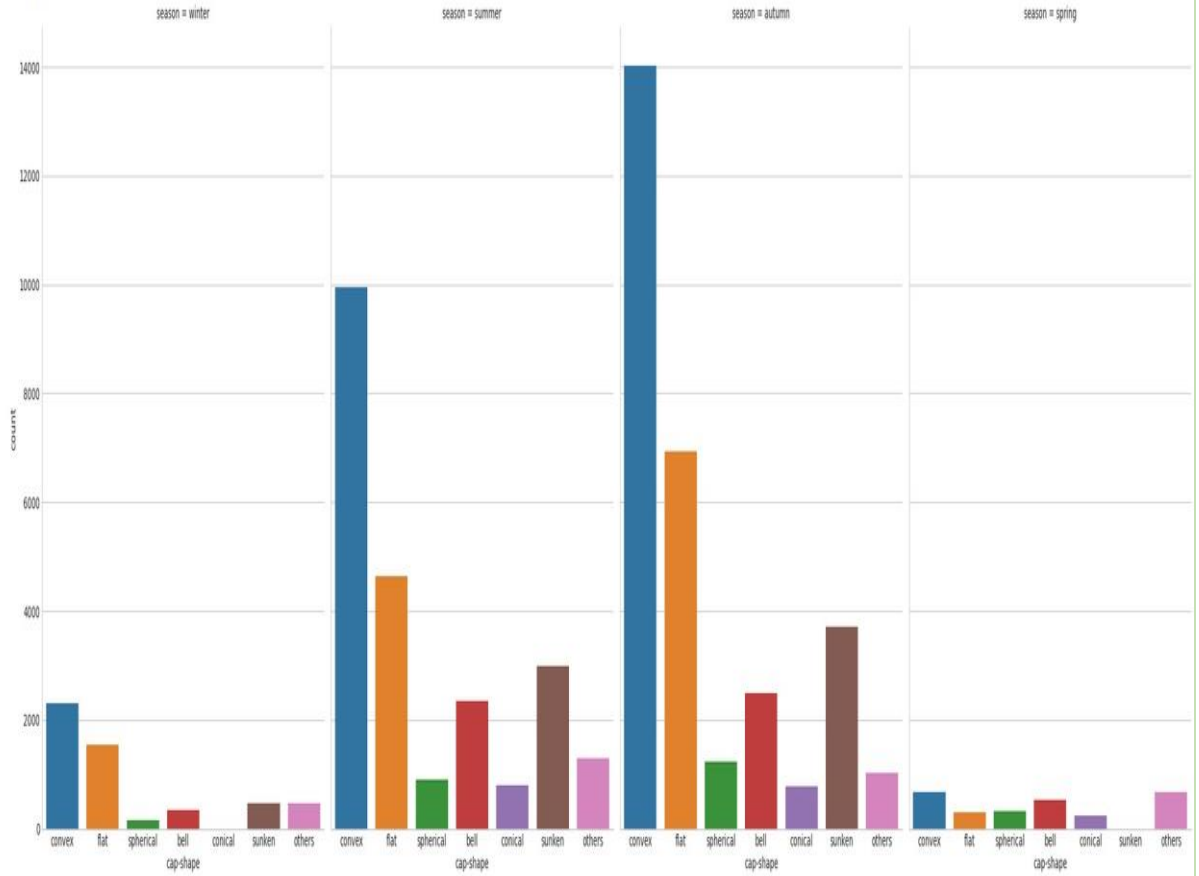
In [490…
```
pval = Asses_variable('season', 'cap-shape')
pvalues_list.append(pval)
```

Table Original

cap-shape	bell	conical	convex	flat	others	spherical	sunken
season							
autumn	2483	778	14019	6931	1025	1227	3714
spring	520	241	666	297	674	329	0
summer	2346	796	9949	4638	1295	891	2983
winter	345	0	2300	1538	466	151	467

P-value is: 0.3060017609499464 7

<Figure size 300x500 with 0 Axes>



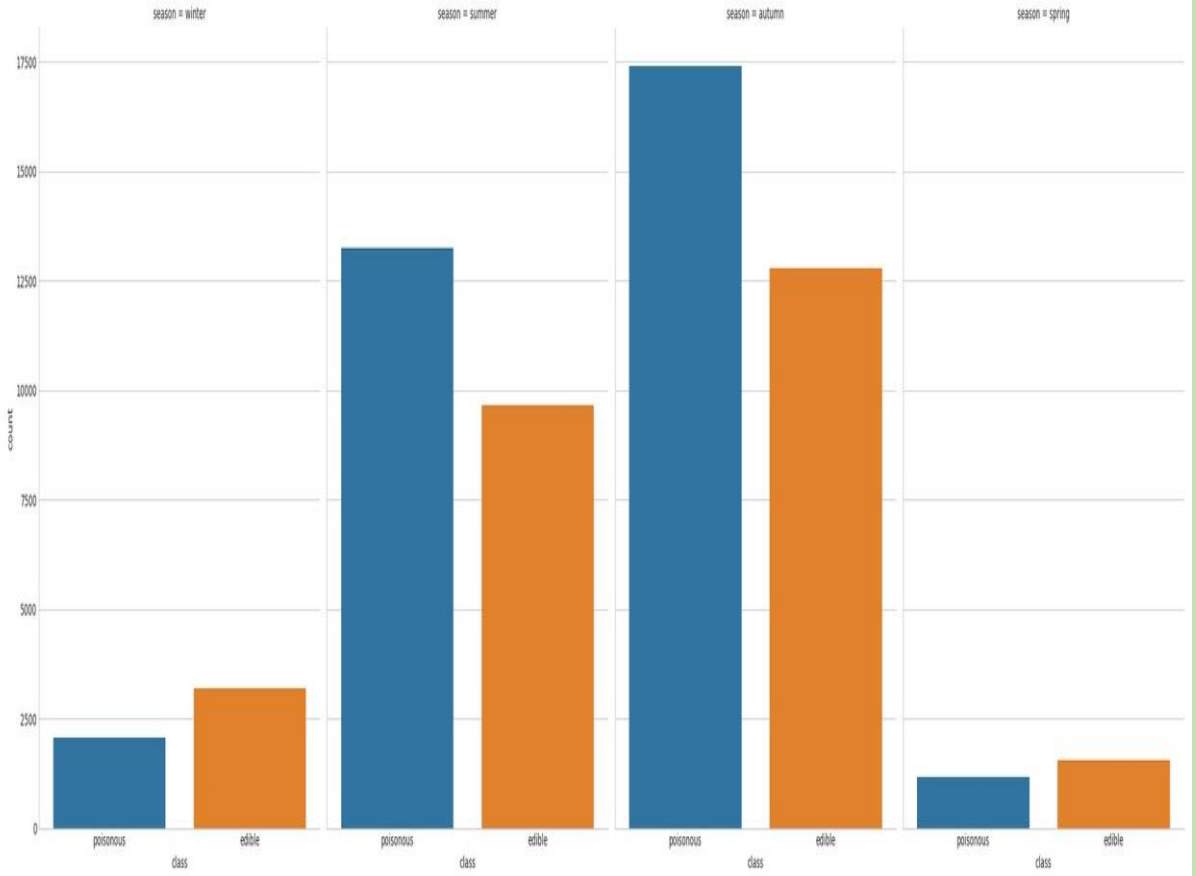
In [491…
```
pval = Asses_variable('season', 'class')
pvalues_list.append(pval)
```

Table Original

class	edible	poisonous
season		
autumn	12785	17392
spring	1553	1174
summer	9647	13251
winter	3196	2071

P-value is: 1.0953323541252613e-40

<Figure size 300x500 with 0 Axes>



In [492…
```
pval = Asses_variable('season', 'cap-surface')
pvalues_list.append(pval)
```

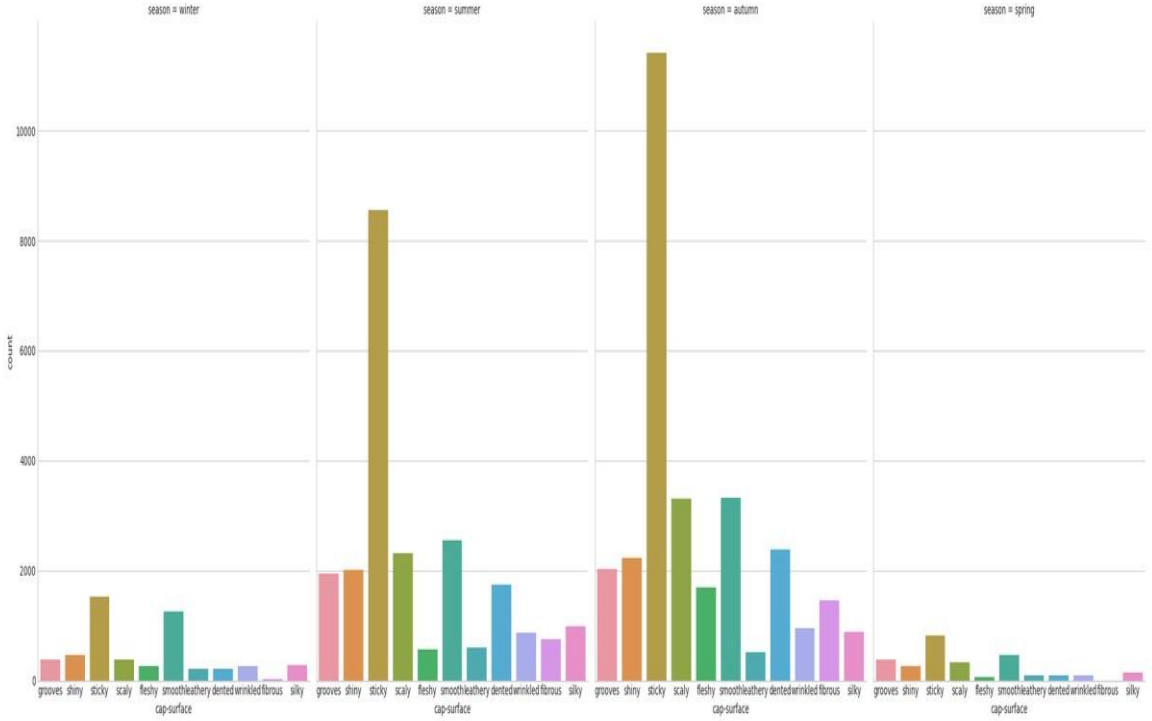
Table Original

cap-surface	dented	fibrous	fleshy	grooves	leathery	scaly	shiny	silky
season								
autumn	2379	1456	1690	2032	512	3304	2237	878
spring	95	0	58	374	86	334	258	150
summer	1743	746	571	1942	598	2321	2013	992
winter	215	23	265	376	216	382	466	283

cap-surface	smooth	sticky	wrinkled
season			
autumn	3331	11413	945
spring	470	815	87
summer	2553	8556	863
winter	1254	1532	255

P-value is: 9.120519468289969e-30

<Figure size 300x500 with 0 Axes>



In [493…
```
pval = Asses_variable('season', 'cap-color')
pvalues_list.append(pval)
```

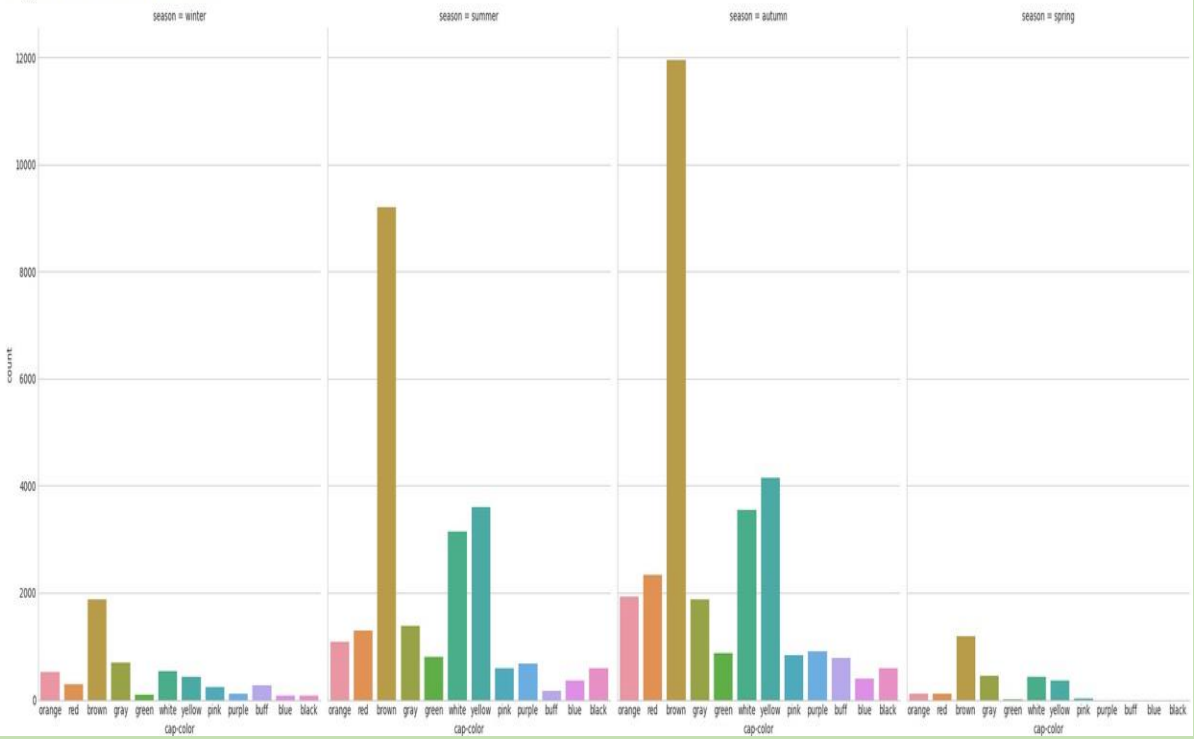
Table Original

cap-color	black	blue	brown	buff	gray	green	orange	pink	purple	red
season										
autumn	597	397	11944	785	1883	870	1924	837	908	2337
spring	0	0	1198	0	451	14	118	24	0	122
summer	595	356	9195	175	1383	799	1092	599	679	1288
winter	87	75	1881	270	703	99	522	243	122	288

cap-color	white	yellow
season		
autumn	3548	4147
spring	438	362
summer	3141	3596
winter	539	438

P-value is: 0.015122017659931063

<Figure size 300x500 with 0 Axes>



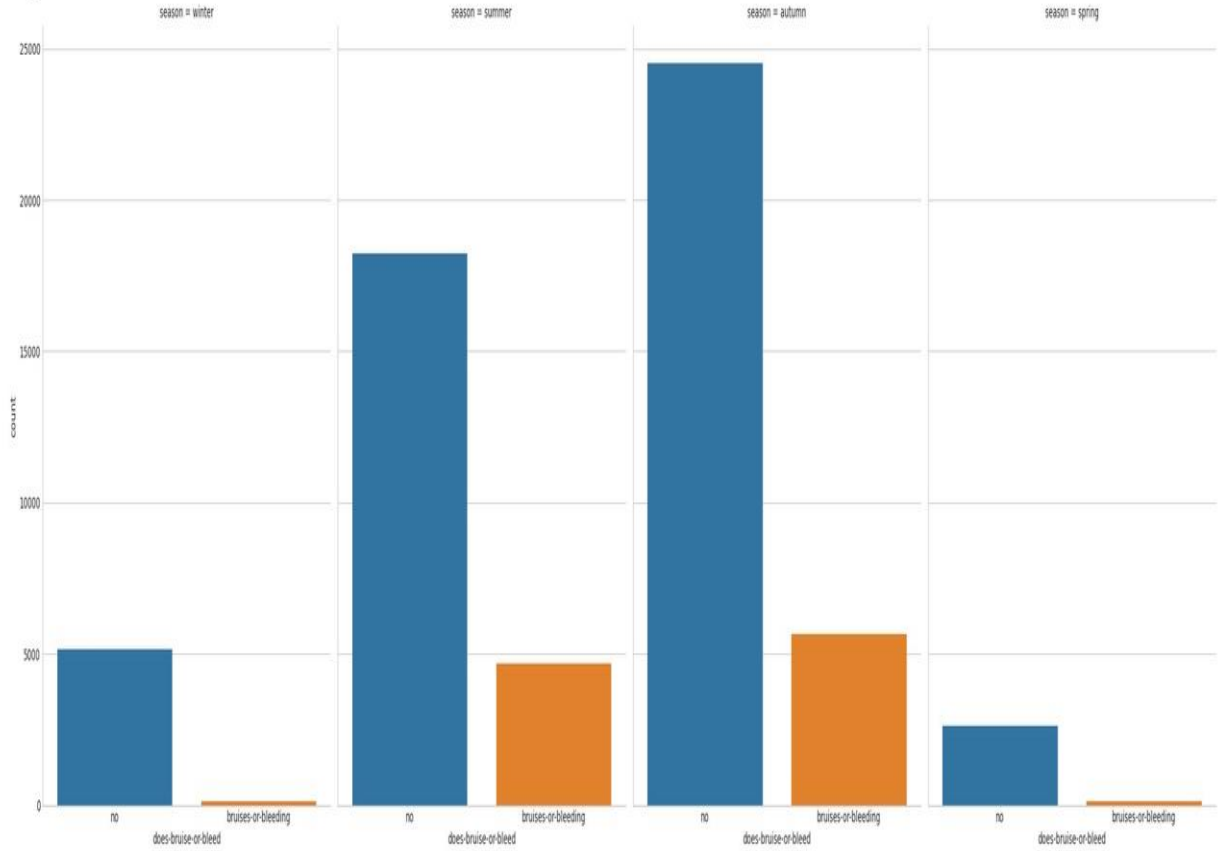
In [494…
```
pval = Asses_variable('season', 'does-bruise-or-bleed')
pvalues_list.append(pval)
```

Table Original

does-bruise-or-bleed	bruises-or-bleeding	no
season		
autumn	5668	24509
spring	116	2611
summer	4681	18217
winter	125	5142

P-value is: 1.285562652843610 4e-36

<Figure size 300x500 with 0 Axes>



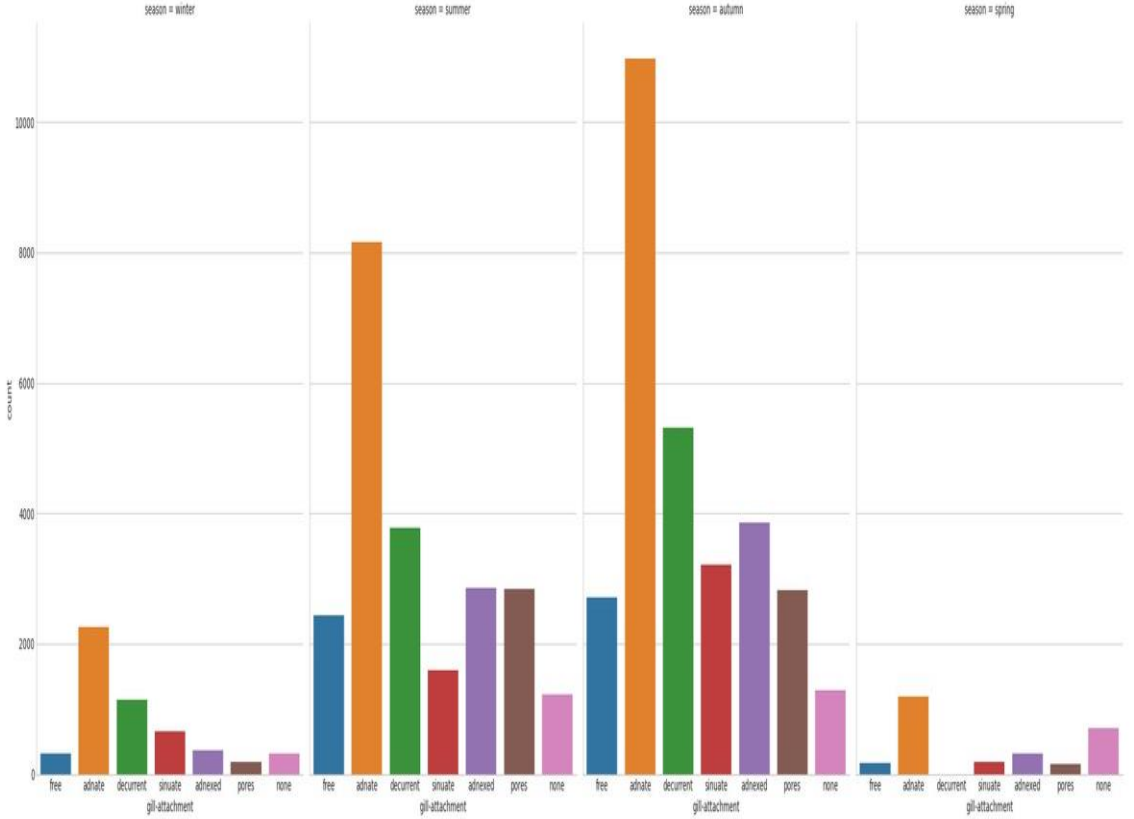
In [495…
```
pval = Asses_variable('season', 'gill-attachment')
pvalues_list.append(pval)
```

Table Original

gill-attachment	adnate	adnexed	decurrent	free	none	pores	sinuate
season							
autumn	10972	3864	5315	2706	1293	2819	3208
spring	1184	317	0	179	709	156	182
summer	8165	2860	3783	2442	1218	2836	1594
winter	2261	372	1149	321	310	190	664

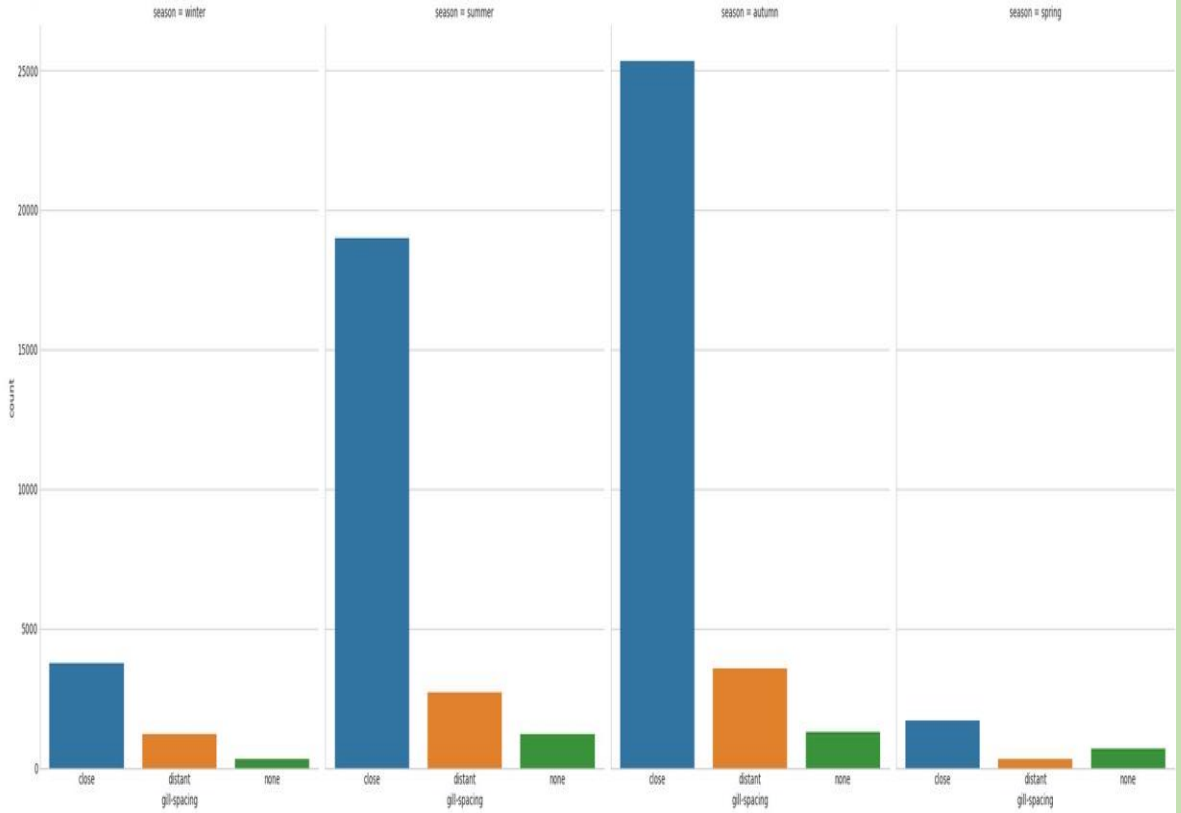
P-value is: 0.0026362803767766 93

<Figure size 300x500 with 0 Axes>



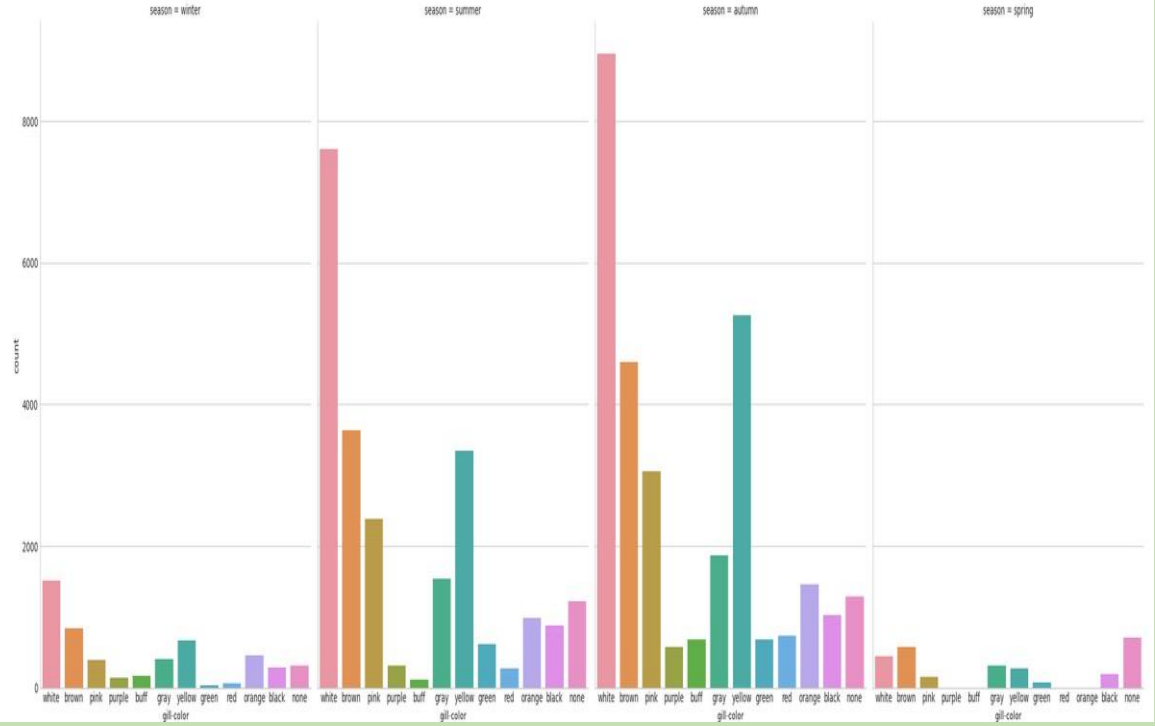
In [496…
```
pval = Asses_variable('season', 'gill-spacing')
pvalues_list.append(pval)
```

```
Table Original
gill-spacing  close  distant  none
season
autumn        25327     3557  1293
spring         1715      303   709
summer        18984     2696  1218
winter         3747     1210   310
P-value is:  8.537123820515668e-36
<Figure size 300x500 with 0 Axes>
```



In [497…
```
pval = Asses_variable('season', 'gill-color')
pvalues_list.append(pval)
```

```
Table Original
gill-color  black  brown  buff  gray  green  none  orange  pink  purple  red  \
season
autumn       1021   4594   677  1870    681  1293    1457  3059     577  736
spring        190    572     0   305     79   709       0   153       0    0
summer        874   3637   114  1542    609  1218     990  2379     310  274
winter        290    842   163   401     30   310     462   392     136   56

gill-color  white  yellow
season
autumn       8950    5262
spring        446     273
summer       7607    3344
winter       1518     667
P-value is:  5.1726529176152304e-11
<Figure size 300x500 with 0 Axes>
```



In [498…
```
pval = Asses_variable('season', 'stem-color')
pvalues_list.append(pval)
```

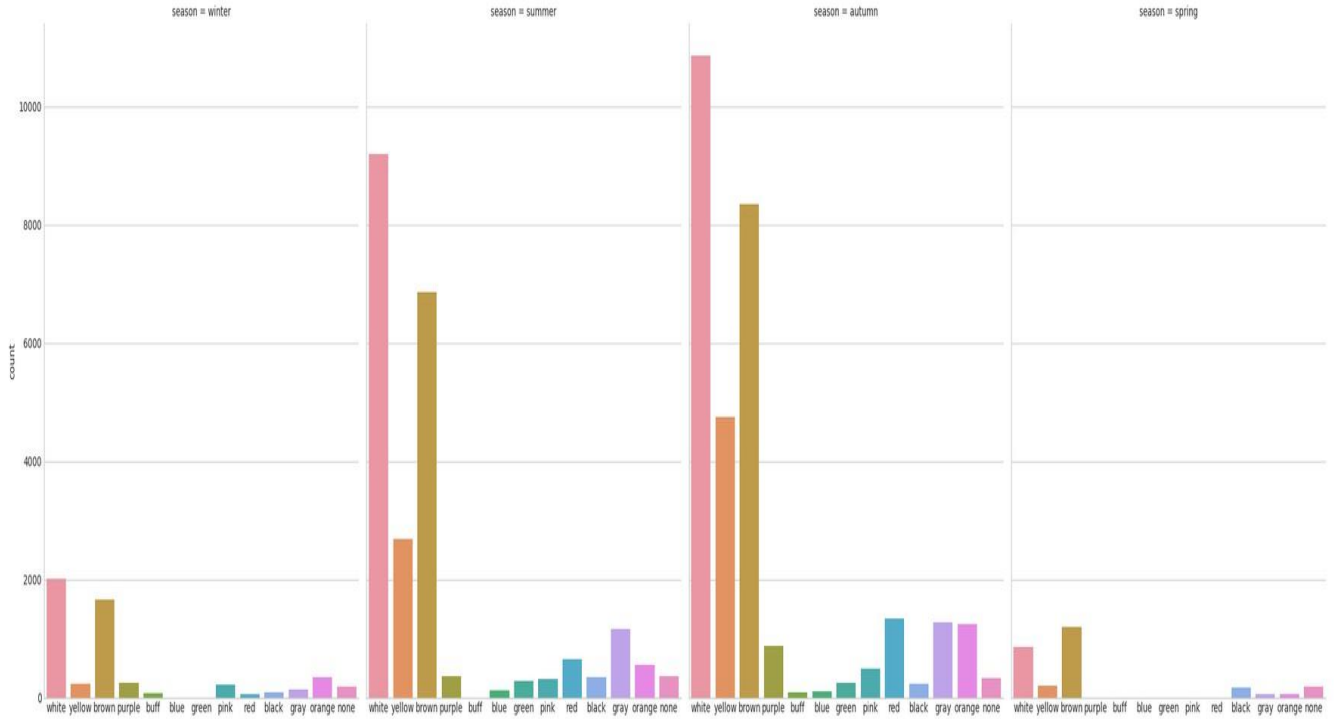
Table Original

season	black	blue	brown	buff	gray	green	none	orange	pink	purple
autumn	235	109	8344	96	1266	257	323	1234	486	876
spring	167	0	1201	0	51	0	183	62	0	0
summer	351	117	6863	0	1169	285	366	549	314	363
winter	84	0	1655	77	140	0	187	342	225	251

season	red	white	yellow
autumn	1338	10859	4754
spring	0	864	199
summer	650	9196	2675
winter	62	2007	237

P-value is: 5.242481280064299e-35

<Figure size 300x500 with 0 Axes>



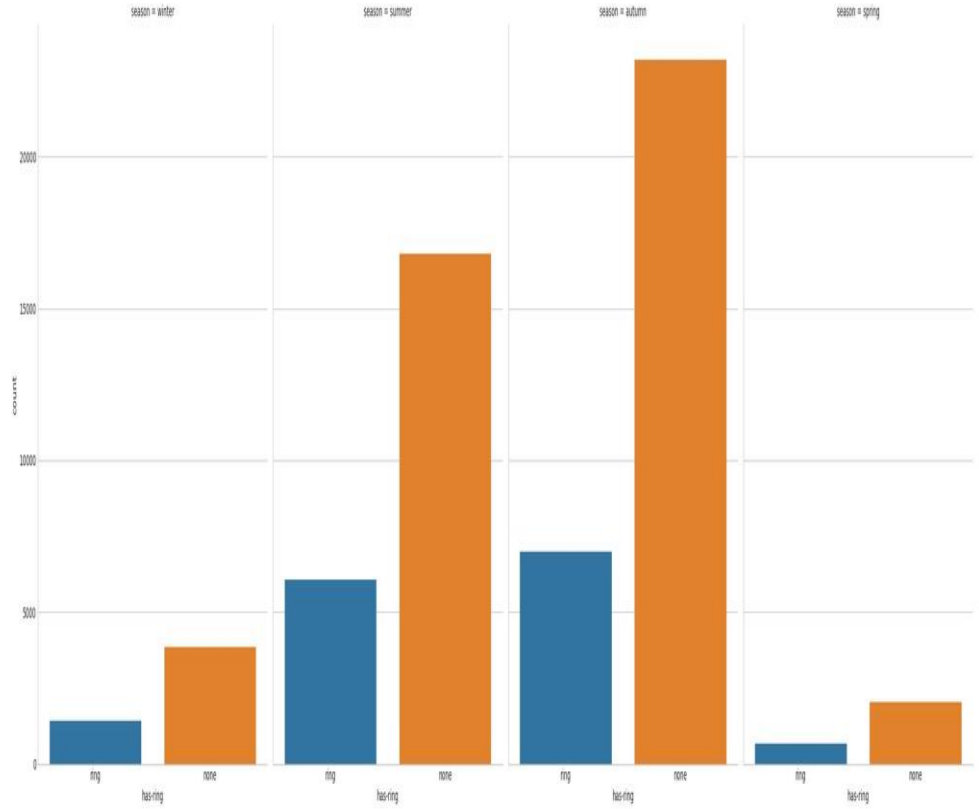
In [499…
```
pval = Asses_variable('season', 'has-ring')
pvalues_list.append(pval)
```

Table Original

season	none	ring
autumn	23184	6993
spring	2048	679
summer	16812	6086
winter	3846	1421

P-value is: 9.77662278130894 2e-22

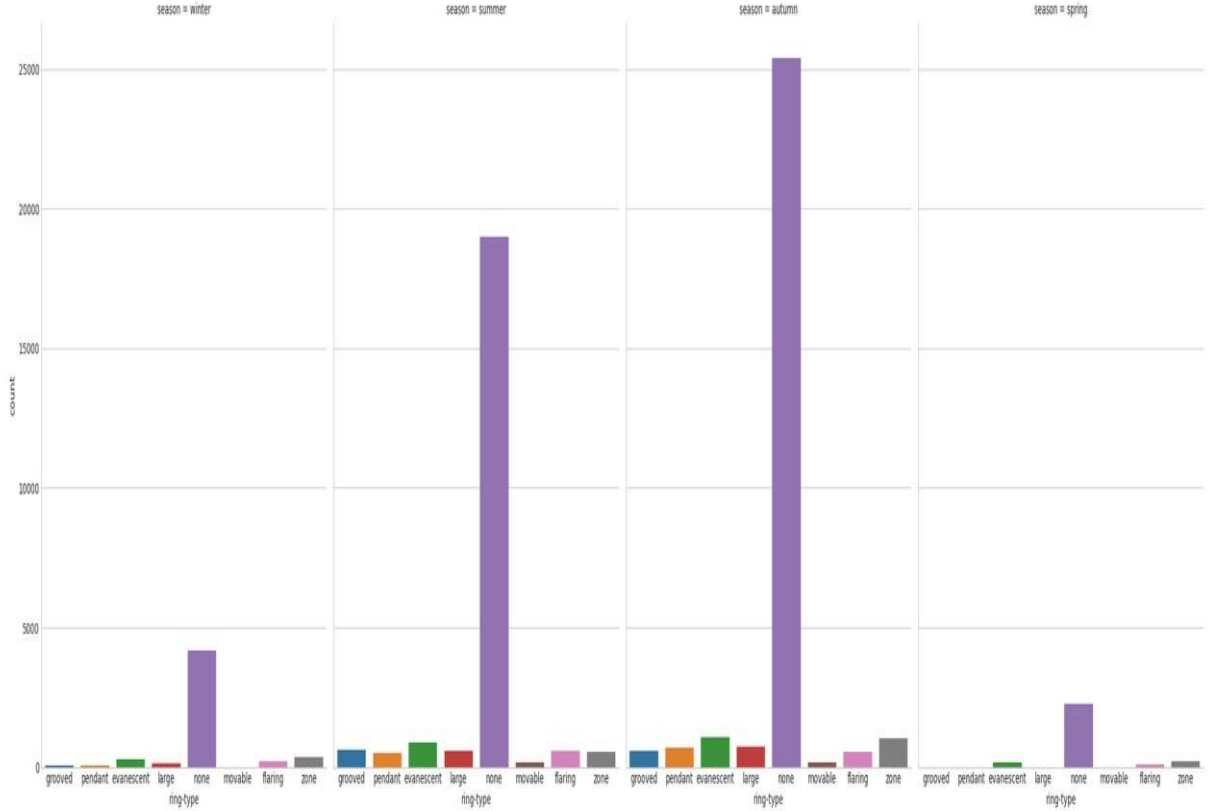
<Figure size 300x500 with 0 Axes>



In [500…

```
pval = Asses_variable('season', 'ring-type')
pvalues_list.append(pval)
```

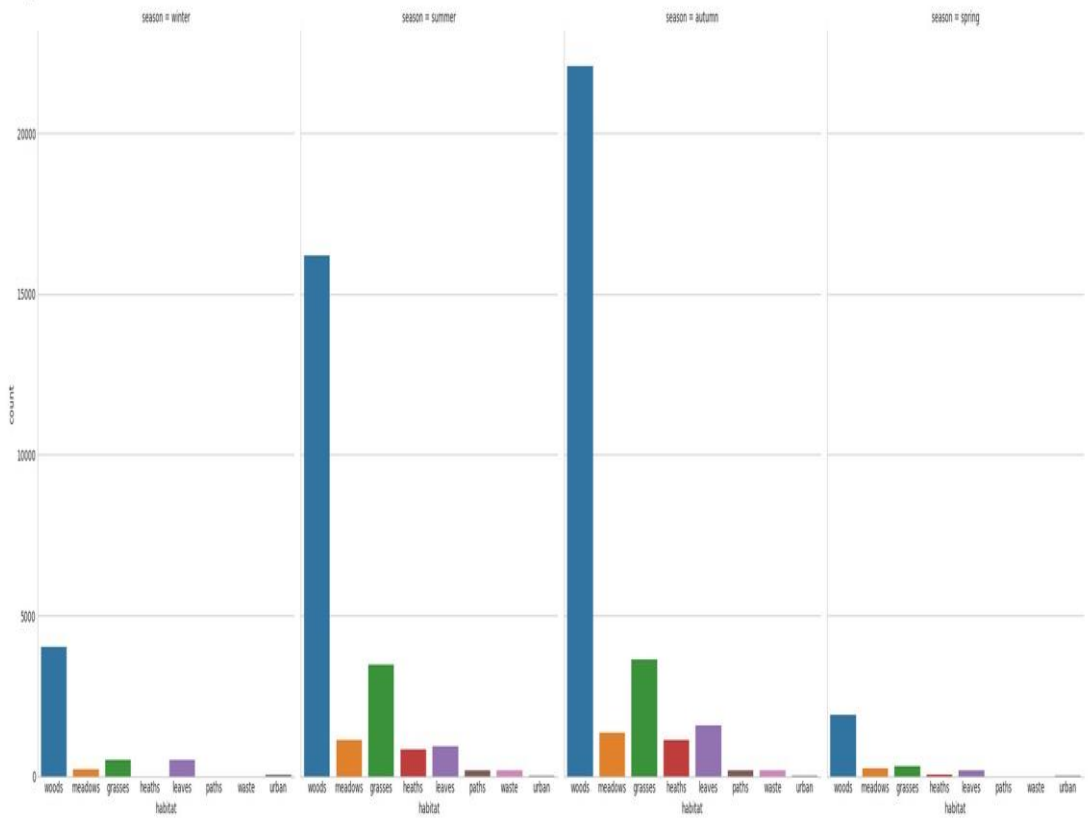
```
Table Original
ring-type  evanescent  flaring  grooved  large  movable   none  pendant  zone
season
autumn           1065      532      568    716      182  25396      703  1015
spring            190       82        0      0        0   2264        0   191
summer            890      593      608    591      171  18997      501   547
winter            290      192       64    120        0   4175       61   365
P-value is:  6.8355145544949244e-18
<Figure size 300x500 with 0 Axes>
```



In [501…

```
pval = Asses_variable('season', 'habitat')
pvalues_list.append(pval)
```

```
Table Original
habitat  grasses  heaths  leaves  meadows  paths  urban  waste  woods
season
autumn      3646    1135    1577     1341    194     27    172  22085
spring       323      46     168      240      0     27      0   1923
summer      3459     820     921     1135    166     25    181  16191
winter       515       0     502      204      0     36      0   4010
P-value is:  0.043513277939369516
<Figure size 300x500 with 0 Axes>
```



Logistic Regression – Random Search for Hyperparameters

```python
#Logistic Regression - Random Search for Hyperparameters

# Grid and Random Search
import scipy.stats as st
from scipy.stats import randint as sp_randint
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV


# Utility function to report best scores
def report(results, n_top=5):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")

# Specify parameters and distributions to sample from
param_dist = {'penalty': ['l2', 'l1'],
                         'class_weight': [None, 'balanced'],
                         'C': np.logspace(-20, 20, 10000),
                         'intercept_scaling': np.logspace(-20, 20, 10000)}

# Run Randomized Search
n_iter_search = 10
lrc = LogisticRegression(multi_class = "multinomial")
random_search = RandomizedSearchCV(lrc,
                                   n_jobs=-1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(random_search.cv_results_)
```

```
RandomizedSearchCV took 10.32 seconds for 10 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.492 (std: 0.000)
Parameters: {'penalty': 'l2', 'intercept_scaling': 5.701322363943065e+19, 'class_weight': None, 'C': 1.2633106315493951e-05}

Model with rank: 2
Mean validation score: 0.492 (std: 0.000)
Parameters: {'penalty': 'l2', 'intercept_scaling': 8.109440638402134e-11, 'class_weight': None, 'C': 2.1939873712343653e-08}

Model with rank: 2
Mean validation score: 0.492 (std: 0.000)
Parameters: {'penalty': 'l2', 'intercept_scaling': 1.2844312729241095e-13, 'class_weight': None, 'C': 3.808536093933881e-13}

Model with rank: 4
Mean validation score: 0.490 (std: 0.003)
Parameters: {'penalty': 'l2', 'intercept_scaling': 1972102542.266217, 'class_weight': None, 'C': 0.2978155669276477}

Model with rank: 5
Mean validation score: 0.488 (std: 0.004)
Parameters: {'penalty': 'l2', 'intercept_scaling': 3.917953758371066e-10, 'class_weight': None, 'C': 32.97249531200038}
```

```
RandomizedSearchCV took 3.36 seconds for 10 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.517 (std: 0.002)
Parameters: {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 10, 'max_features': 8, 'min_samples_leaf': 8, 'min_samples_split': 19}

Model with rank: 2
Mean validation score: 0.516 (std: 0.004)
Parameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 10, 'max_features': 10, 'min_samples_leaf': 10, 'min_samples_split': 4}

Model with rank: 3
Mean validation score: 0.516 (std: 0.004)
Parameters: {'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 3, 'min_samples_leaf': 2, 'min_samples_split': 8}

Model with rank: 4
Mean validation score: 0.516 (std: 0.006)
Parameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_features': 6, 'min_samples_leaf': 10, 'min_samples_split': 12}

Model with rank: 5
Mean validation score: 0.515 (std: 0.004)
Parameters: {'bootstrap': False, 'criterion': 'entropy', 'max_depth': None, 'max_features': 3, 'min_samples_leaf': 6, 'min_samples_split': 4}
```