Classification of Mushrooms in the Seasons they grow in, and Analyzing the Characteristics based on the Classification

Mohammed Mukarram Muneer Ahmed[1], Mounisha Madala[1], Rishitha Dammalapati[1], Sahithi Reddy Boda[1], Shreyas Aswar[1], Sreekar Adusumilli[1], Vishra shah[1].

[1]Indiana University-Purdue University, Indianapolis, IN 46202, USA

fnmoham@iu.edu, mmadala@iu.edu, sdammala@iu.edu, sboda@iu.edu, saswar@iu.edu, sradus@iu.edu, vishshah@iu.edu.

Abstract: The idea for this project is to understand relationship of the various characteristics of mushrooms. and the season they grow in to build a classifier that predicts the best season for mushroom growth. The salient features of mushrooms were examined from the dataset to find out how tightly or loosely features are corelated to each other and the season that they grow in. A quantitative approach is used by implying correlational study design. We used SQL and python for this study. Hypothesis testing is done by using chi-square analysis, where we rejected the null hypothesis and found out that all the features have significant relationship with season. For Improving accuracy, we did under sampling and oversampling with Synthetic Minority Oversampling Technique (SMOTE), where K-Nearest Neighbours Model and Random Forest Classifier Model gave the best predictions for mushroom growth.

Keywords: Mushroom. Season. Mushroom growth. Data Analysis. Data Visualization. SQL. Python.

1.Project Scope:

1.1 Introduction:

Mushrooms are one of those bell-shaped fungi which are known for their antioxidant, anti-inflammatory and anti-cancer effects. It is known about mushrooms that they possess healing and cleansing properties. Mushrooms are a 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” (Wagner et al., 2021). As with all other flora, different species of mushrooms require different climate conditions which facilitate growth.

The primary goal behind this project is to identify a favourable season for growth of different mushroom species. To facilitate this identification, a dataset containing 173 species of mushrooms from 23 closely related families is selected. This dataset has 21 feature attributes such as the diameter and shape of the head of the mushrooms, the stem height and width, habitat the mushroom grows in, etc. Out of these 21 features, 18 of them are nominal(qualitative) variables, and the rest are metrical(quantitative) variables.

1.2 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.

According to our research question, we framed our hypothesis as below:

- Null hypothesis - The characteristics of Mushrooms such as cap-shape, cap-surface, stem-diameter etc 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.

1.3 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? To answer such questions, we aim to examine the factors(variables) listed out below. We plan to build a classifier that predicts the best season for mushroom growth.

Variable Information:

● Nominal Variables - Shape of the mushroom cap, Surface of the mushroom cap, Color of the mushroom cap, Whether the mushroom bruise or bleed, Attachment of gill to the mushroom, spacing of gill to the mushroom, Color of the gill, Stem root, Color of the stem, Type of veil, Type of color of the veil, Ring is present or not, Type of ring, Color of the spore, Habitat, Season

● Metric Variables - Diameter of the mushroom cap, Width of the stem of mushroom, Height of the stem of mushroom

2.Methodology:

2.1 Steps of the Project:

Type of study: This is a correlational and classification study since we're going to find the relation between the different variables and based on the correlation, we are predicting the season in which a particular mushroom will grow in.

The methodology of our study involves 8 steps which will be explained in detail later in the report.

1. Data Description
2. Data Collection
3. Research Hypothesis
4. Selection of Research Method
5. Application of Research Method
6. Data Analysis
7. Application of Models
8. Performance Analysis

2.2 Original Team Members and Responsibilities

Our team offers a wide range of expertise and experience. Below is a list of the original team members we had for the project, along with the roles we decided on at the beginning.

Table.1. Original Responsibilities of team members for this project

Team Members	**Background**	**Responsibilities**
Shreyas Aswar	Bachelor of Engineering in Computer Science, prior expertise in SQL and Python.	Data cleaning, Project Proposal, Feature Selection, Random Forest Model Building.
Vishra Shah	Bachelor of Dental Surgery, MBA in Hospital and Healthcare Management.	Data Cleaning, Exploratory Data Analysis, Project Proposal, SVM Model building.
Sreekar Adusumilli	Bachelor of Technology in Biotechnology	Data cleaning, Feature Selection, Final Project Report, Test/ train Data set.
Mohammad Mukarram	Bachelor of Dental Surgery	Data Cleaning, Exploratory Data Analysis, KNN Model Building, Final project report.
Mounisha Madala	Pharm D (Doctor of Pharmacy)	Data Cleaning, Normality test using Python, Data visualization, Model Accuracy Evaluation.
Sahithi Reddy Boda	B.Sc.Biotechnology, MSc. Biochemistry and Molecular Biology.	Data Cleaning, Exploratory Data Analysis, Final project report, Naïve Bayes Model Building.
Rishitha Dammalapati	Bachelor of Dental Surgery	Data Cleaning, Data visualization, Debugging, Model Accuracy Evaluation.

2.3 Actual Contributions from Individual team members:

Initially we divided our responsibilities and started working according to that. But additional responsibilities came in during the work, so some of our duties were changed and we have worked according to our previous knowledge areas in the field.

Table.2. Revised Responsibilities of team members for this project

Team Members	**Background**	**Responsibilities**
Shreyas Aswar	Bachelor of Engineering in Computer Science, prior expertise in SQL and Python.	Project Proposal, Data Cleaning, SQL connection, Feature Selection, Project Presentation, Logistic Regression Model, Proofreading.
Vishra Shah	Bachelor of Dental Surgery, MBA in Hospital and Healthcare Management.	Project proposal, Data Visualization, Project Presentation, SVM model building, Stochastic Gradient Descent Model, Proofreading.
Sreekar Adusumilli	Bachelor of Technology in Biotechnology	SQL connection, Data cleaning, Project Report, Test/Train Data set, KNN Model Building.
Mohammad Mukarram	Bachelor of Dental Surgery	Project Proposal, Data visualization, Project Presentation, Random Forest Classifier Model.
Mounisha Madala	Pharm D (Doctor of Pharmacy)	Data cleaning, Project Report, Proofreading, Decision Tree Classifier Model, Model Accuracy Evaluation.
Sahithi Reddy Boda	B.Sc. Biotechnology, MSc. Biochemistry and Molecular Biology.	Data cleaning, Project Report, Exploratory Data Analysis, Naïve Bayes Model Building.
Rishitha Dammalapati	Bachelor of Dental Surgery	Feature selection, Data visualization, Debugging, Gradient Boosting Trees Model, Model Accuracy Evaluation.

3.Data Description:

● It was External Secondary Data Research. This data set includes descriptions of hypothetical samples corresponding to twenty-three 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**.**

4.Data Collection:

• We collected our Secondary Data from Mushroom Dataset from UCI Irvine Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/Secondary+Mushroom+Dataset

•*Number of Records* –There were 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.

5.Data Extraction and Storage

5.1 Data Extraction: The first step in the data extraction and storage stage was the selection of attributes from our files. We did a thorough review of the attributes in all the different files and chose the attributes that were most in line with our aim. Our aim was "to build and evaluate different machine learning models that classify the mushroom species and predict the best season to grow them."

The 21 attributes we selected below are:

1. Shape of the Mushroom cap
2. Surface of the mushroom cap
3. Class of the Mushroom
4. Color of the mushroom cap
5. Whether the mushroom bruise or bleed
6. Attachment of gill to the mushroom
7. Spacing of gill to the mushroom
8. Color of the gill
9. Stem root
10. Stem surface
11. Color of the stem
12. Type of Veil
13. Type of color of the veil
14. Ring is present or not
15. Type of Ring
16. Color of the spore
17. Habitat
18. Season
19. Diameter of mushroom cap
20. Stem width of mushroom
21. Stem height of mushroom

The explanation of the most essential metric variables we employed is provided below. They are:

- **Cap-diameter(m):** will it have a larger cap diameter?

  float number in cm

- **Stem-width(m):** What will be the average width of the mushroom stem that grows in summer?

  float number in mm

- **Stem-height(m):** Will a mushroom that grows in the monsoon have larger stem-height?

  float number in cm

The explanation of the most essential categorical variables we employed are provided below. They are:

Table.3. Showing Categorical variables from the dataset

Sr. No.	**Column Names**	**Attributes**
1	**class**	poisonous=p,edible=e(binary)
2	**cap-shape(n)**	bell=b, conical=c, convex=x, flat=f, sunken=s, spherical=p, others=o
3	**cap-surface(n)**	fibrous=i, grooves=g, scaly=y, smooth=s, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e, dented=d
4	**cap-color(n)**	brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k
5	**does-bruise-bleed(n)**	bruises-or-bleeding=t,no=f
6	**gill-attachment(n)**	adnate=a, adnexed=x, decurrent=d, free=e, sinuate=s, pores=p, none=f, unknown=?
7	**gill-spacing(n)**	close=c, distant=d, none=f
8	**gill-color(n)**	see cap-color + none=f
9	**stem-root(n)**	bulbous=b, swollen=s, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r
10	**stem-surface(n)**	see cap-surface + none=f
11	**stem-color(n)**	see cap-color + none=f
12	**veil-type(n)**	partial=p, universal=u

13	**veil-color(n)**	see cap-color + none=f
14	**has-ring(n)**	ring=t, none=f
15	**ring-type(n)**	cobwebby=c, evanescent=e, flaring=r, grooved=g, large=l, pendant=p, sheathing=s, zone=z, scaly=y, movable=m, none=f, unknown=?
16	**spore-print-color(n)**	see cap color
17	**habitat(n)**	grasses =g, leaves=l, meadows=m, paths=p, heaths=h, urban=u, waste=w, woods=d
18	**season(n)**	spring=s, summer=u, autumn=a, winter=w (Wagner et al., 2021)

5.2 Data Import:

The first step, as in any data modelling project, is to import the data from the dataset.

- We imported the secondary dataset into the 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 data frame by connecting them using Pymysql Library. We usedthe file ‘names.txt,’ which stored the abbreviation and full forms of Mushroom’s attributes, to rename the entries in our dataset to their respective full forms.

SQL CONNECTION

```
# 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)
```

Fig.1. Importing libraries for SQL connection

6. Data Analysis:

We must evaluate, examine, and summarize the important data contained in the dataset. Therefore, we conducted exploratory data analysis to gain a basic understanding of the distribution of our data, null values, to assess the suitability of the statistical techniques we are considering for data analysis, and to examine the correlation between various variables. It enables a thorough comprehension of the dataset, the definition or rejection of hypotheses, and the solid construction of predictive models.

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)
```

Fig.2. Exploratory Data Analysis

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
```

Fig.3. Information about the data frame

6.1 Descriptive Statistics:

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. We used the describe() function to do descriptive statistics for both categorical and quantitative variables to learn about the unique, top, frequency, and count, as well as the mean, standard deviation, minimum, and maximum values.

```
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

Fig.4. Describing all the categorical features

7. Data Cleaning:

Data cleaning is a very important step that 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 that create an imbalance in the data. We performed data cleaning in Python.

In the process of cleaning up the data, the first thing we did was use the df.dulplicated() function to locate duplicate values. We discovered that 146 rows have duplicate values, thus we need to get rid of those ones. Following the removal of the duplicate values using df.drop duplicates (), our dataset now has 60923 rows. Later, we discovered the null values using the df.isnull.sum() function, and we gained a thorough understanding of our dataset. We eliminated null values that represented over 20% of the total data. After eliminating the null values that were larger than 20%, we imputed the remaining null values using mode because all the null values are categorical, and mode will do the best to fill the categorical null values.

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

Fig.5. Deletion of Duplicate values

OBSERVATION OF NULL VALUES

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

Out[86]: <AxesSubplot:>

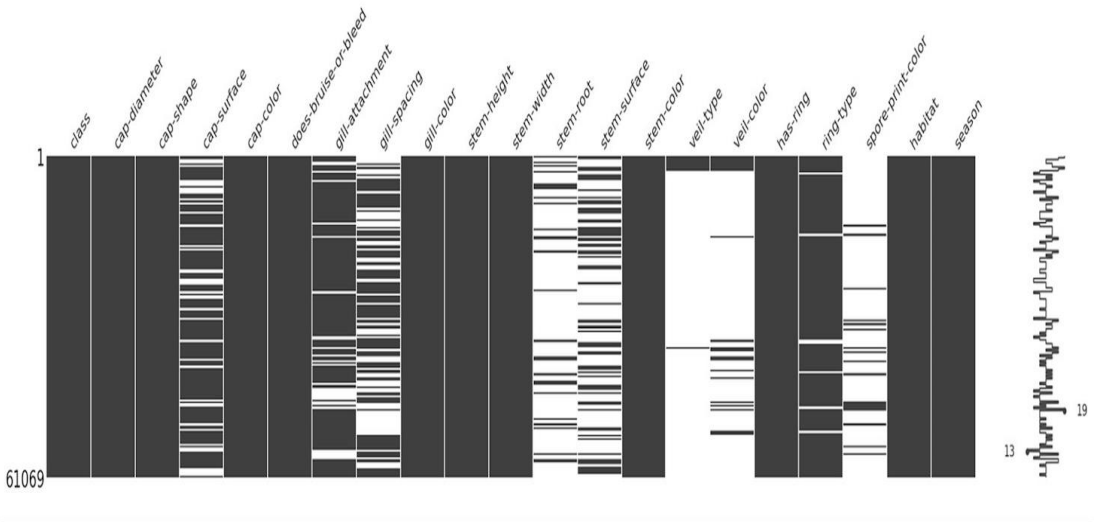


Fig.6. Observation of Null values

```
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)
```

Fig.7. Dropping of columns which are greater than 20% values

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
```

Fig.8. Imputing missing values with MODE

8. Data Correlation and Association:

Correlation is a statistical method for determining out whether there is a relationship between two variables or datasets and how significant that relationship could be. We performed correlation on all variables with each other to find out their strength and the linear relationship of the two variables with each other.

However, descriptive analysis revealed that we only had three numerical variables, and the remaining ones are all categorical data. As a result, we were unable to construct a correlation matrix using categorical data, so it is limited to numerical variables. We plotted a correlation matrix for continuous variables "stem-width," "stem-height," and "cap-diameter" by using Spearman correlation method to determine the relationship between them and a seaborn heatmap was used to visualize this matrix. We found that "stem-width" and "cap-diameter" were highly correlated. One of these highly correlated variables needed to be dropped as our model may give less importance to other variables. We decided to drop the variable "stem-width."

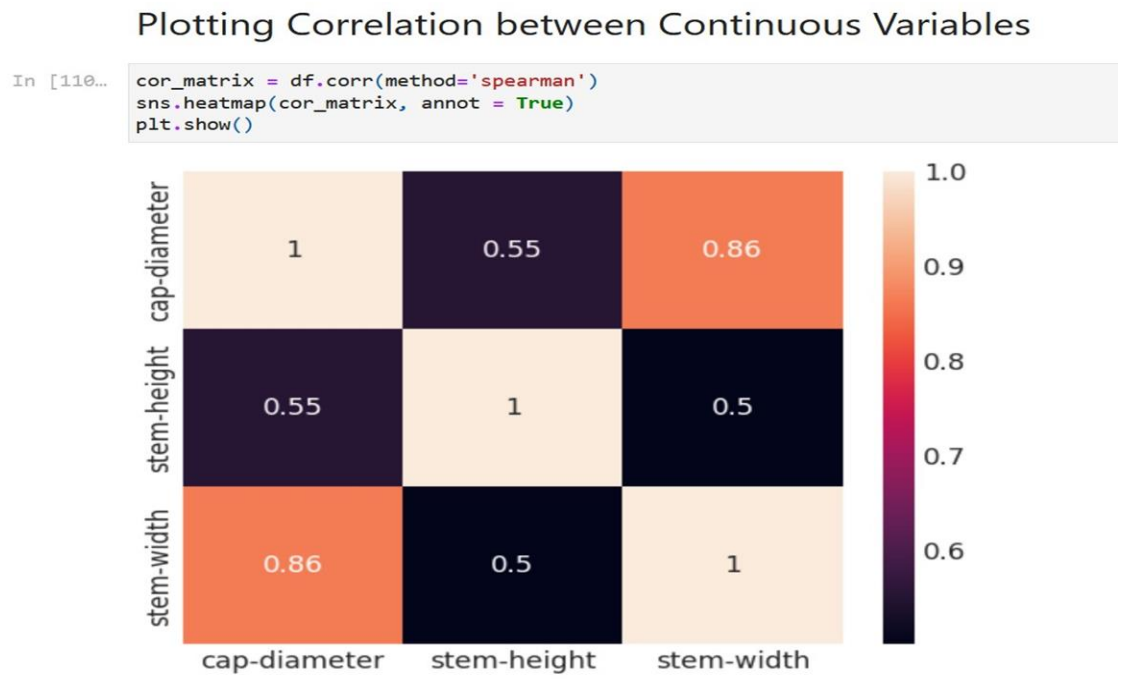


Fig.9. Plotting correlation between Continuous variables by using Spearman Correlation Method

9.Data Visualization:

We are trying to visualise all the variables after doing imputation, deleting duplicating rows and deleting null values to understand the attributes of each variable and their distribution for each variable.

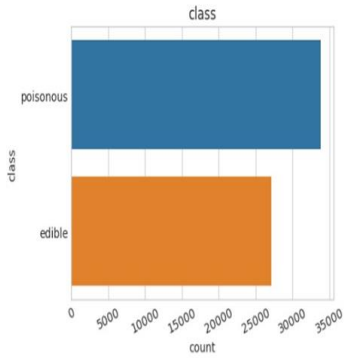
BIVARIATE ANALYSIS AND COLUMN DESCRIPTION

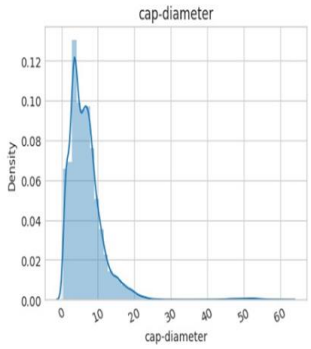
For the variable named cap surface, we have added dented=d after researching on internet

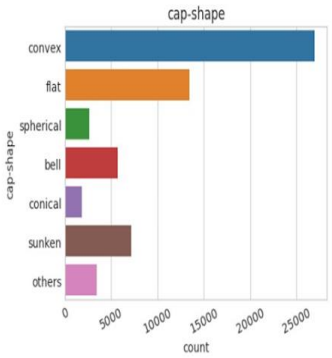
In [107…

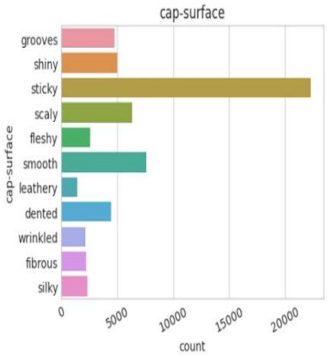
```
def replace_data_with_attributes(df) :
    with open('./names.txt') as file:
        for line in file.readlines():
            line = line.split(':')
            columnName = line[0].strip()
            values = line[1].split(',')
            for value in values:
                value = value.split('=')
                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)
```

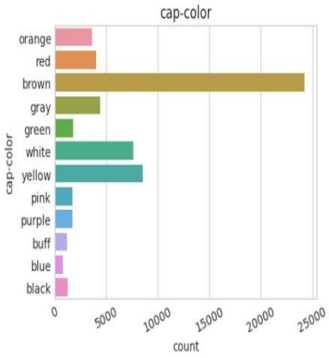
Fig.10. Bivariate Analysis and plotting the distribution of each feature

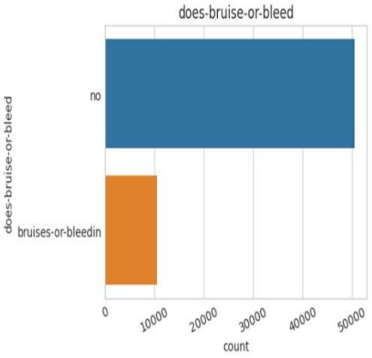

class
poisonous
edible
class
0
5000
10000
15000
20000
25000
30000
35000
count

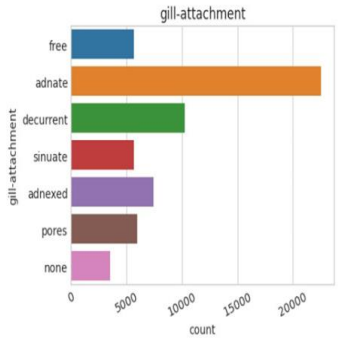

cap-diameter
Density
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
50
60
cap-diameter

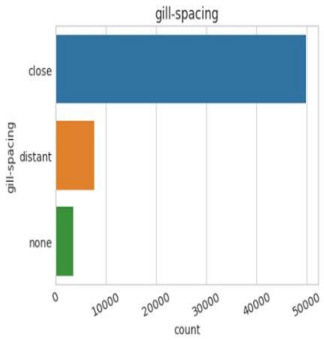

cap-shape
convex
flat
spherical
bell
conical
sunken
others
cap-shape
0
5000
10000
15000
20000
25000
count

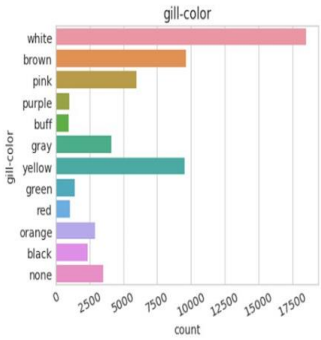

cap-surface
grooves
shiny
sticky
scaly
fleshy
smooth
leathery
dented
wrinkled
fibrous
silky
cap-surface
0
5000
10000
15000
20000
count

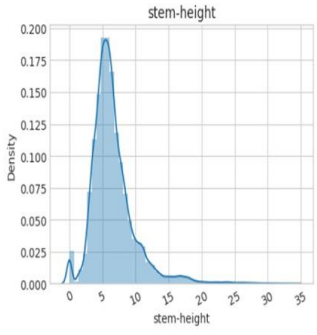

cap-color
orange
red
brown
gray
green
white
yellow
pink
purple
buff
blue
black
cap-color
0
5000
10000
15000
20000
25000
count

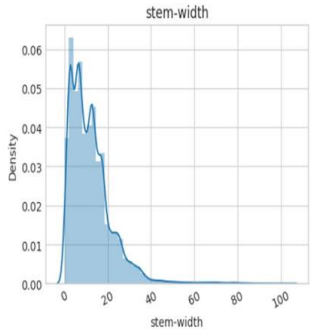

does-bruise-or-bleed
no
bruises-or-bleedin
does-bruise-or-bleed
0
10000
20000
30000
40000
50000
count

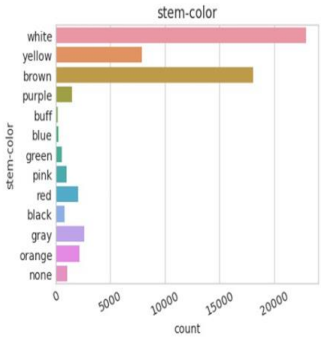

gill-attachment
free
adnate
decurrent
sinuate
adnexed
pores
none
gill-attachment
0
5000
10000
15000
20000
count


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


stem-width
Density
0.06
0.05
0.04
0.03
0.02
0.01
0.00
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

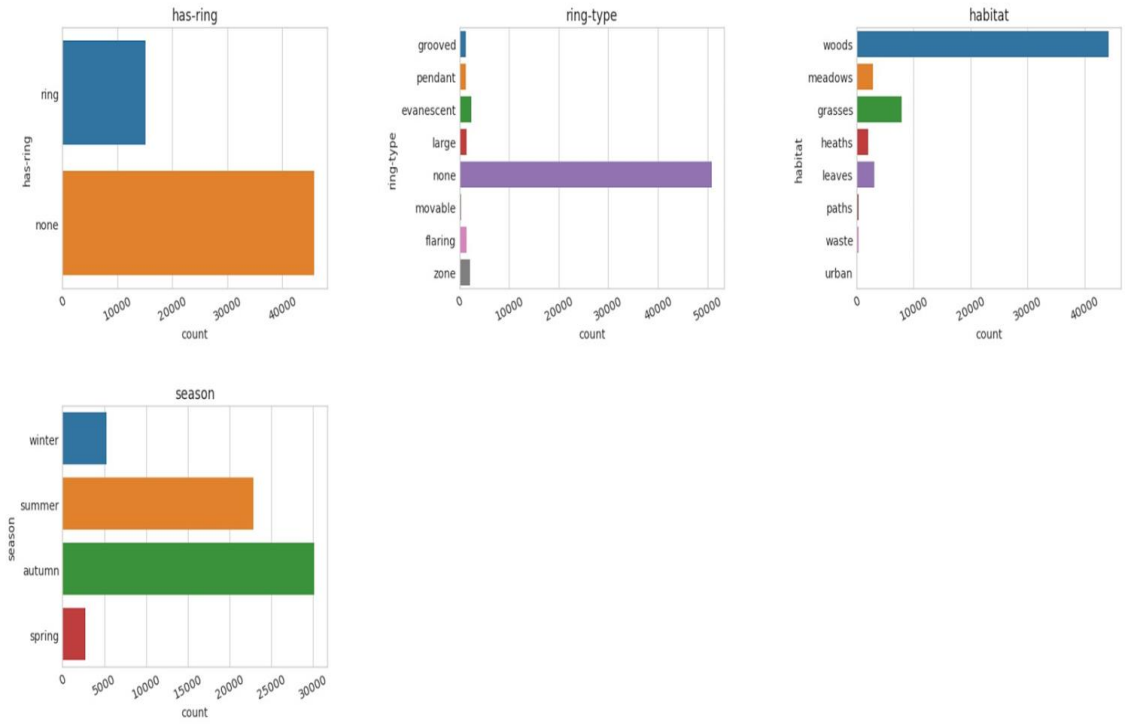


Fig.11,12,13. are the plots showing the frequency distribution of the columns with their attribute names for all the columns after imputing the data.

10.Creation of a New Data frame:

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

Fig.14. Showing creation of new data frame to perform data analysis

11.Hypothesis Testing:

We performed Chi-square testing because the rows and columns of a table are unordered (i.e., are nominal factors), and 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'
```

Fig.15. Mapping numbers to season

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
```

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.

Fig.16. Testing Hypothesis

For all the variables, we found a p value less than 0.05, hence we rejected the 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 conditions.

12.Feature Encoding:

Most machine learning models can only comprehend numerical data. As a result, it is necessary to convert the relevant features' categorical values into numerical ones. This Process is called Feature Encoding. The Three widely used procedures for translating categorical values to numeric values are carried out in two separate ways. Label Encoding, and One-Hot Encoding. Binary Coding. For our project, we used Label encoding and One hot encoding. Because label encoding makes it appear as though values are ranked, one hot encoding is the preferred method for converting a categorical variable into a numerical variable in the majority of cases. However, one disadvantage of one hot encoding is that you must create as many new variables as the original categorical variable has unique values. We tried both methods to encode data and made two separate databases for these encoding methods. One hot encoding is done by dividing the column into numerous columns; it transforms the categorical data into numeric data. Depending on which column contains the appropriate value, the numbers are substituted with 1s or 0s. The relatively straightforward label encoding strategy entails turning each value in a column into a number.

12.1 Label Encoding:

The process of changing the labels into a numerical form so that they can be read by machines is known as label encoding. The operation of those labels can then be better decided by machine learning algorithms. In supervised learning, it is a crucial pre-processing step for the structured dataset.

In label encoding in Python, we substitute a numerical value between 0 and the number of classes minus 1 for the categorical value. We use 0, 1, 2, 3, and 4 when the categorical variable value belongs to one of the five distinct classes.

Here we used LabelEncoder() and transformed the categorical data into numerical data so that it is easy for the machine to understand and helps in ML algorithms.

Feature Encoding

Label Encoding

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

Fig.17. Label Encoding for Categorical Variables

12.2 One Hot Encoding:

One hot encoding is the transformation of categorical variables into a form that ML algorithms could use to make better predictions.

It is the most common method used for encoding categorical data to numerical values. It works very well unless your categorical variable has many different Variables. Variables with more than 15 different values are usually not used with one hot encoding. With fewer values, it might not be the best option in some situations, but that varies. New binary columns are created by one hot encoding, indicating the presence of each potential value from the initial data. It also causes high memory consumption as there are large number of columns.

We dropped the target variable “season” and converted all the categorical values into numeric as show in the image below. We saved this encoded data into dataset_bin_enc.head().

One Hot Encoding

```
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	habit
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')
```

Fig.18. One hot encoding for Categorical Variables

12.3 Feature Correlation:

Correlation helps to distinguish one variable from another. The good variables and the target have a high correlation, so it makes sense to use correlation for feature selection. In addition, there should be a correlation between the variables and the goal but no correlation between them.

Predicting one attribute from another can be made easier with correlation it also indicates casual relationship between variables.

High correlation features are more linearly dependent and, as a result, have a similar effect on the dependent variable. As a result, when two features have a high correlation, we can eliminate one of them.

We plotted a heatmap by generating a seaborn plot to see the correlation between the both one-hot encoded and label encoded data frame. We performed feature scaling by using Standard Scaler, since our numerical variables vary more than our label encoded variables. Feature correlation is plotted for both datasets to see association between them.

12.4 CRAMER'S V Correlation:

It is a measure of how closely two nominal variables are linked to one another. It has the following elements: Zero means that the two variables don't have any relationship. 1 indicates a strong connection between the two variables.

At the point when there is in excess of a 2 X 2 possibility, Cramer's V addresses the affiliation or relationship between two factors. The relationship between two categorical variables can be examined using this strategy.

It allows to understand the correlation between two categorical features in one data set.

A seaborn correlation matrix was used to display the correlation, and this correlation was carried out with the assistance of chi_contingency from scipy.stats.

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
```

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

Fig.19. Cramer's V Correlation to see high correlated columns

```
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()
```

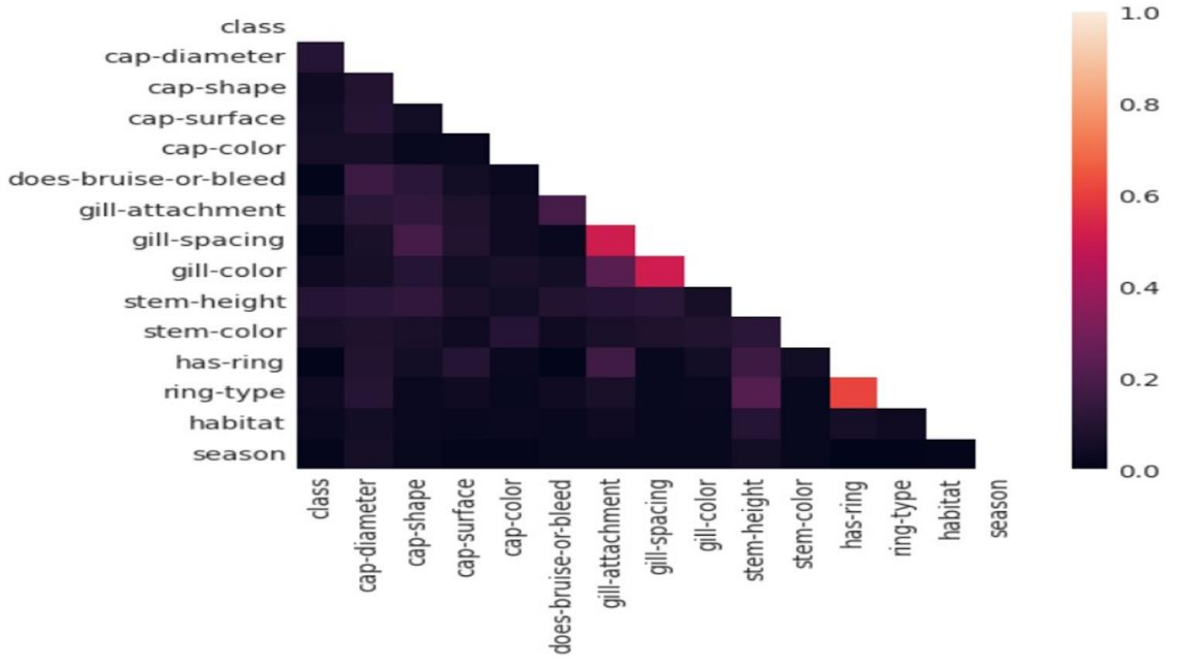


Fig.20. Plotting heatmap to see correlation between Season and other Variables

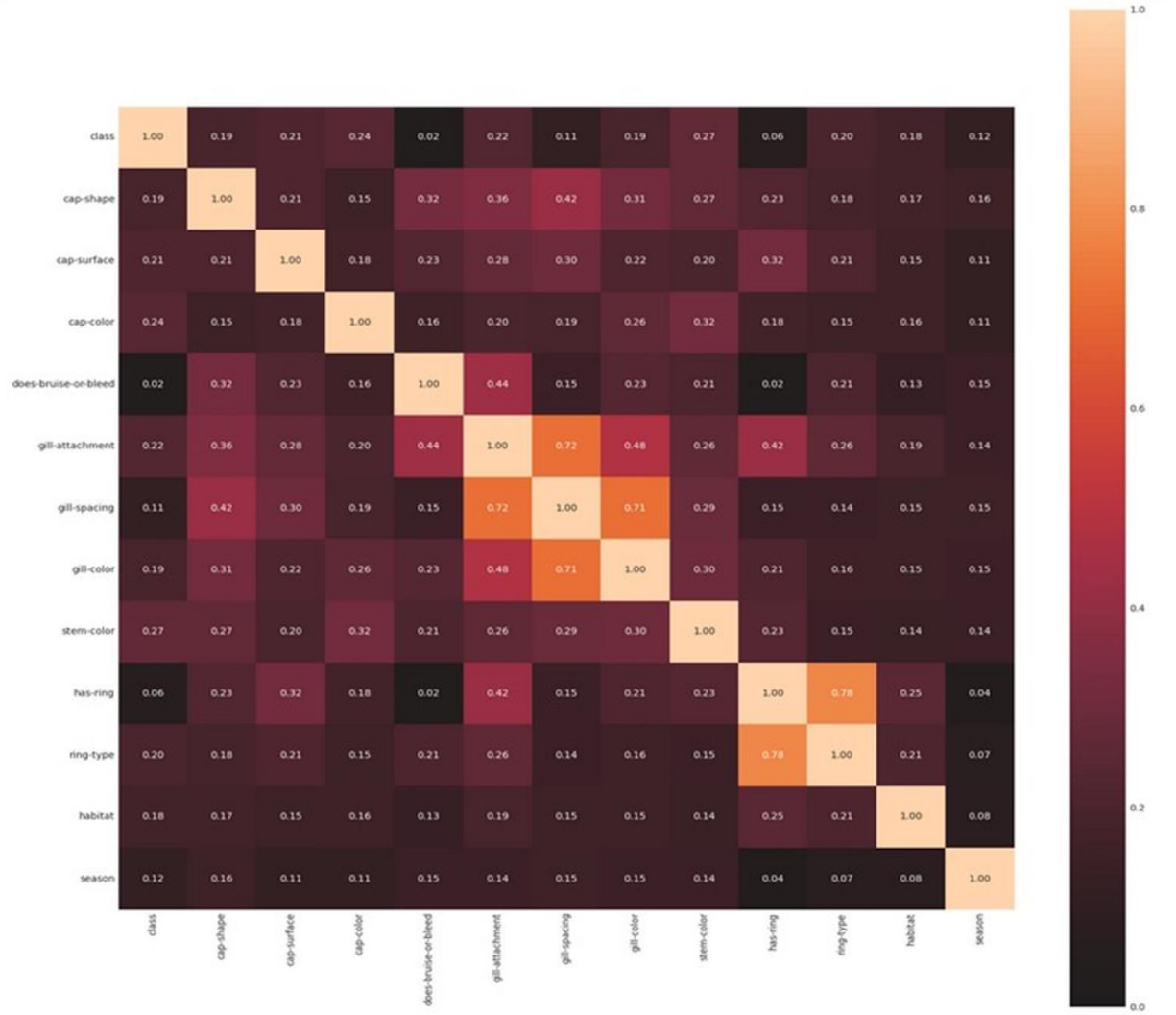


Fig.21. Plotting heatmap to see correlation between season and other variables

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

Fig.22. Deleting High correlated features ‘gill-spacing’, ‘has-ring’ and normalization

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())
```

Fig.23. Finding Correlation between 4 seasons and other one hot encoded feature

12.5 Feature Importance:

Techniques that calculate a score for each of a model's input features are referred to as "feature importance," and the scores simply represent the "importance" of each feature. A feature with a higher score indicates that it will have a greater impact on the model used to predict a particular variable.

We find out that ‘cap-diameter’ and ‘stem-height’ are very important, and this result was given to the ML by adding it to a database importance.

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));
```

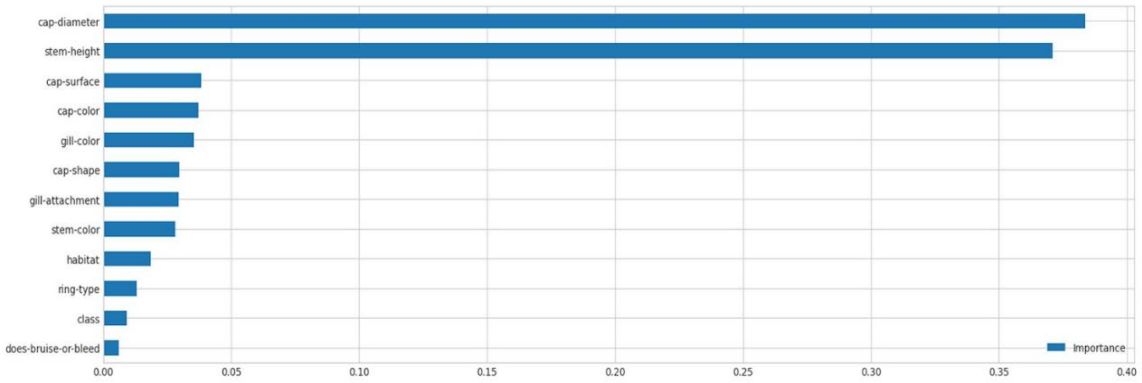


Fig.24. Using Random Forest to see importance of features

```
…
# 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_);
```

Fig.25. Calculating Principal Component Analysis (PCA) to observe variance between both data sets

As we see from the below graphs, we have better Variance for the Label encoding so we choose it as it can give us better accuracy for the Machine learning Algorithms.

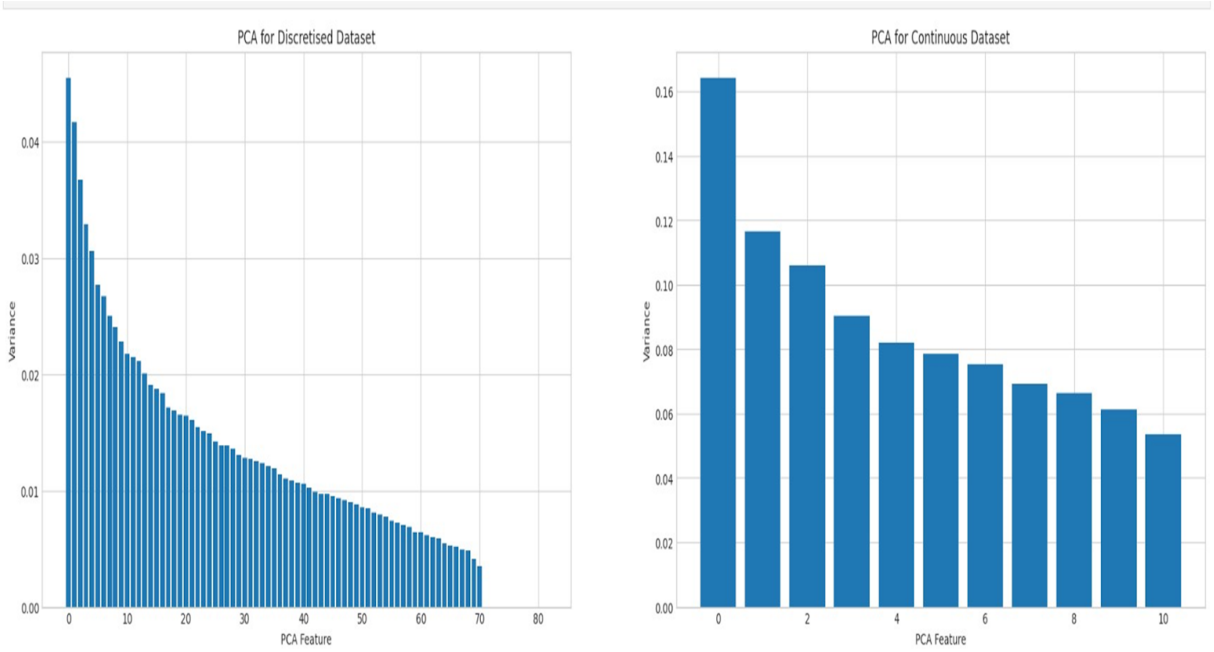


Fig.26. Variance for Discretised and Continuous Dataset

Here we split the data into Training and Testing data where we divide the data into 70:30 ratio.

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

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()
```

Fig.27. Selecting Label-encoded dataset

13.Application of Models:

We used 8 machine learning models to test and train the dataset. They are:

13.1 Logistic Regression Model

13.2 K – Nearest Neighbors Model

13.3 Gaussian Naïve Bayes Model

13.4 Linear SVC Model

13.5 Stochastic Gradient Descent Model

13.6 Decision Tree Classifier Model

13.7 Random Forest Classifier Model

13.8 Gradient Boosting Trees Model

13.1. Logistic Regression Model:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a Logistic Regression model. We then drew the ROC curve using all the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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)
```

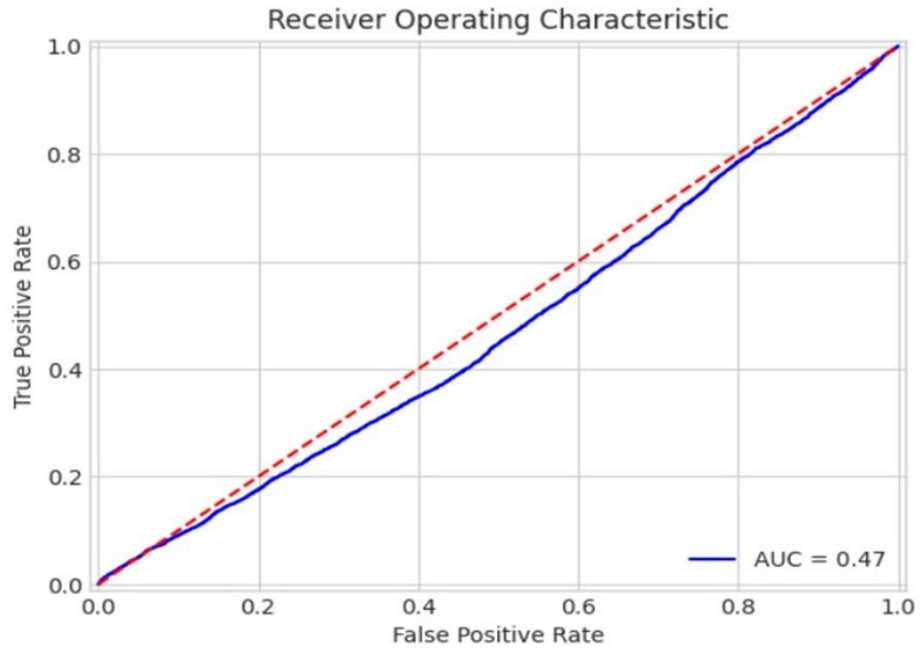


Fig.28. Performing Logistic Regression and plotting ROC curve between FPR and TPR

13.2.K – Nearest Neighbors Model:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a K-Nearest Neighbors model. We then drew the ROC curve using all of the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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

Fig.29. Performing K-Nearest Neighbors model and plotting ROC curve between FPR and TPR

13.3. Gaussian Naïve Bayes Model:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a Gaussian Naive Bayes model. We then drew the ROC curve using all of the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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
```

Fig.30. Performing Gaussian Naïve Bayes Model and plotting ROC curve between FPR and TPR

13.4. Linear SVC Model:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a Linear SVC model. We then drew the ROC curve using all of the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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))
```

```
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)
```

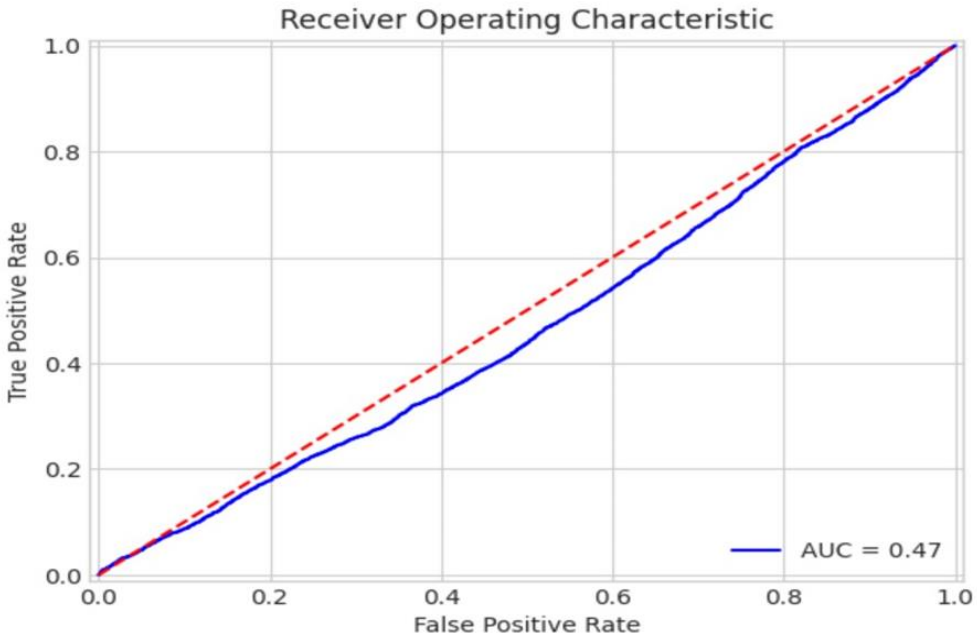


Fig.31. Performing Linear SVC Model and plotting ROC curve between FPR and TPR

13.5.Stochastic Gradient Descent:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a stochastic gradient descent model. We then drew the ROC curve using all the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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
```

```
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
```

Fig.32. Performing Stochastic Gradient Descent Model and plotting ROC curve between FPR and TPR

13.6. Decision Tree Classifier:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a decision tree classifier model. We then drew the ROC curve using all of the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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)
```

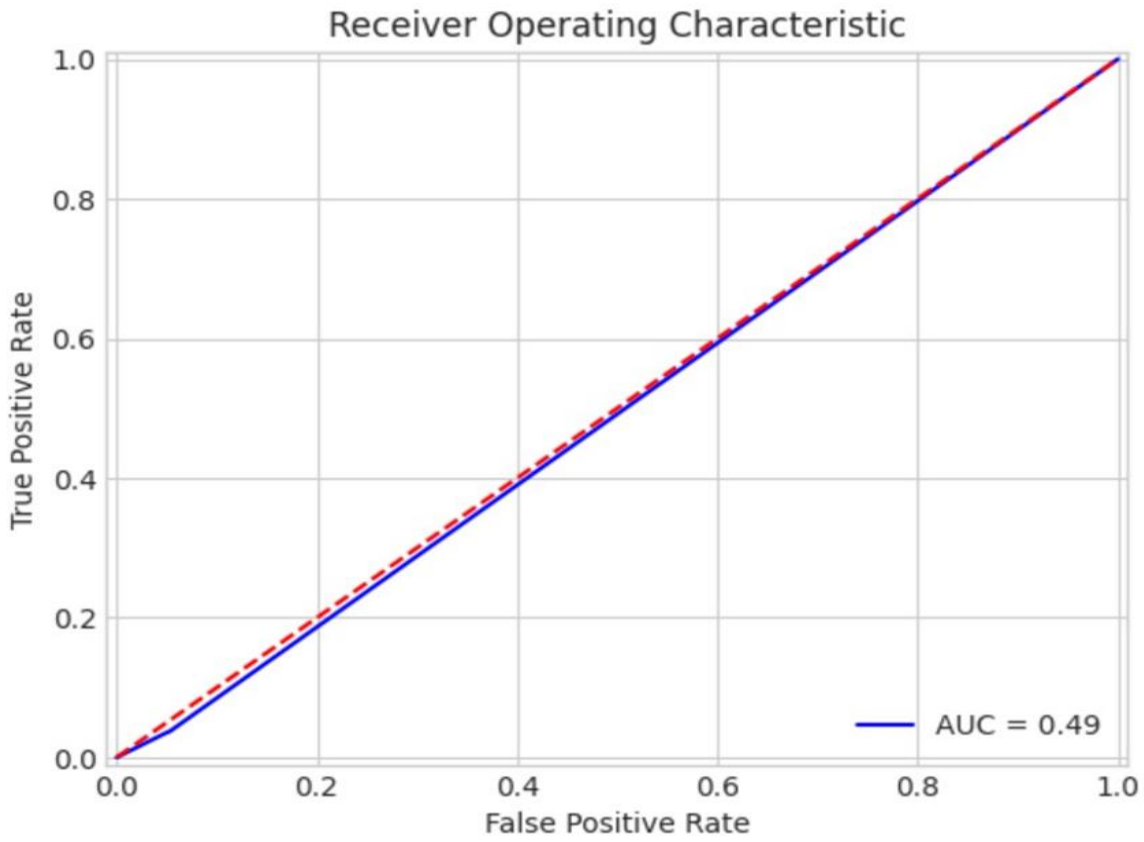


Fig.33. Performing Decision Tree Classifier Model and plotting ROC curve between FPR and TPR

13.7. Random forest classifier:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a random forest classifier model. We then drew the ROC curve using all of the independent variables as x and the dependent variable (season) as y. We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
#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
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
```

```
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
```

```
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
```

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

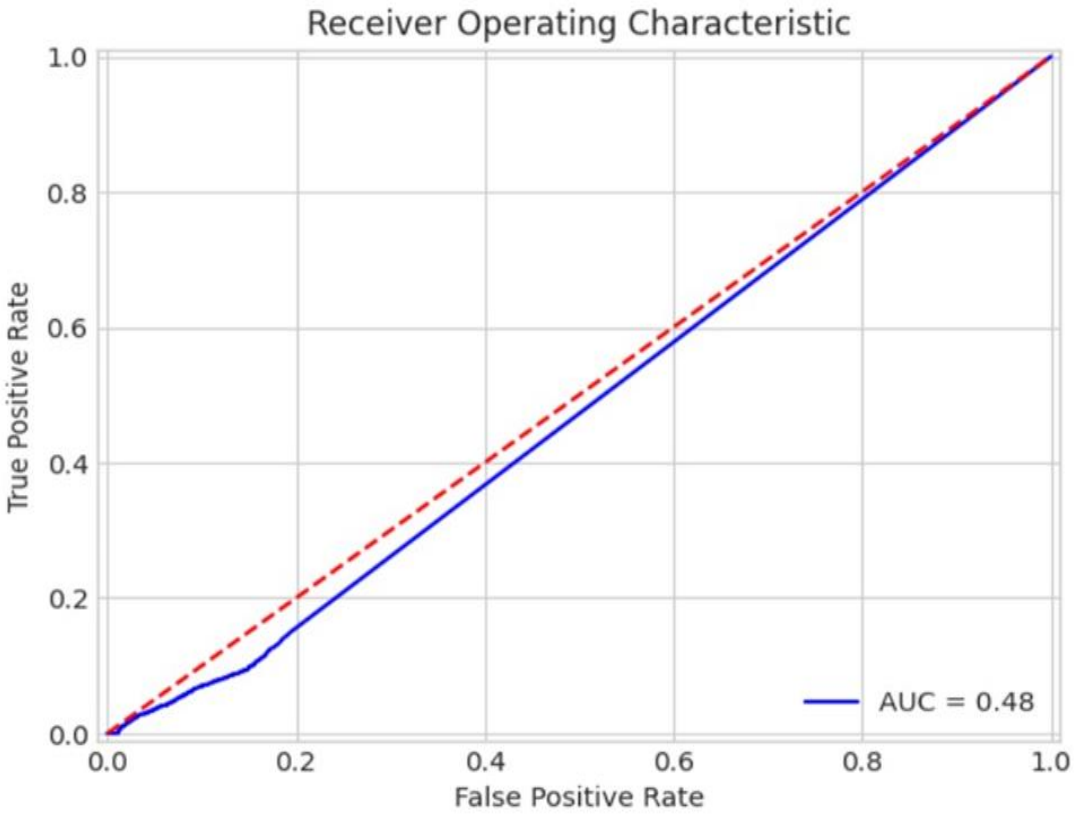


Fig.34. Performing Random Forest Classifier Model and plotting ROC curve between FPR and TPR

13.8. Gradient boosting trees:

By dividing the data into train and test data using the train_test_splitting as x_train, y_train, x_test, and y_test, we were able to construct a gradient boosting trees model. We then drew the ROC curve using all of the independent variables as x and the dependent variable (season) as y.

We then calculated the model's performance using the metrics precision, recall, f1 score, and support.

```
# 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
```

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

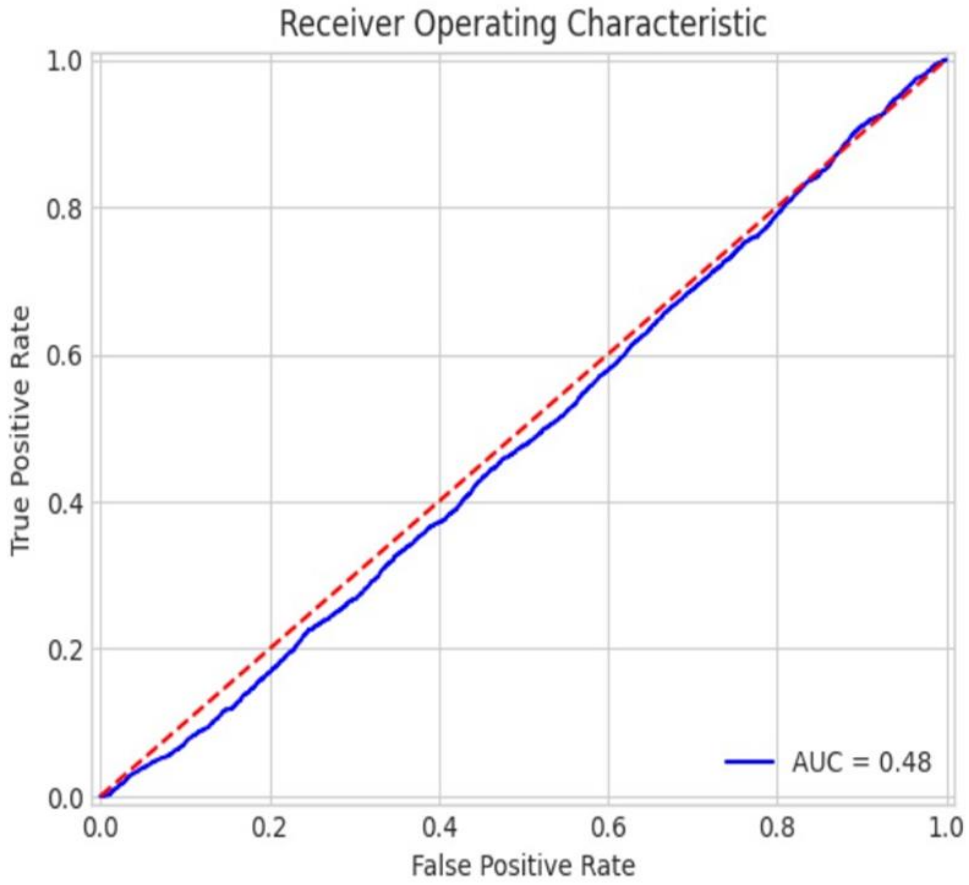


Fig.35. Performing Gradient Boosting Trees Model and plotting ROC curve between FPR and TPR

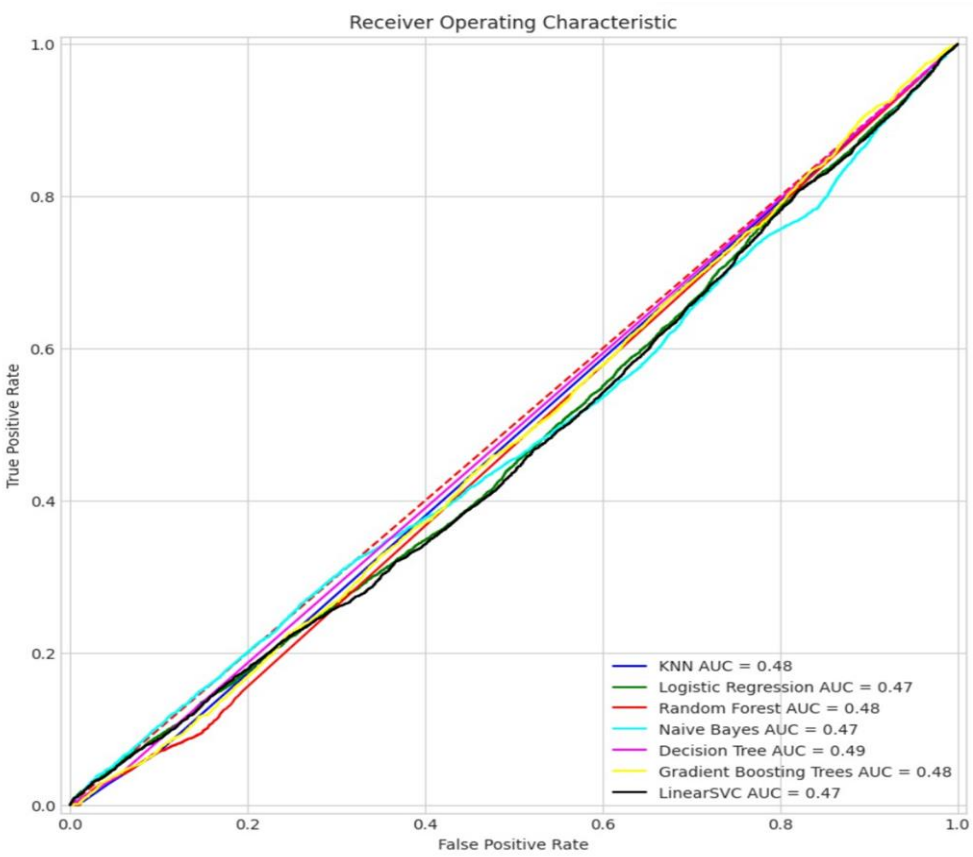


Fig.36. ROC Curve for all machine learning models

14. 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 correlated.

•For dealing with this problem, Oversampling and Under sampling on our datasets and Prediction Models were performed to increase the level of accuracy.

15.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)
```

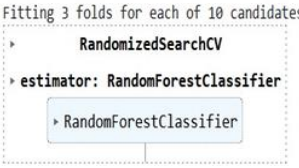
[744…]
```
X_train_us.shape
```

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]:



[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_
```

```
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

Fig.37. Performing Under Sampling by random Under sampler

Lower accuracy can be seen for Under Sampling and even after combining two minority classes, no significant difference was seen as the data is still highly imbalanced and for overcoming this problem over-sampling is tried with the minority classes.

SMOTE – Synthetic Minority Oversampling Technique has been used for overcoming the inaccuracy for imbalanced data as it uses the K-nearest approach to add synthetic data rather than just replicating the existing data.

16.Oversampling with SMOTE:

Oversampling is the method for balancing class distribution using synthetic minority oversampling by randomly increasing minority class. In this method, the Random Forest Classifier had the best accuracy score (82%), followed by the K-Nearest Neighbor model (71%), and the Gaussian Naive Bayes model (20%) had the lowest accuracy score.

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)
```

Fig.38. Performing Oversampling by Synthetic Minority Oversampling Technique

16.1 Random Forest on oversampled data:

Similar to how we handled raw data, we were able to create a Random Forest Model by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

Random Forest on Oversampled data

In [757…

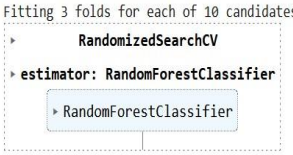
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 600, num = 10)]

# 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 [759]: best_random = rf_random_sm.best_estimator_
          best_random

Out[759]: RandomForestClassifier
          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
```

Fig.39. Performing Random Forest on Oversampled Data

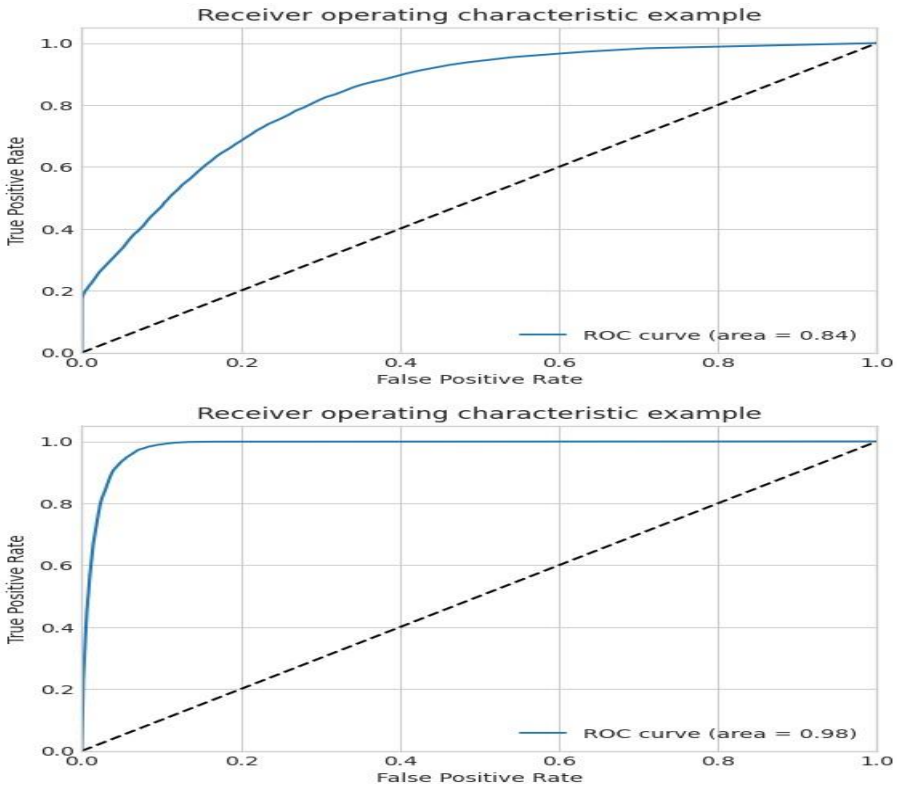


Fig.40. Plotting ROC curves between FPR and TPR for autumn and Spring Seasons

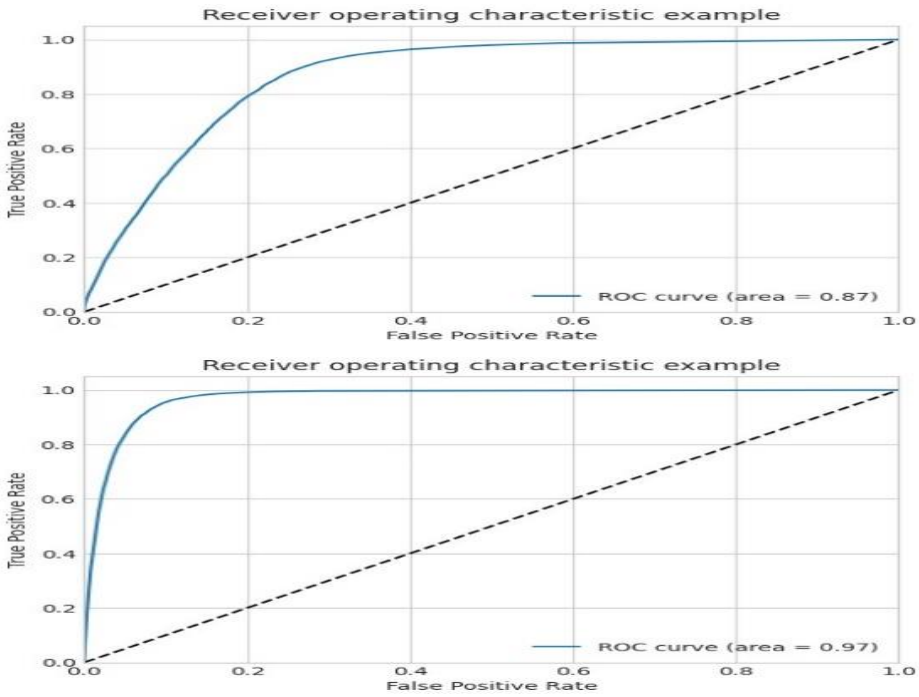


Fig.41. Plotting ROC curves between FPR and TPR for Summer and Winter Seasons

16.2. Logistic Regression on Oversampled Data:

Similar to how we handled raw data, we were able to create a Logistic Regression Model by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

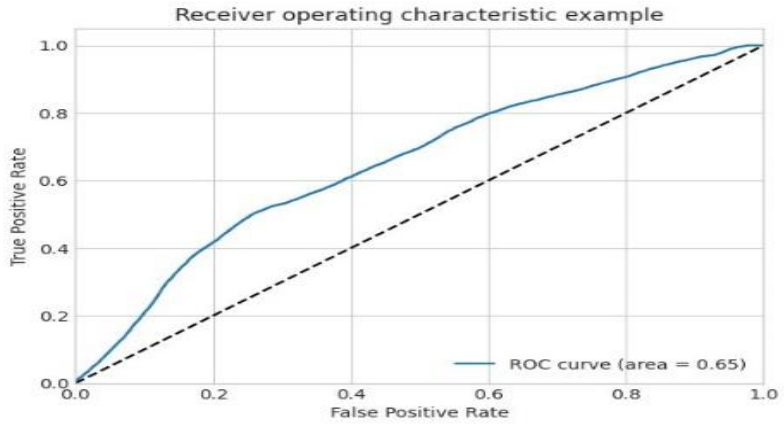
Logistic Regression on oversampled data

In [766…

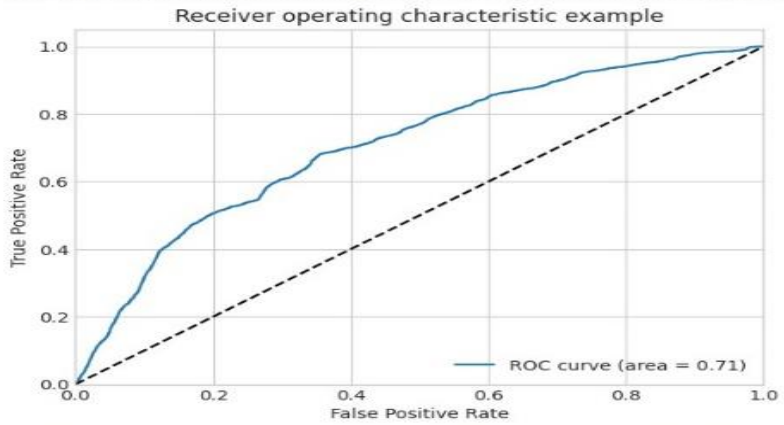
```
# # classifier
clf = OneVsRestClassifier(LogisticRegression(random_state=0))
y_score = clf.fit(X_train, y_train).predict_proba(X_test)

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: 0.037706481899884 52


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
```

Fig.42. Performing Linear regression model on Oversampled data

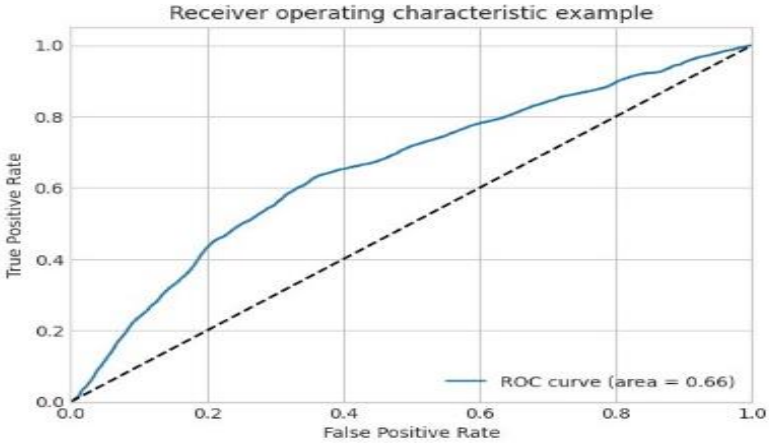



```
[CV] END bootstrap=False, max_depth=10, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   1.3s
```




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

Fig.43. Plotting ROC Curves between FPR and TPR for autumn and Spring seasons




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

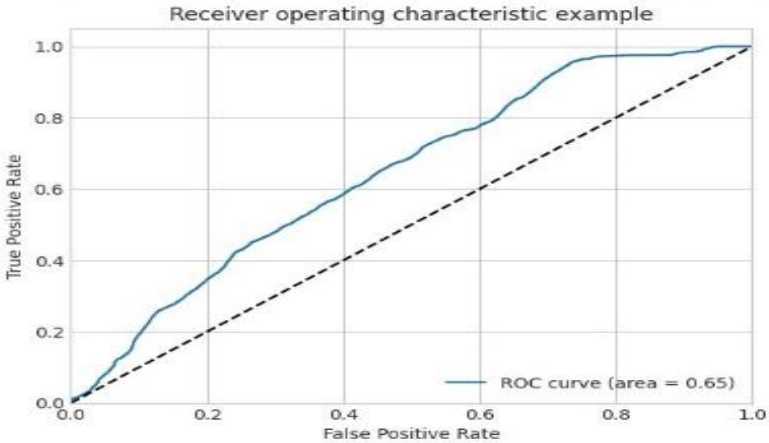


Fig.44. Plotting ROC Curves between FPR and TPR for Summer and Winter Seasons

16.3. KNN on Oversampled Data:

Similar to how we handled raw data, we were able to create a KNN Model by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

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

Fig.45. Performing KNN model on Oversampled data

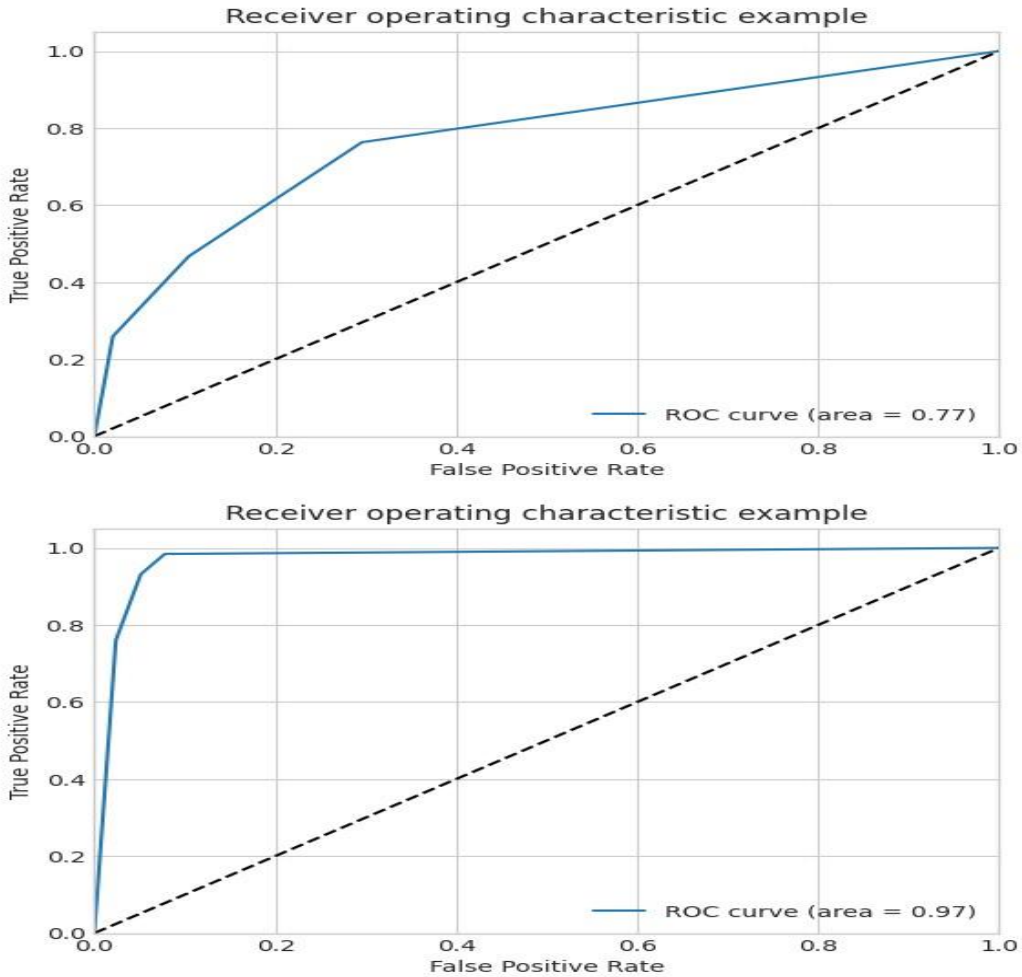


Fig.46. Plotting ROC Curves between FPR and TPR for autumn and Spring seasons

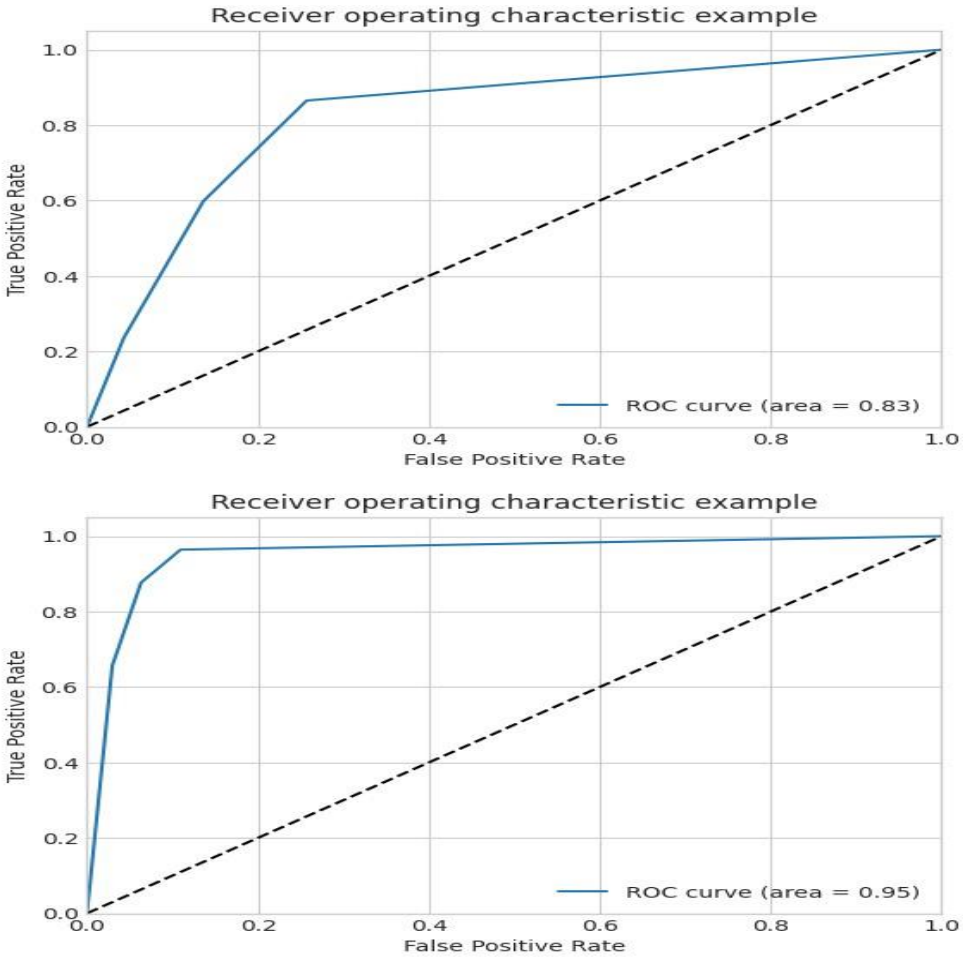


Fig. 47. Plotting ROC Curves between FPR and TPR for Summer and Winter Seasons

16.4. Gaussian Naive Bayes on oversampled data:

Similar to how we handled raw data, we were able to create a Gaussian Naive Bayes method by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

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

Fig.48. Performing Gaussian Naïve Bayes Model on Oversampled Data

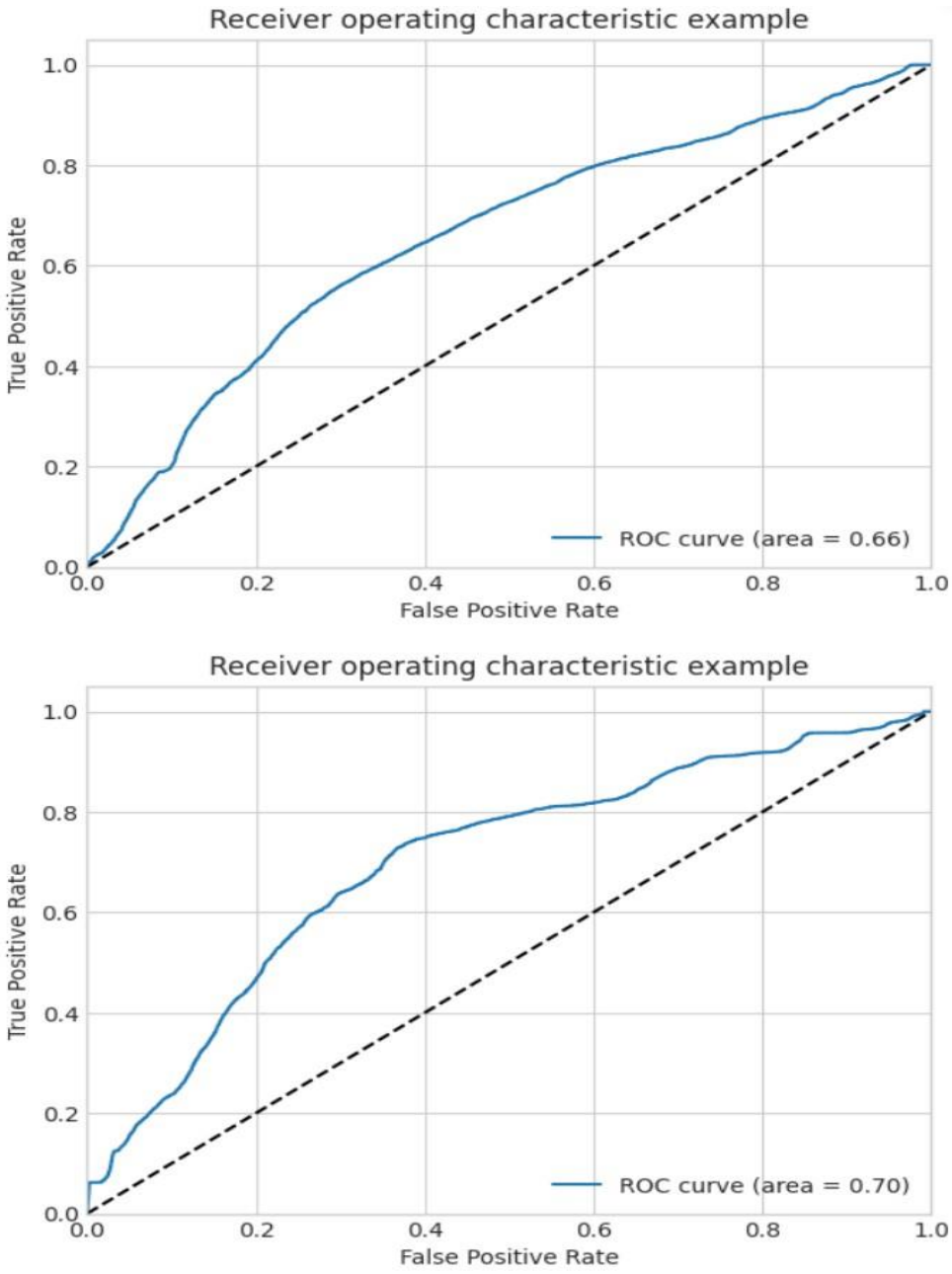


Fig.49. Plotting ROC curves between FPR and TPR for autumn and Spring Seasons

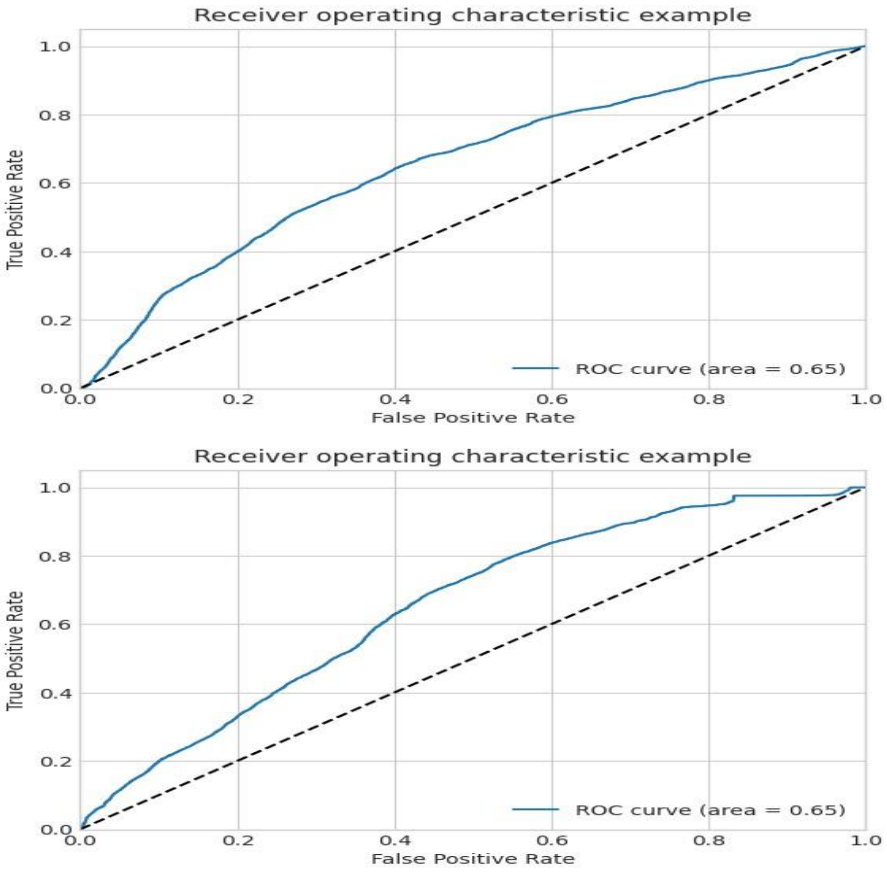


Fig.50. Plotting ROC Curves between FPR and TPR for Summer and Winter Seasons

16.5. Decision Tree Classifier on oversampled data:

Similar to how we handled raw data, we were able to create a Decision Tree Model by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

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

Fig.51. Performing Decision Tree model on Oversampled data

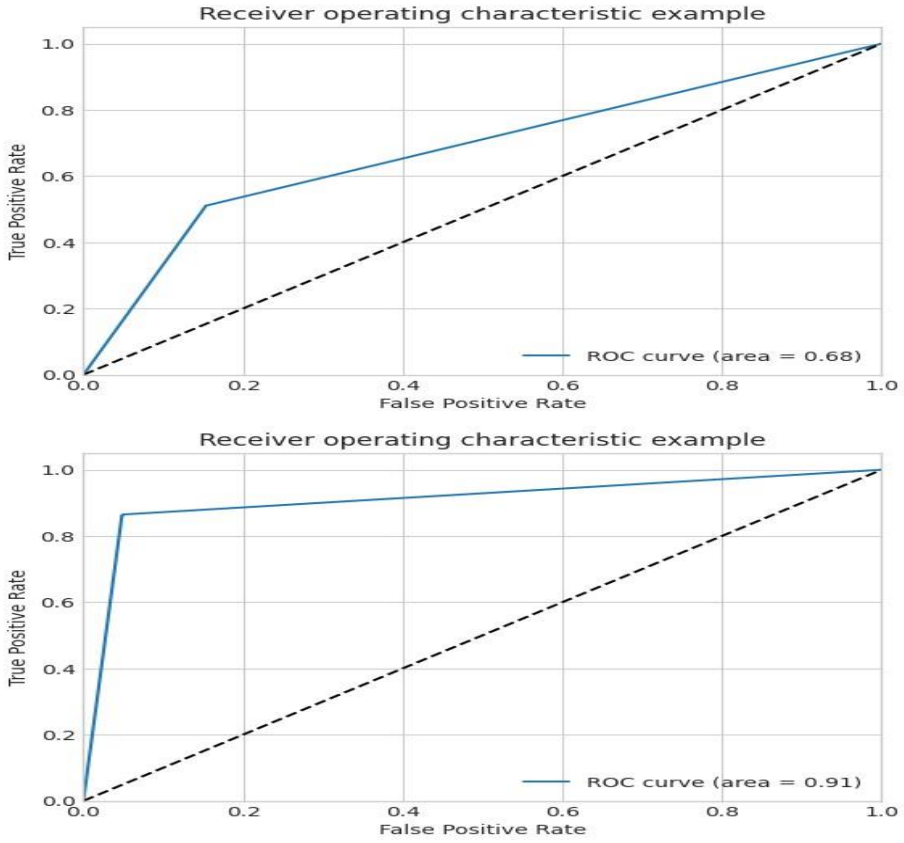


Fig.52. Plotting ROC curves between FPR and TPR for autumn and spring seasons

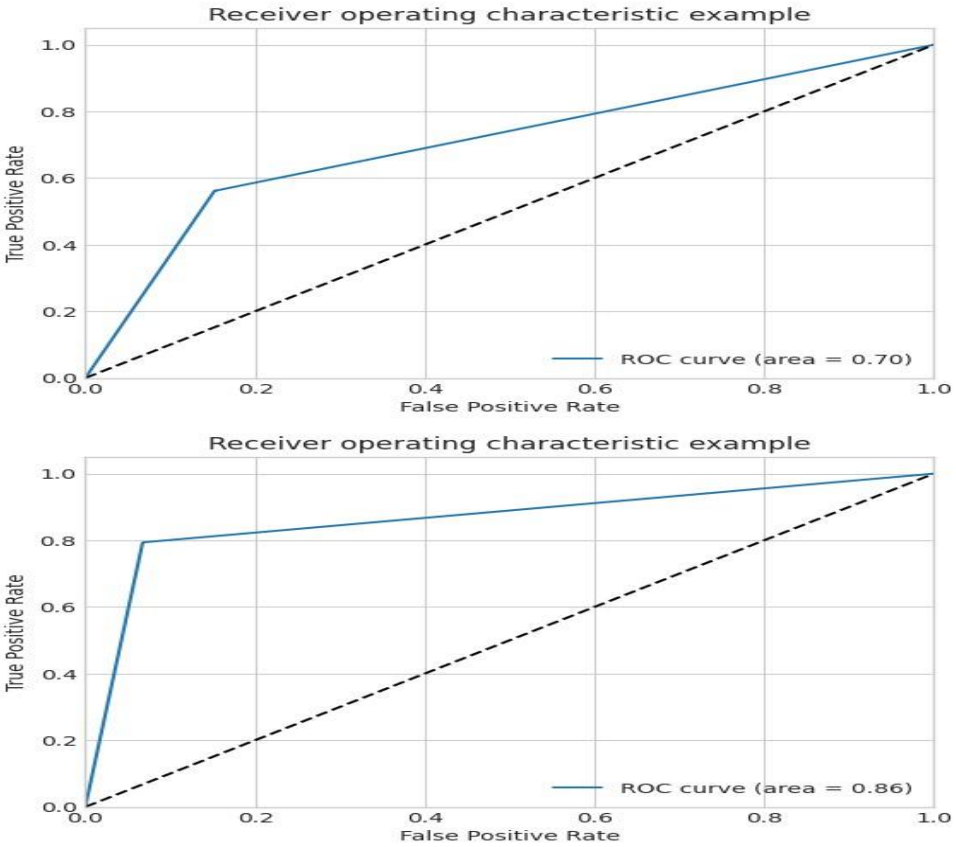


Fig.53. Plotting ROC curves with FPR and TPR for Summer and Winter Seasons

16.6. Gradient Boost on oversampled data:

Similar to how we handled raw data, we were able to create a Gradient Boost Model by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

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
```

Fig.54. Performing Gradient Boost model on oversampled data

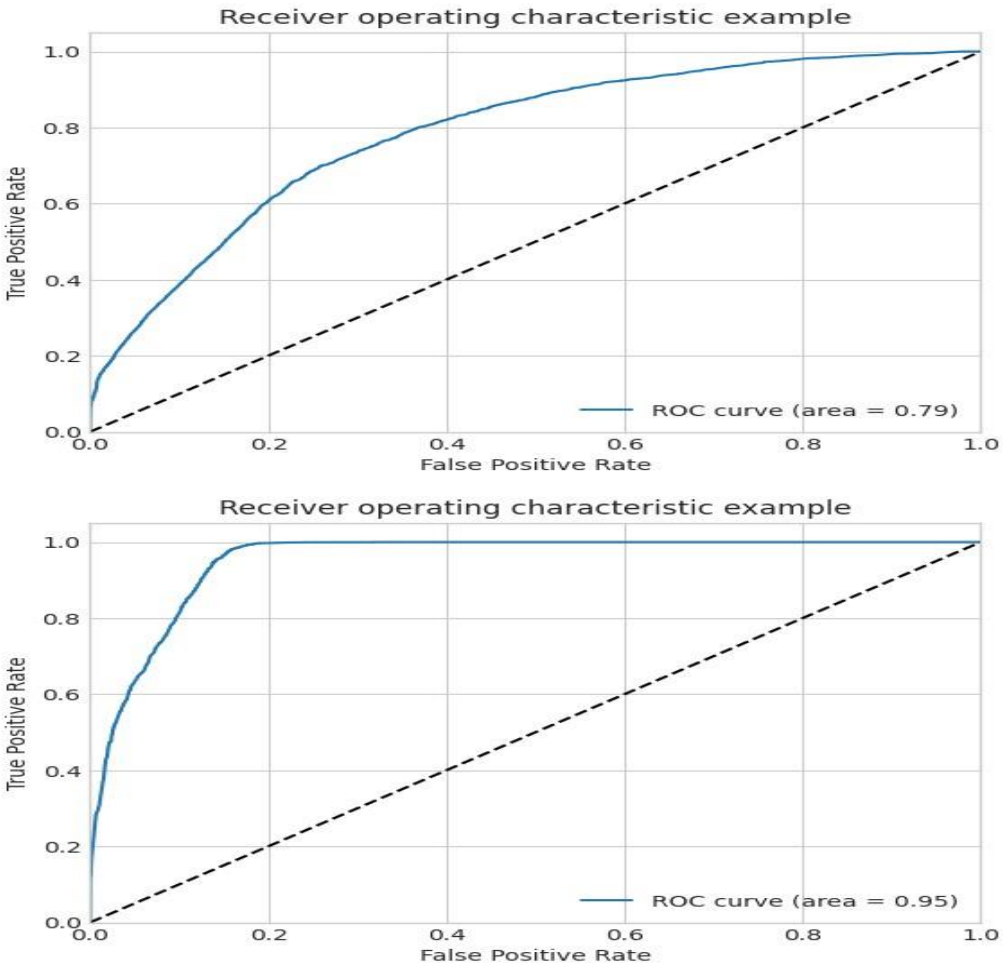


Fig.55. Plotting ROC curves between FPR and TPR for autumn and spring seasons

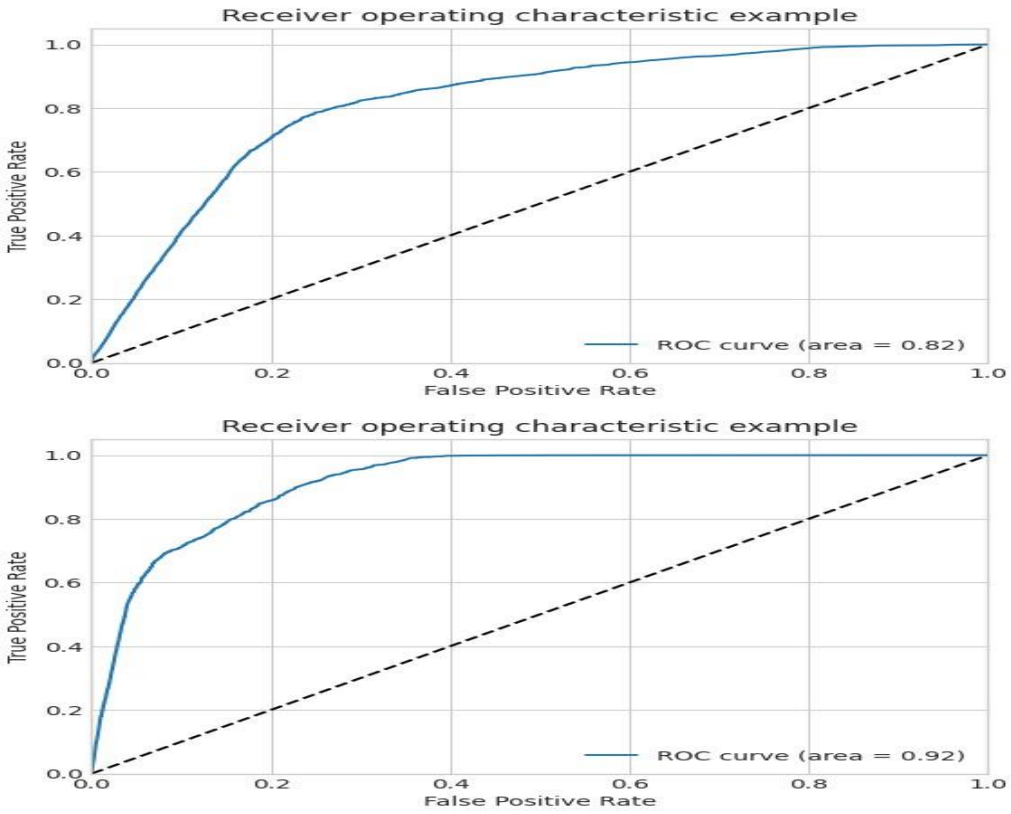


Fig.56. Plotting ROC Curves between FPR and TPR for Summer and Winter Seasons

16.7. Linear SVC on oversampled data:

Similar to how we handled raw data, we were able to create a Linear SVC Model by separating the data into train and test data using the train_test_splitting as x_train, y_ train, x_test, and y_test. Then, we used the dependent variable (season) as y and all the other variables as x to draw the ROC curve. The model's performance was then calculated using the metrics precision, recall, f1 score, and support.

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
```

Fig.57. Performing Linear SVC model on Oversampled data

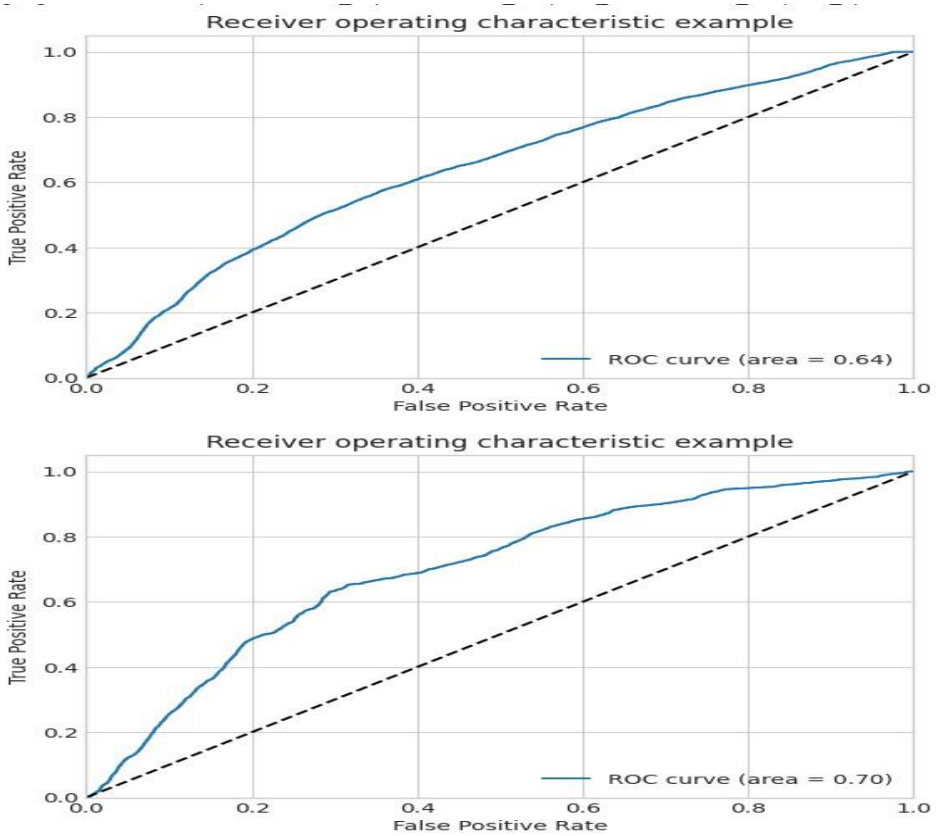
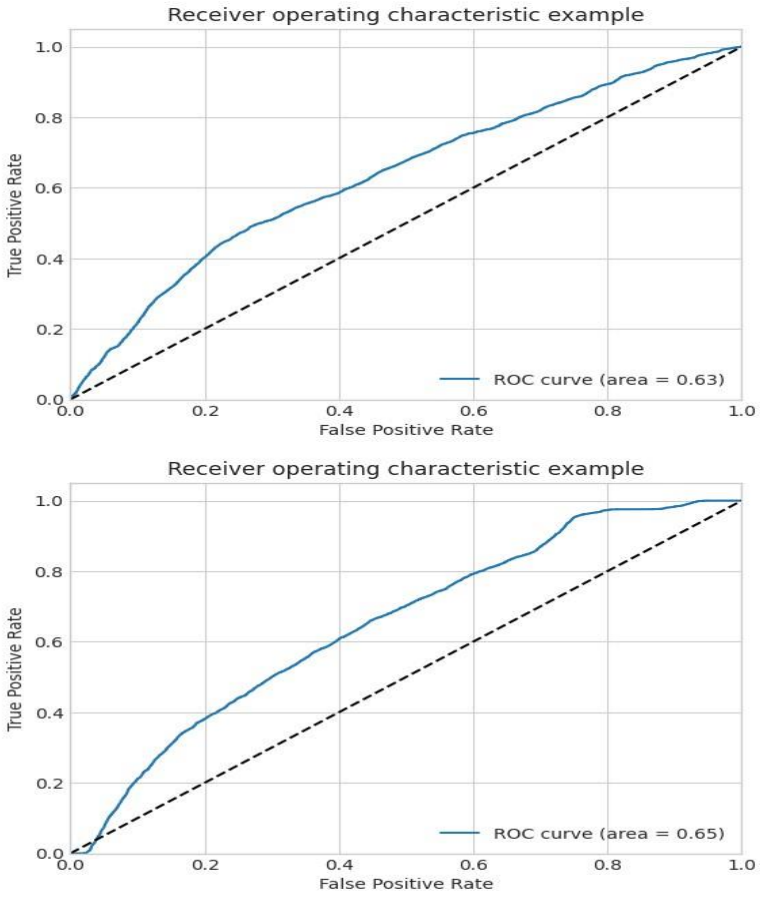


Fig.58. Plotting ROC curves between FPR and TPR for autumn and spring seasons




```
[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
```

Fig.59. Plotting ROC curves between FPR and TPR for Summer and Winter Seasons

17. Summary of Findings:

Table.4. Performance Analysis of Models

Machine Learning Algorithms	Accuracy in Percentage		Area Under the Curve (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 Naive 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

17.1. Performance Analysis of the models:

- For the Raw Data (not over/under sampled) almost all algorithms performed 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 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 class is significantly high.
- From "Learning from Imbalanced Datasets" (Page 55, 2018) we concluded that for Multiclass Classification problem accuracy is not always the best measure.

18.Limitations and Challenges:

- We found anomalies in the dataset, some entries with label 'd’ for a categorical variable have 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.

19. Learnings:

- Being from various backgrounds and study fields we understood together as a team how to divide the work amongst ourselves and learnt how to do Python and SQL as a part of this project.
- As a team, we came up with solutions for working with the imbalanced data and learnt that 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.

A. 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
```

Fig. A1. Libraries imported in python

Association between all the variables

```
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
```

Fig. A2. Association between all the variables

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

```
Out[87]: <AxesSubplot:>
```

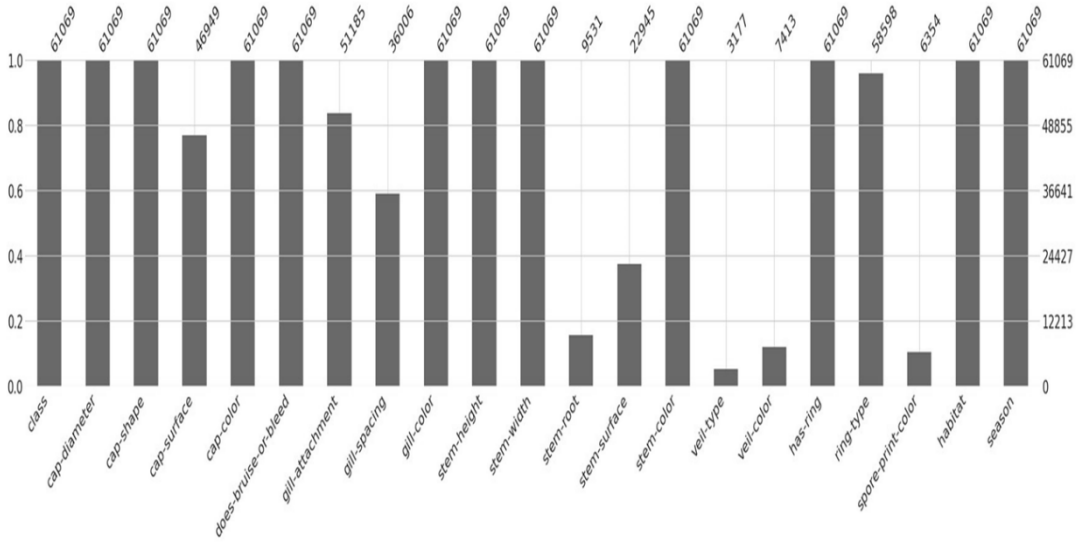


Fig. A3. Missingno Bar Plot

```
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:  0.30600176094994647
<Figure size 300x500 with 0 Axes>
```

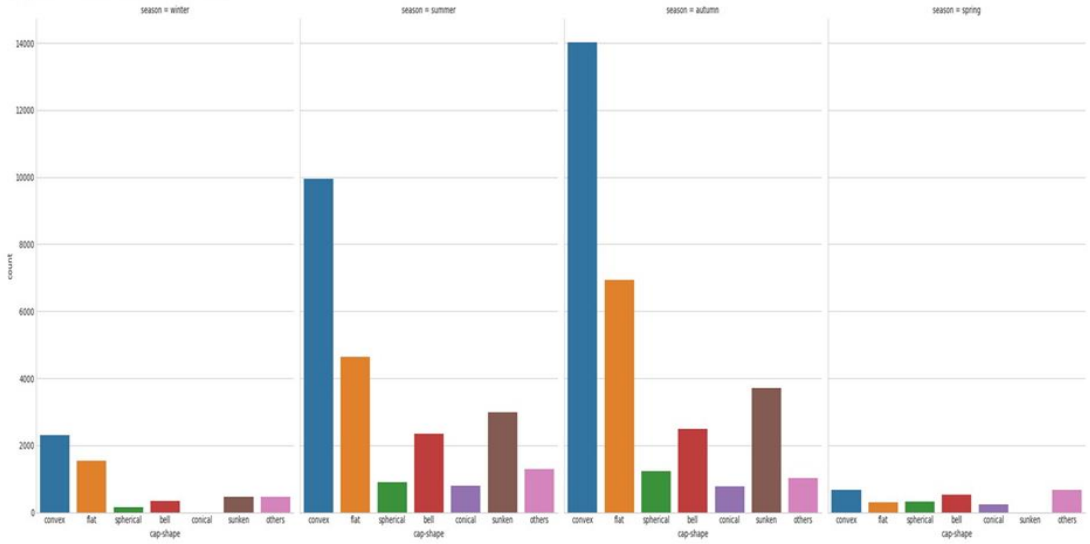


Fig. A4. Figure showing description and graph with listed p-values of association of cap-shape with season

```
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>
```

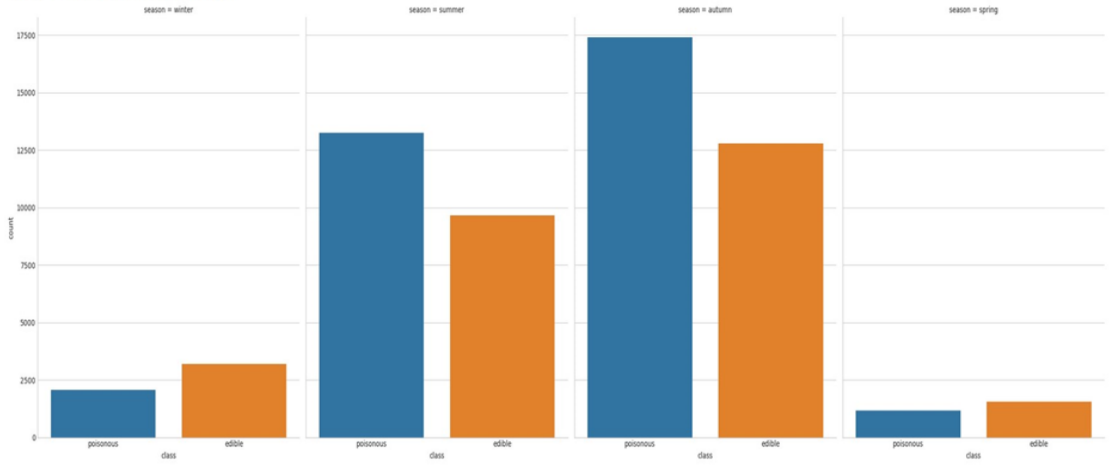


Fig. A5. Figure showing description and graph with listed p-values of association of class with season

```
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>
```

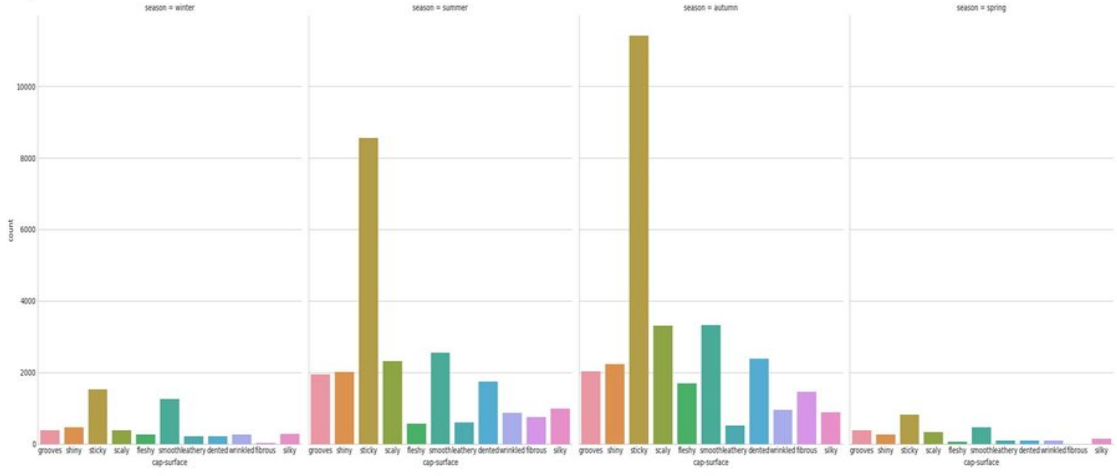


Fig.A6. Figure showing description and graph with listed p-values of association of cap-surface with season

```
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>
```

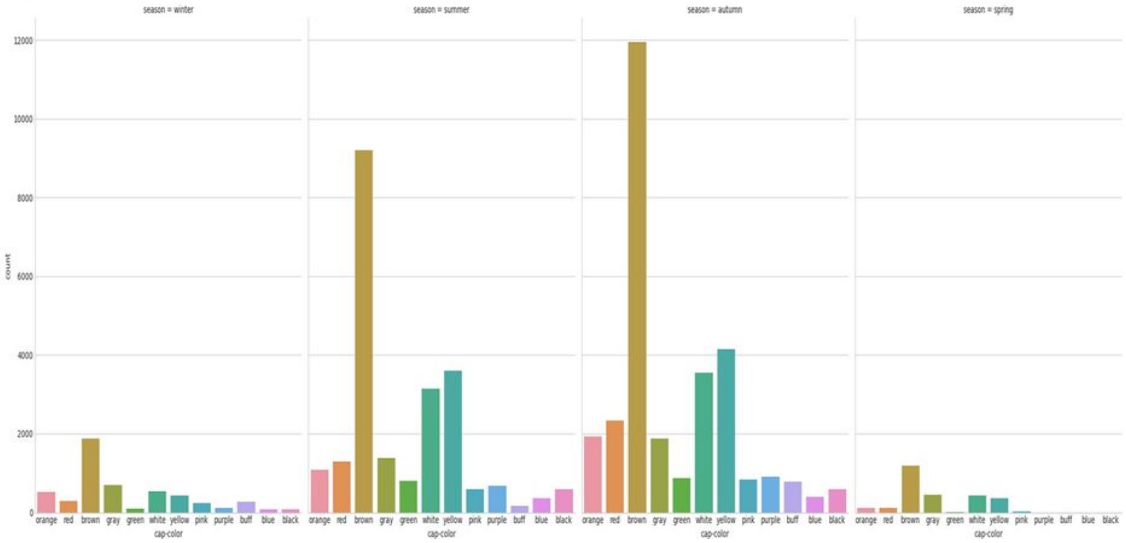


Fig.A7. Figure showing description and graph with listed p-values of association of cap-color with season

```
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:  1.2855626528436104e-36
<Figure size 300x500 with 0 Axes>
```

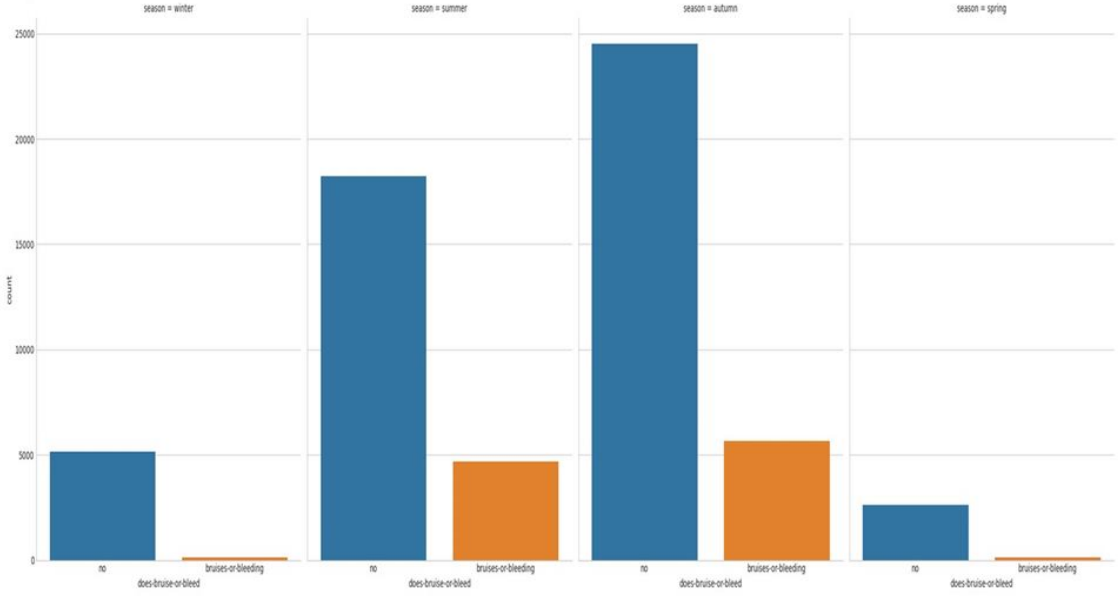


Fig.A8. Figure showing description and graph with listed p-values of association of does-bruise-or-bleed with season

```
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:  0.002636280376776693
<Figure size 300x500 with 0 Axes>
```

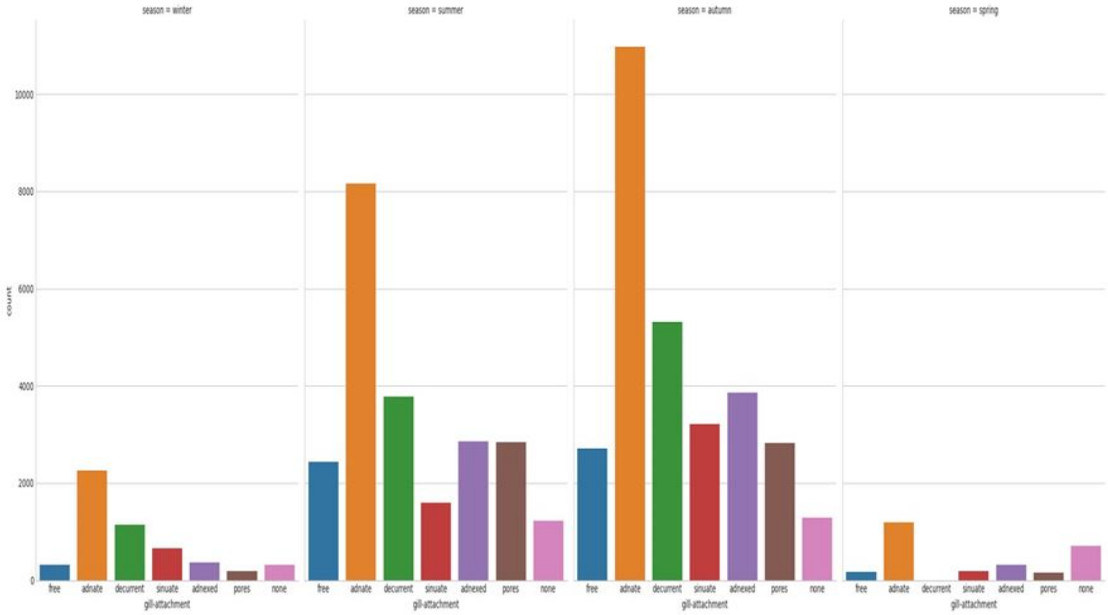


Fig.A9. Figure showing description and graph with listed p-values of association of gill-attachment with season

```
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>
```

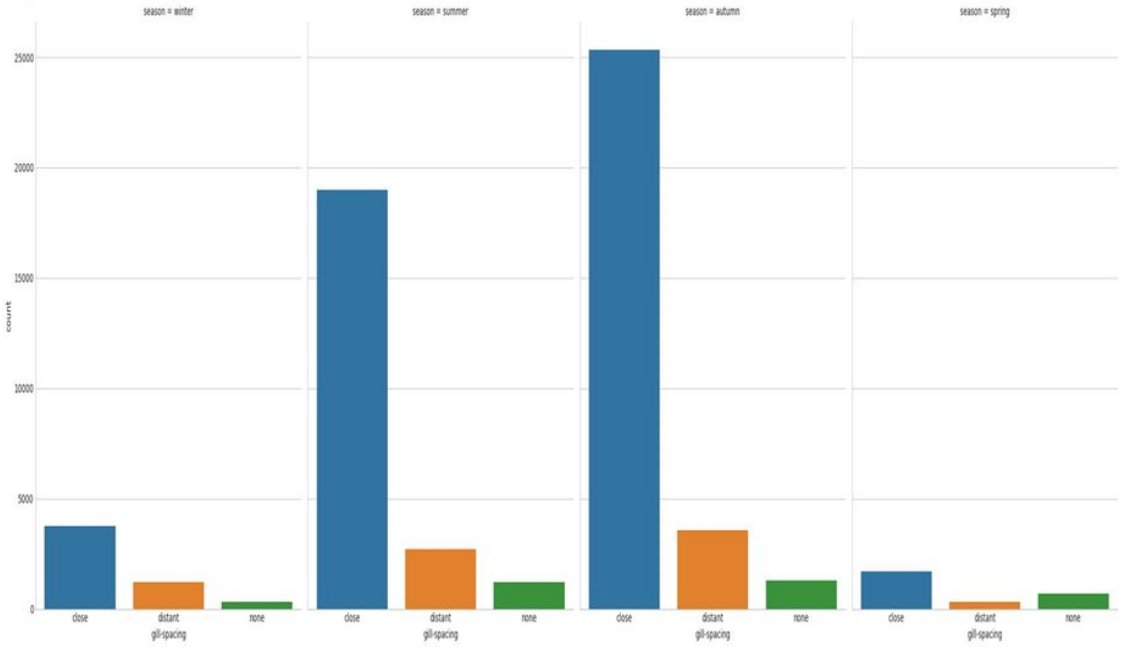


Fig.A10. Figure showing description and graph with listed p-values of association of gill-spacing with season

```
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>
```

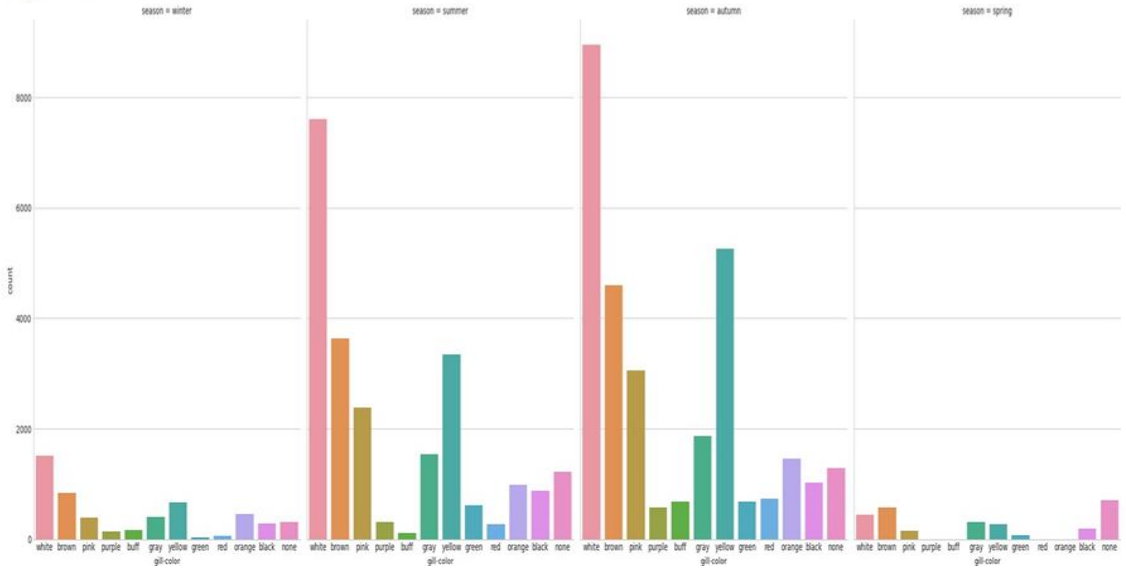


Fig.A11. Figure showing description and graph with listed p-values of association of gill-color with season

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

```
Table Original
stem-color  black  blue  brown  buff  gray  green  none  orange  pink  purple  \
season
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

stem-color  red  white  yellow
season
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>
```

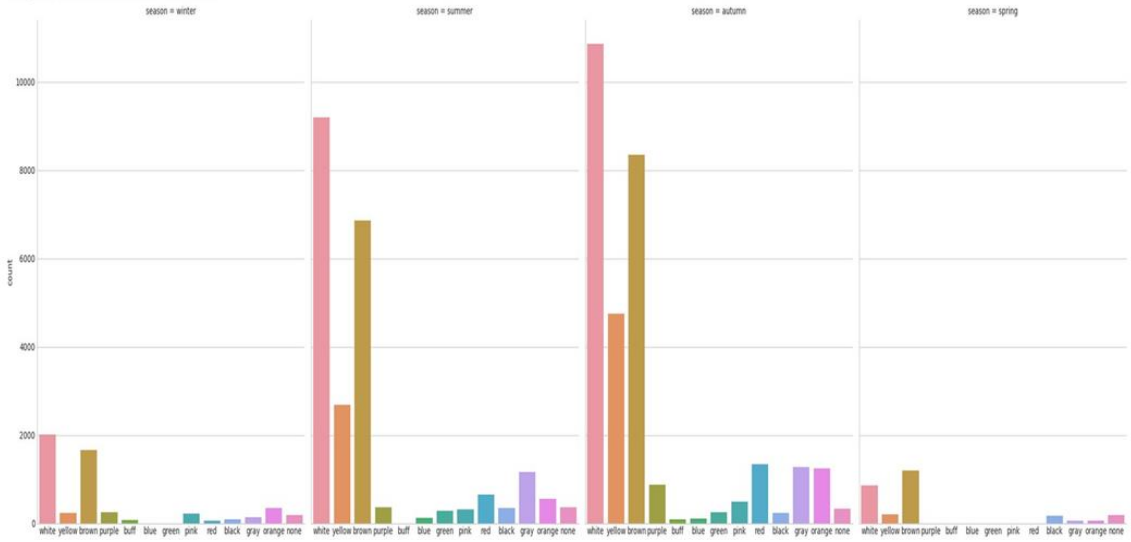


Fig.A12. Figure showing description and graph with listed p-values of association of stem-color with season

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

```
Table Original
has-ring   none  ring
season
autumn    23184  6993
spring     2048   679
summer    16812  6086
winter     3846  1421
P-value is:  9.776622781308942e-22
<Figure size 300x500 with 0 Axes>
```

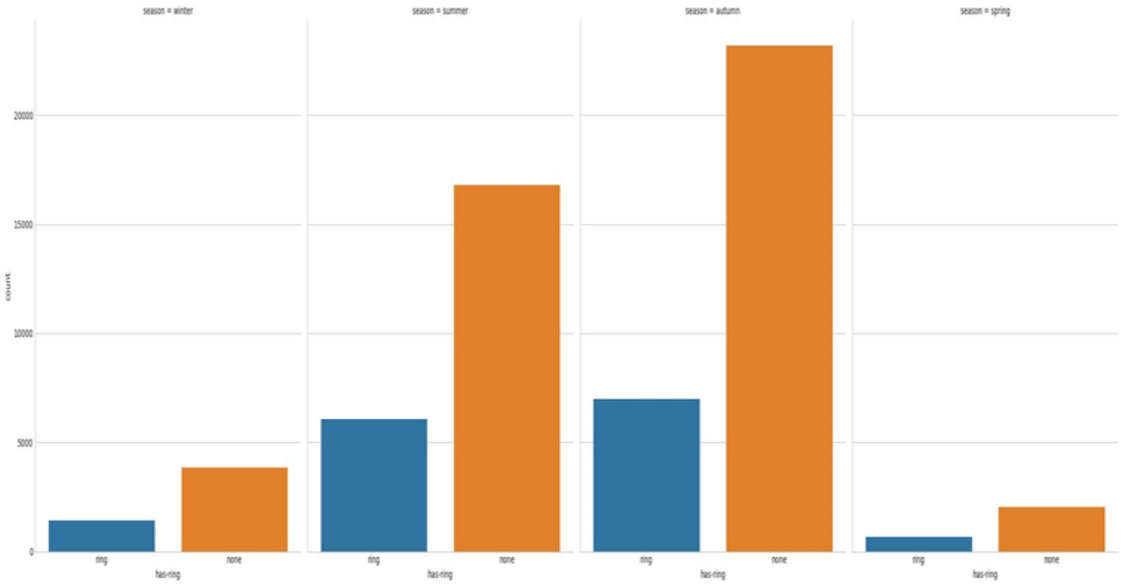


Fig.A13. Figure showing description and graph with listed p-values of association of has-ring with season

```
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>
```

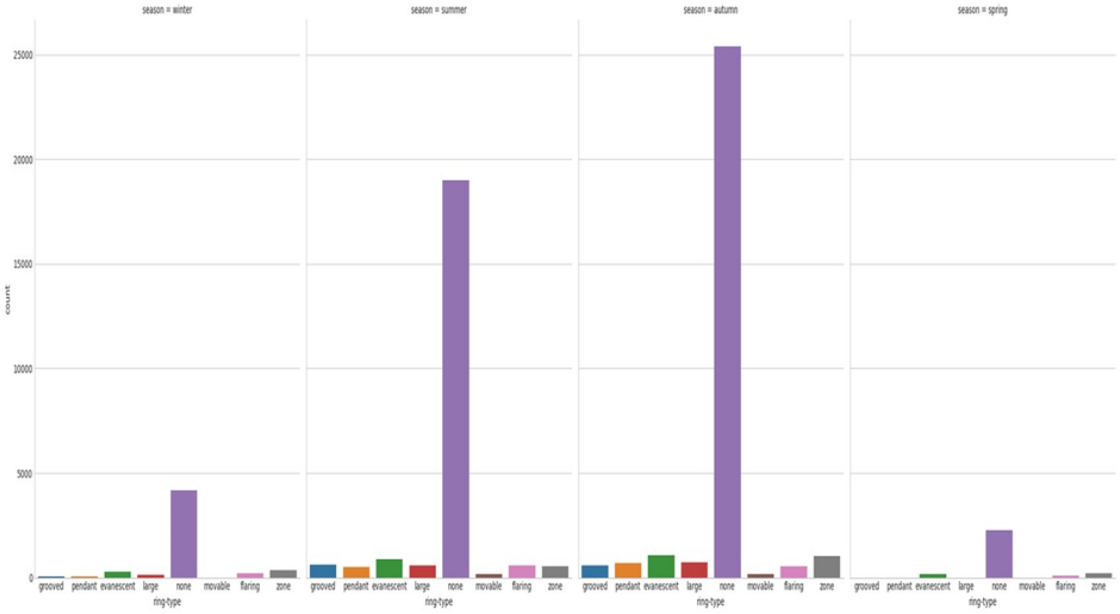


Fig.A14. Figure showing description and graph with listed p-values of association of ring-type with season

```
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>
```

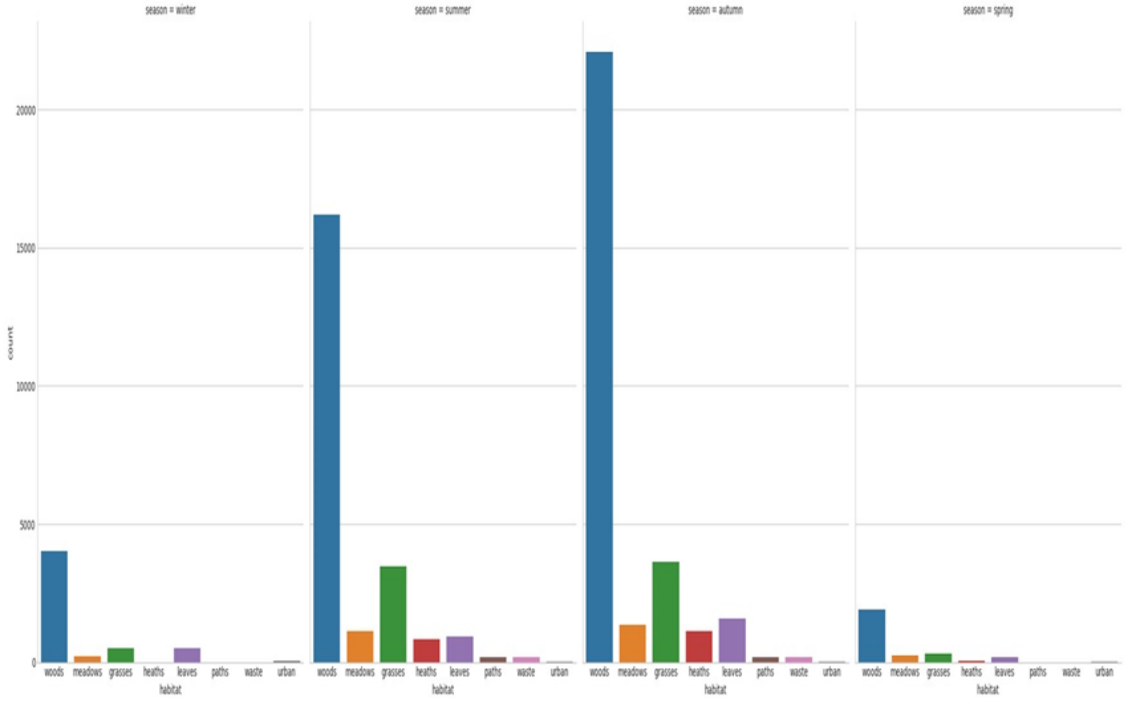


Fig.A15. Figure showing description and graph with listed p-values of association of habitat with season

```
#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_)
```

Fig.A16. Logistic Regression- Random Search for Hyperparameters

```
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}
```

Fig.A17. RandomizedSearchCV

```
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}
```

Fig.A18. RandomizedSearchCV

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