Mushroom Classification & Semi-supervised Learning

Group 2 Members:

Binaya Kumar Chaudhary
Hemant Raj Singh
Rohan Aryan-C0912902
Sani Asnain-C0906772
Shreya Baral (C0913115)

Table of Content

"In 2022, sales of fresh and processed mushrooms amounted to approximately 694.47 million Canadian dollars, an increase from around 653.51 million the previous year."

Business Case Evaluation

Why classify mushrooms can be helpful?

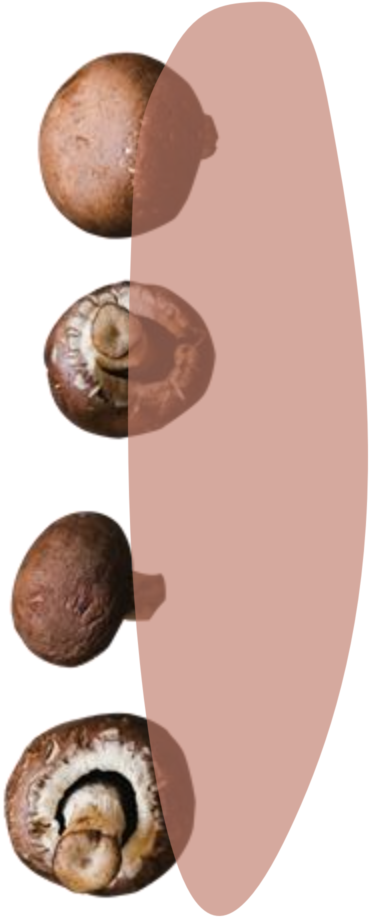
- Risks of consuming Toxic Mushrooms
- The need for accurate and Efficient Testing
- Improved Food Safety for Consumers
- Reduced Healthcare costs from Mushroom Poisoning

Data Identification

A mushroom containing 173 species (353 mushrooms per species) .

Obtained from the UCI dataset repository.

Can be accessed at the following link:
https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset.



Dataset Features

Class information:

1. **class**: poisonous=p, edible=e (binary)

Variable Information: (n: nominal, m: metrical; nominal values as sets of values)

1. **cap-diameter (m):** float number in cm
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
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

Dataset Features

1. **stem-height (m):** float number in cm
2. **stem-width (m)**: float number in mm
3. **stem-root (n)**: bulbous=b, swollen=s, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r
4. **stem-surface (n)**: see cap-surface + none=f
5. **stem-color (n)**: see cap-color + none=f
6. **veil-type (n)**: partial=p, universal=u
7. **veil-color (n)**: see cap-color + none=f
8. **has-ring (n)**: ring=t, none=f
9. **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=?
10. **spore-print-color (n)**: see cap color
11. **habitat (n)**: grasses=g, leaves=l, meadows=m, paths=p, heaths=h, urban=u, waste=w, woods=d
12. **season (n)**: spring=s, summer=u, autumn=a, winter=w

Data Acquisition and Filtering

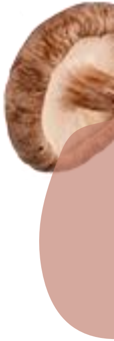
Data source: CSV format

File size: below 5MB

Library used: pandas, sklearn, seaborn, numpy

Function used to load dataset: read_csv()

Splitting Data



```
[ ] #dividing the features in numerical and categorical types
    Target='class'
    features=[col for col in df.columns if col not in [Target]]
    numeric_features=['cap-diameter','stem-height','stem-width']
    categorical_features=[col for col in features if col not in numeric_features]
    print('the target variable is:',Target)
    print('='*90)
    print('numeric_features',numeric_features)
    print('lenght:',len(numeric_features))
    print('='*90)
    print('categorical_features:',categorical_features)
    print('lenght:',len(categorical_features))
```

```
the target variable is: class
==========================================================================================
numeric_features ['cap-diameter', 'stem-height', 'stem-width']
lenght: 3
==========================================================================================
categorical_features: ['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-
lenght: 17
```

Data Validation & Cleansing

61,069 rows
Multivariate, Edibility prediction

used Pipelines

Column Transformer

Missing Data Handling

	missing_percent
veil-type	94.797688
spore-print-color	89.595376
veil-color	87.861272
stem-root	84.393064
stem-surface	62.427746
gill-spacing	41.040462
cap-surface	23.121387
gill-attachment	16.184971
ring-type	4.046243
class	0.000000
stem-color	0.000000
habitat	0.000000
has-ring	0.000000
stem-width	0.000000
cap-diameter	0.000000
stem-height	0.000000
gill-color	0.000000
does-bruise-or-bleed	0.000000
cap-color	0.000000
cap-shape	0.000000
season	0.000000

We encountered large percentage of missing values in the data set, along with some special characters like '?, nan'

We took up imputation for handling the missing data

```
[ ] #replacing special characters with NA
    df.replace(['?','nan'],pd.NA,inplace=True)
```

```
[ ] #imputing nan values with mode
    for columns in df.columns:
        most_common_value=df[columns].mode()[0] #to use the first most occuring value, as there can be a tie between value occurance
        df[columns].fillna(most_common_value,inplace=True)
```

```
[ ] #checking if the fillna worked.
    df.isnull().sum()
```

Data Normalization

To pre-process the data for model building we took up the approach of using pipelines.

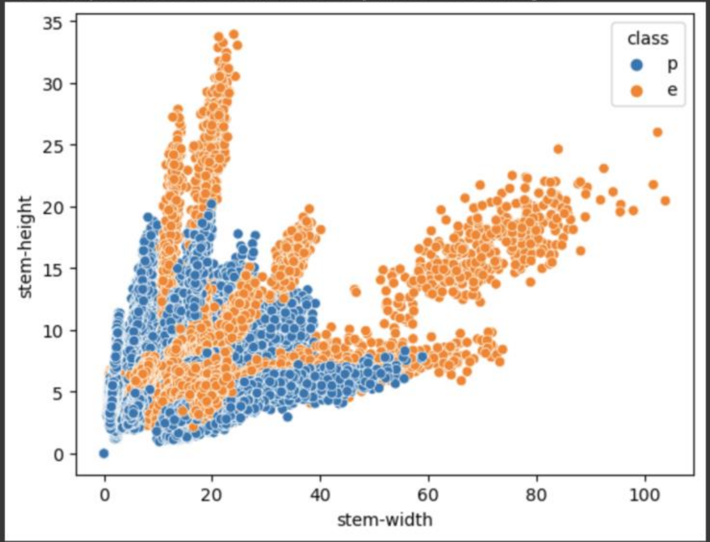
We created separate pipeline for categorical and numerical features, and integrated those pipelines using a column transformer

```
#converting the colums to lists:
numerical_list=list(numeric_features)
cateorical_list=list(categorical_features)

#pipeline for categorical features
pipeline_categorical = Pipeline([
  ('onehot', OneHotEncoder(drop='first')),
])
#pipeline for numerical features
pipeline_numerical = Pipeline([
  ('scaler', StandardScaler()),
])

#combined pipeline using column transformer
pipeline_full = ColumnTransformer([
    ("numerical", pipeline_numerical, numerical_list),
    ("categorical", pipeline_categorical, cateorical_list),
])
```

Exploratory Data Analysis

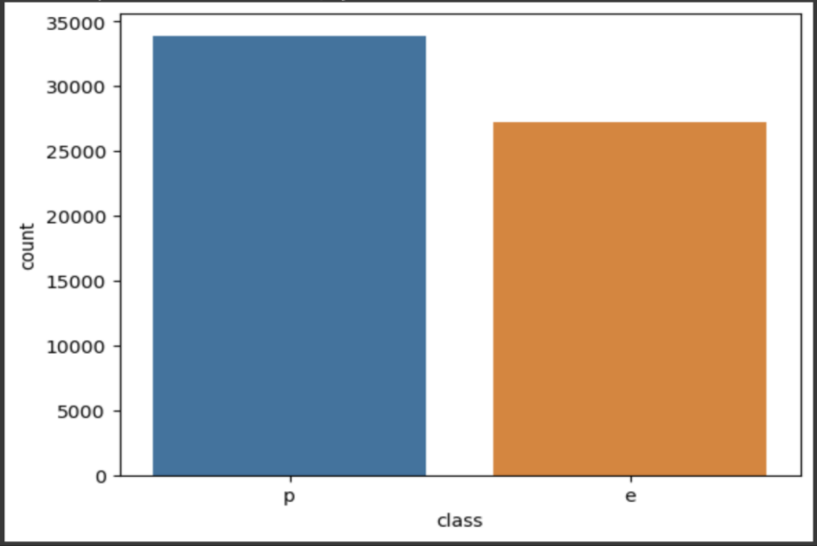
```
sns.scatterplot(data=df,y='stem-height',x='stem-width',hue='class')
```

```
<AxesSubplot:xlabel='stem-width', ylabel='stem-height'>
```



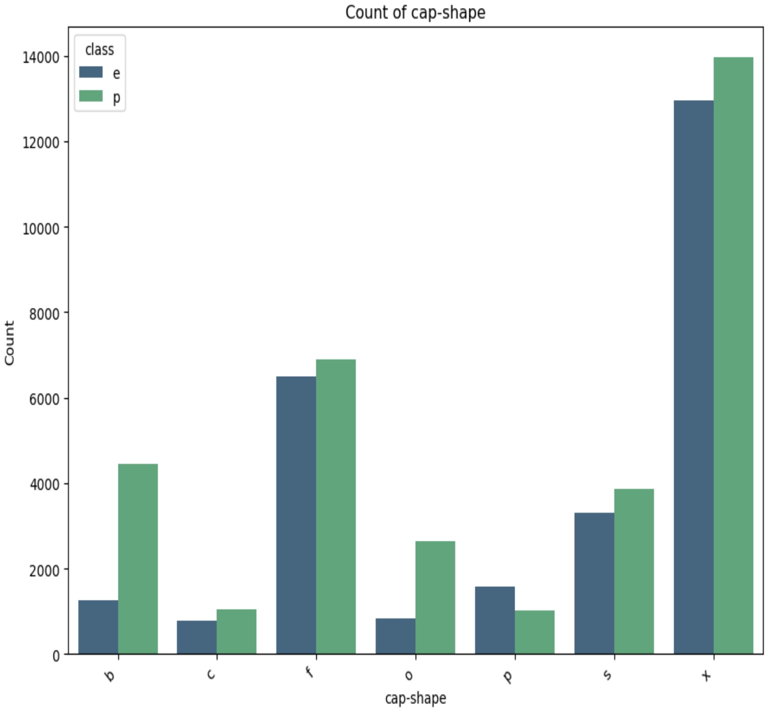

```
#checking target label bias
sns.countplot(x='class', data=df)
```

```
<AxesSubplot:xlabel='class', ylabel='count'>
```

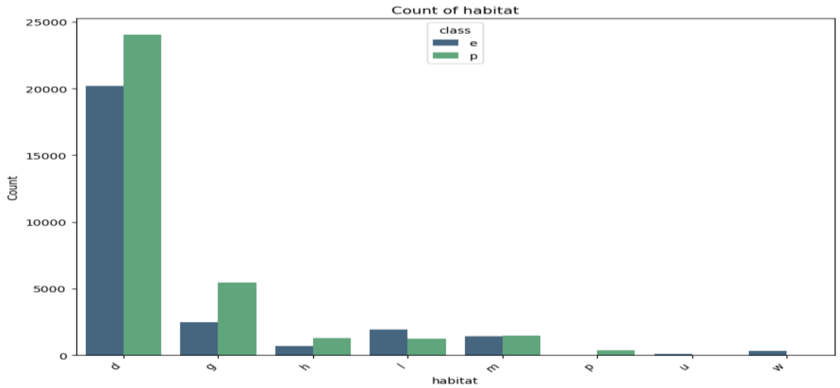


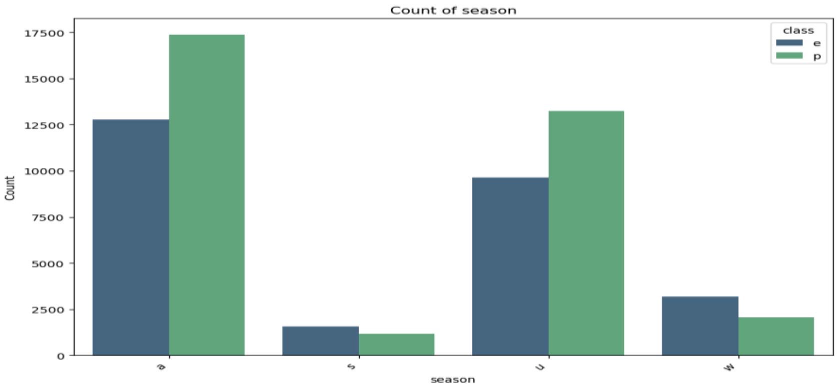
Exploratory Data Analysis

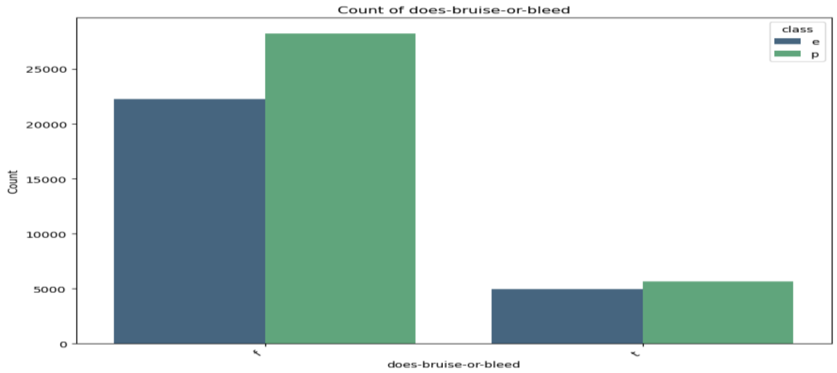
```python
def plot_count(x,ax):
    group = df.groupby([f'{x}','class'])['class'].count().reset_index(name='Count')
    sns.barplot(data=group,x=x,y='Count',hue='class',palette='viridis',ax=ax)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
    ax.set_title(f'Count of {x}')
cols = df.columns.tolist()
fig, axes = plt.subplots(10, 2, figsize=(18, 6 * 11))

for index,column in enumerate(cols[1:]):
    row = index // 2
    col = index % 2
    ax = axes[row,col]
    plot_count(column, ax)

plt.tight_layout()
plt.show()
```

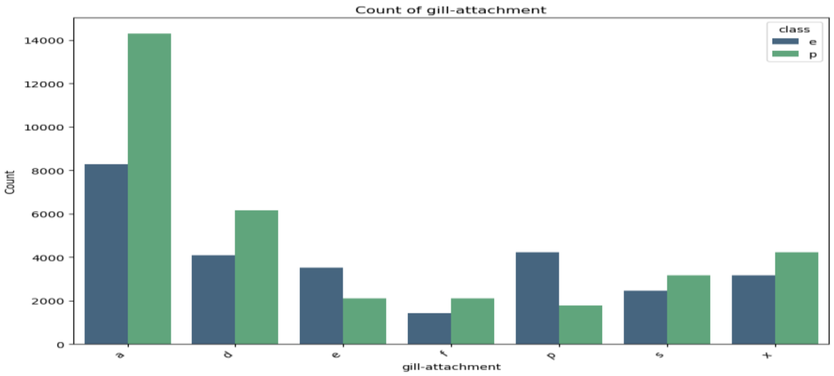


Exploratory Data Analysis

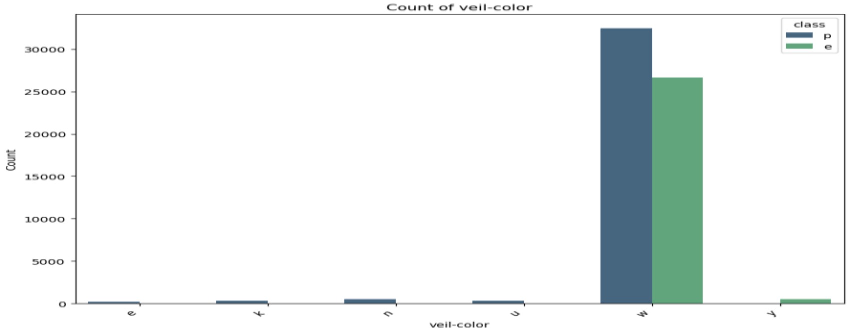


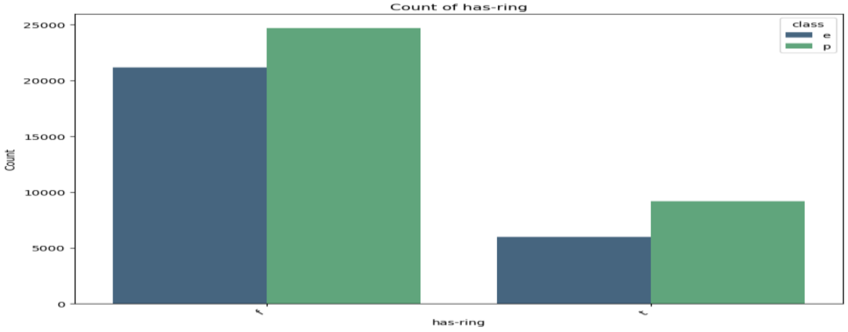


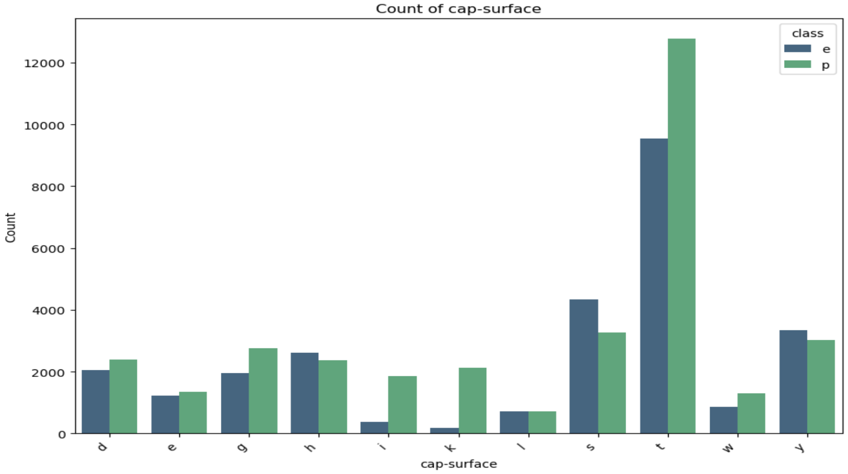


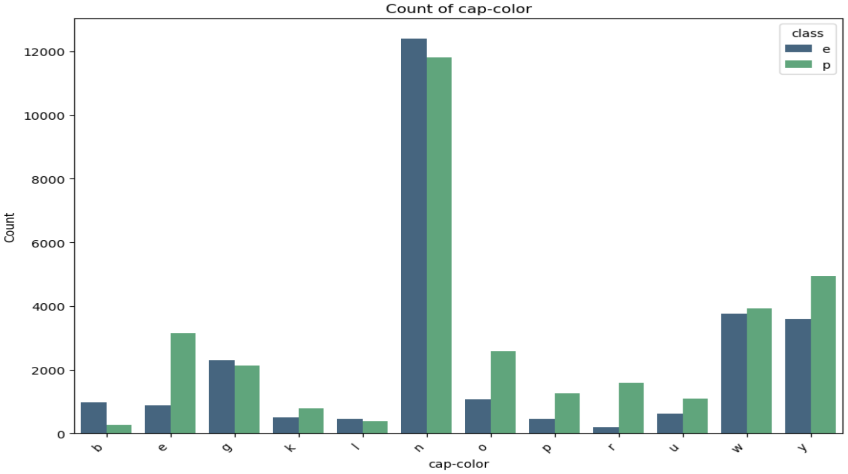


Exploratory Data Analysis









Semi-Supervised Learning

Semi-supervised approach, where characteristics of both supervised and unsupervised learning approaches are inhibited. This is a different approach to train the model, in this case a classifier. First with a smaller portion of labeled data and then a larger portion of unlabelled data. This approach pre-trains the model to understand the inherent data properly.

In our project, We used the last 20,000 records of our primary data set to fit into K-Means Clustering. For this we divided the data into labelled and unlabelled, trained the model on a small set and made predictions for a larger set, we then replaced these 20,000 records with 'Pseudo-labels' and used the now integrated data set for our other models.

Goal of using this approach is to uncover intricate relationship between the features with the target feature and pre-train the model for it, to have better results with the larger original dataset.

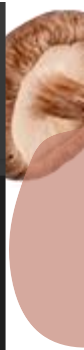
K-Means Clustering

- 20,000 records of primary data set to fit into K-Means Clustering
- We pre-processed the data using pipelines
- We then split the data into labelled and unlabelled segments

For semi-supervised approach, we made sure to have a smaller training set and have a larger testing set to have more predicted labels.

```
[ ] #using the last 20k records
    df1=df.loc[41069:61069]

[ ] #shape of the segmented data
    df1.shape

    (20000, 21)

    #x wil have labelled data which will be used to train the model
    #y will have unlabelled data which will be used to test the model
    #in semi supervised approach, training data is in smaller quantity compared to testing data
    X=df1[features]
    y=df1[Target]
    print('the feature set shape is:',X.shape)
    print('the Target set shape is:',y.shape)

    the feature set shape is: (20000, 20)
    the Target set shape is: (20000,)
```

K-Means Clustering



We did not require to use any additional technique to determine the number of clusters as we were dealing with a only 2 classes.

Shape of the data set after the split and preprocessing is shown in this step here.

As we had high number of categorical variables. We ended up generating around 87 columns after pre-processing

Creating Labelled and Unlabelled Data

Here: Xk_train and yk_train are labelled data and Xk_test is unlabelled Data

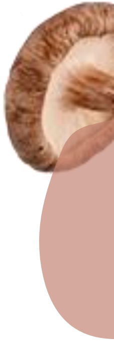
```
[ ] #now we split the feature and the target set in 20:80 ratio using train test split
    #we dont need a y test to compare the pred vs actual in this case, as the whole purpose is to just train the model
    #and gather pseudo labels to be merged into the main dataset.
    from sklearn.model_selection import train_test_split
    Xk_train, Xk_test, yk_train,_ = train_test_split(X, y, test_size=0.8, stratify=y) #stratified sampling based on the target
    print(f"Xk_train.shape: {Xk_train.shape}")
    print(f"Xk_test.shape: {Xk_test.shape}")
    print(f"yk_train.shape: {yk_train.shape}")
```

```
Xk_train.shape: (4000, 20)
Xk_test.shape: (16000, 20)
yk_train.shape: (4000,)
```

```
[ ] #pre-processing using pipelines.
    pipeline_full.fit(Xk_train)
    Xk_train_transformed=pipeline_full.fit_transform(Xk_train)
    Xk_test_transformed=pipeline_full.fit_transform(Xk_test)
    print(Xk_train_transformed.shape)
    print(Xk_test_transformed.shape)
```

```
(4000, 87)
(16000, 87)
```

K-Means Clustering



```
[ ] #now we train a kmeans clustering model on the training data and predict the testing data withit
    #we dont need to use any other method to determine the number of clusters to be formed since we already know the classes
    from sklearn.cluster import KMeans
    kmeans=KMeans(n_clusters=2,random_state=42)
    kmeans.fit(Xk_train_transformed,yk_train)

    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    ▾ KMeans
    KMeans(n_clusters=2, random_state=42)
```

```
[ ] pred_labels=kmeans.fit_predict(Xk_test_transformed)
```

```
[ ] print(pred_labels)

    [1 0 0 ... 1 0 1]
```

```
#assigning actual class names.
mapping={0:'e',1:'p'}
pred_labels=[mapping[num] for num in pred_labels]
```

```
[ ] print(pred_labels)

    ['p', 'e', 'e', 'e', 'p', 'p', 'e', 'p', 'p', 'e', 'p', 'p', 'e', 'p', 'p', 'p', 'e', 'p', 'e', 'p', 'p', 'e', 'e', 'p', 'p', 'p', 'p', 'p', 'e', 'e', 'p', 'e', 'p', 'p', 'p', 'p', 'e', 'p', 'p', 'p', 'e', 'p', 'p', '
```

Data Aggregation and Representation

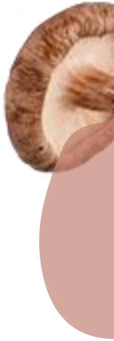
```
#merging the data with main dataset df1:
df_temp = pd.DataFrame({'class': pred_labels}, index=Xk_test.index)
df_20=pd.merge(Xk_test, df_temp, left_index=True, right_index=True, how='right')
#arranging the 'class' variable in the right order
df_20=df_20.set_index('class', append=True).reset_index(level=-1)

#merging Xk_train and yk_train:
merged_df=pd.concat([Xk_train, yk_train], axis=1)
#arranging class under the right column
merged_df=merged_df.set_index('class', append=True).reset_index(level=-1)
#merging df_20 amd merged_df
final_df=pd.concat([df_20,merged_df])
```

```
[ ] #removing the last 20k records with actual lables and replacing it with pseudo lables
df=df.iloc[:-20000]
```

```
[ ] #merging the pseudo lable dataset to main dataset
df=pd.concat([df,final_df])
```

We then predicted the labels, we got an array of 0's and 1's

We mapped the data accordingly and integrated it into the main dataset



Model Building

KNearestNeighbours

Random Forest Classifier

Decision Tree Classifier

Logistic Regression

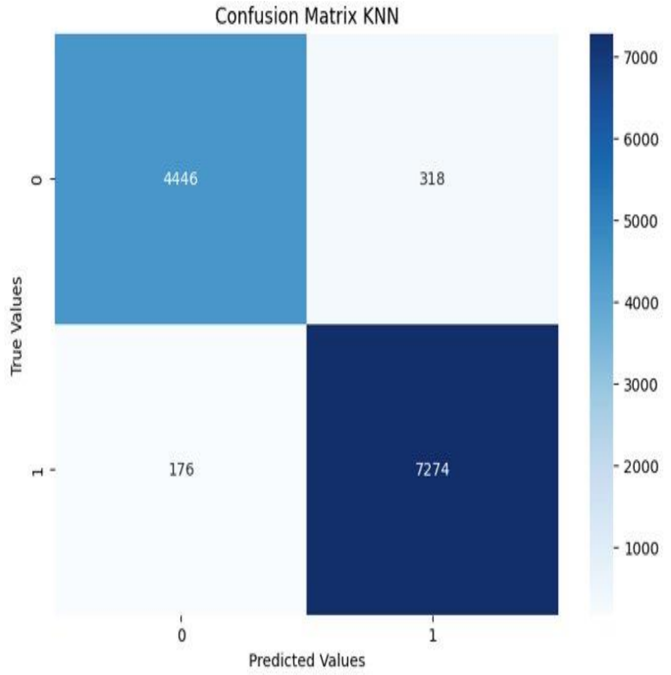
KNearestNeighboursM

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import cross_val_predict
    from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score
```

```
[ ] knn_classifier=KNeighborsClassifier()
```

```
[ ] knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    y_probas_trees = cross_val_predict(knn_classifier,X_train,y_train, cv=4, method="predict_proba")
    y_tree_scores = y_probas_trees[:, 1] # score = proba of positive class
    roc_auc_trees = roc_auc_score(y_train,y_tree_scores)
    test_accuracy = accuracy_score(y_test, y_pred)
    print(f"Test Set Accuracy for model {knn_classifier} is {test_accuracy:.2f}")
```

```
Test Set Accuracy for model KNeighborsClassifier() is 0.96
```

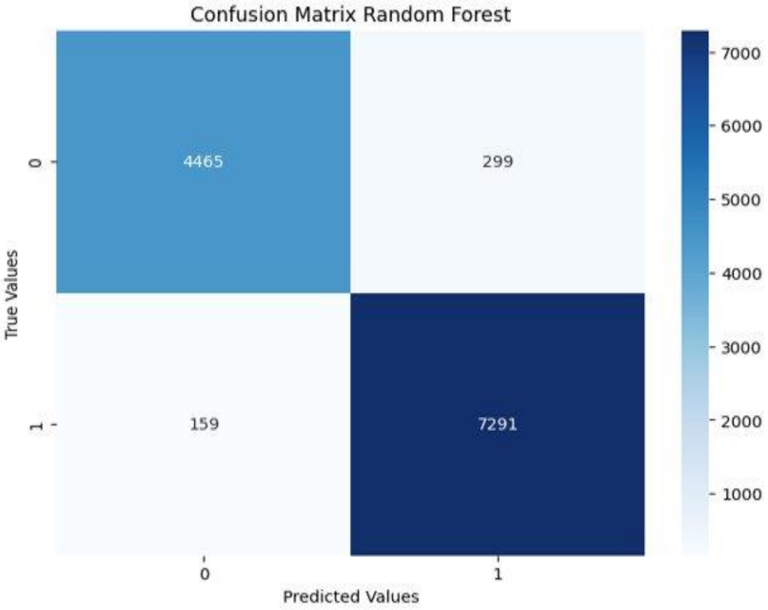


Random Forest

```
[ ] from sklearn.ensemble import RandomForestClassifier
```

```
[ ] rf_classifier = RandomForestClassifier(random_state = 42)
    rf_classifier.fit(X_train, y_train)

    y_pred = rf_classifier.predict(X_test)
    y_probas_trees = cross_val_predict(rf_classifier,X_train,y_train, cv=4, method="predict_proba")
    y_tree_scores = y_probas_trees[:, 1] # score = proba of positive class
    roc_auc_trees = roc_auc_score(y_train,y_tree_scores)
    test_accuracy = accuracy_score(y_test, y_pred)
    print(f"Test Set Accuracy for model {rf_classifier} is {test_accuracy:.2f}")
```

```
Test Set Accuracy for model RandomForestClassifier(random_state=42) is 0.96
```

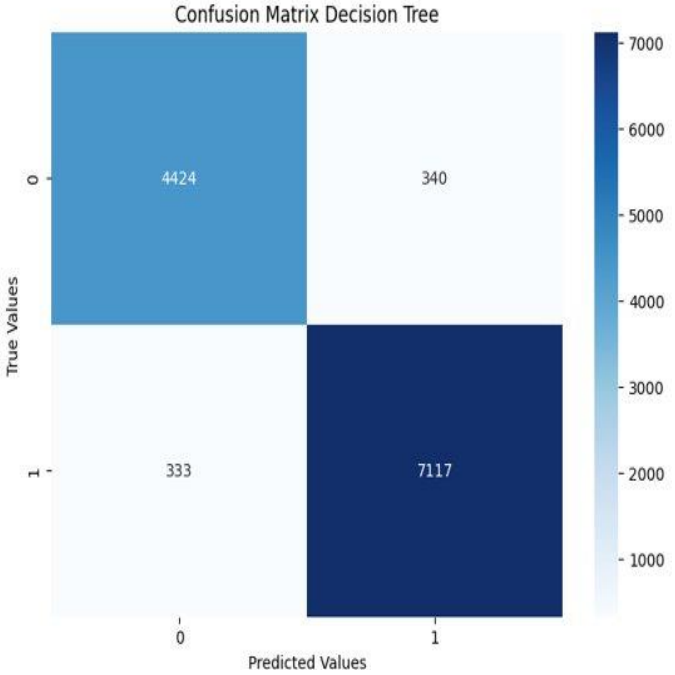


Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_probas_trees = cross_val_predict(clf,X_train,y_train, cv=4, method="predict_proba")
y_tree_scores = y_probas_trees[:, 1] # score = proba of positive class
roc_auc_trees = roc_auc_score(y_train,y_tree_scores)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Set Accuracy for model {clf} is {test_accuracy:.2f}")
```

```
Test Set Accuracy for model DecisionTreeClassifier(random_state=42) is 0.94
```

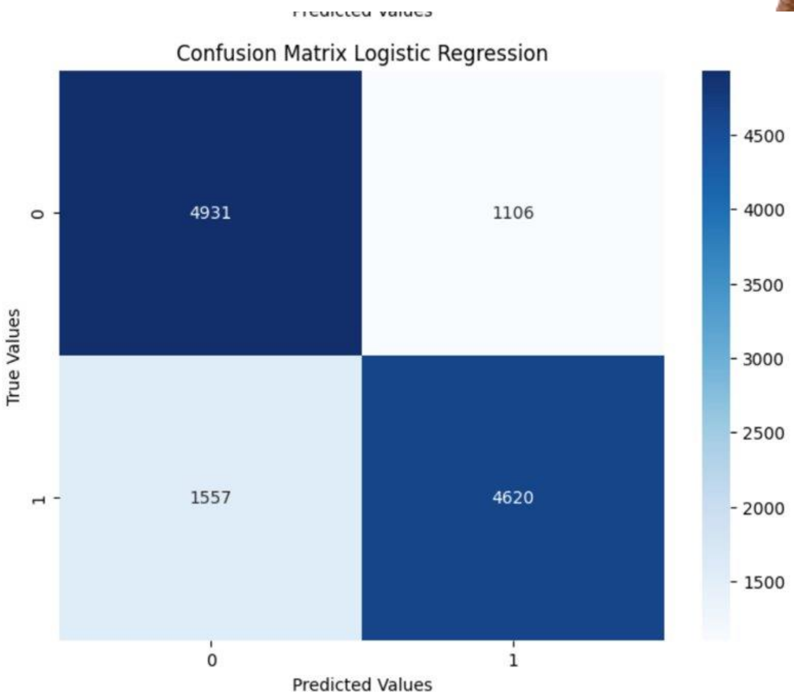


Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

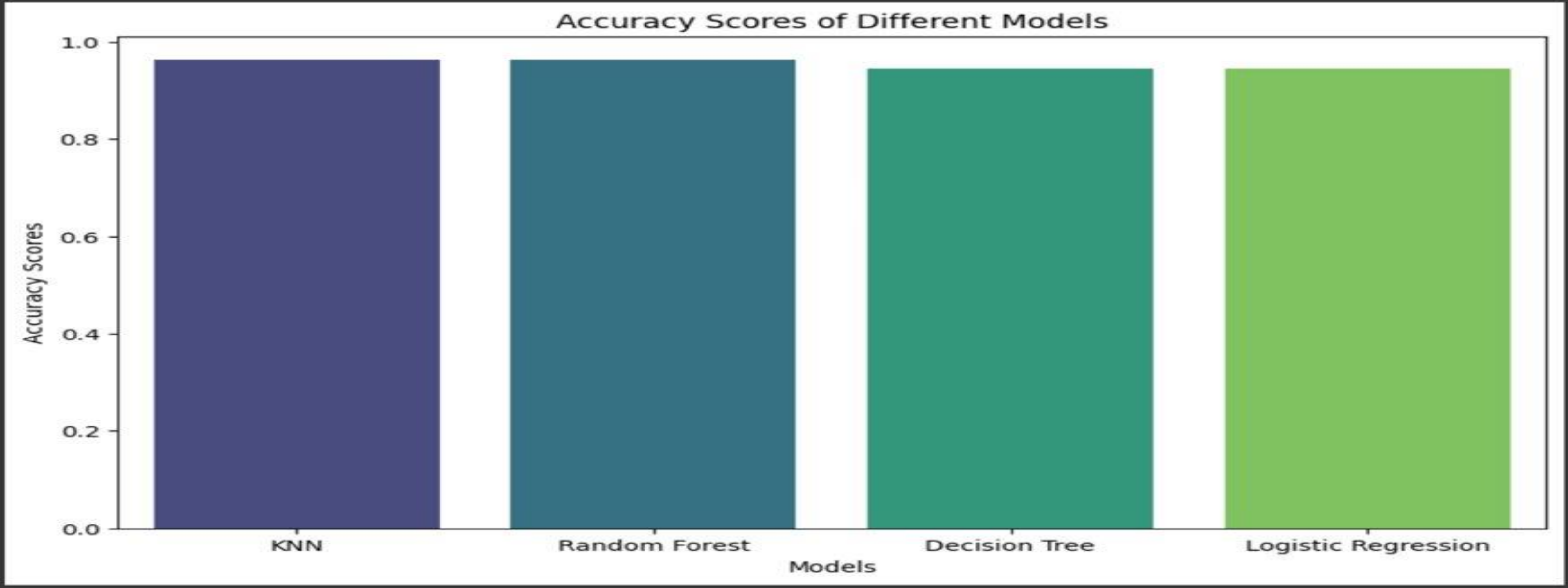
```
[ ] logreg_model = LogisticRegression(max_iter=10000)
    logreg_model.fit(X_train, y_train)
    y_pred = logreg_model.predict(X_test)

    y_probas_trees = cross_val_predict(logreg_model,X_train,y_train, cv=4, method="predict_proba")
    y_tree_scores = y_probas_trees[:, 1] # score = proba of positive class
    roc_auc_trees = roc_auc_score(y_train,y_tree_scores)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Test Set Accuracy for model {logreg_model} is {test_accuracy:.2f}")
```

```
Test Set Accuracy for model LogisticRegression(max_iter=10000) is 0.94
```



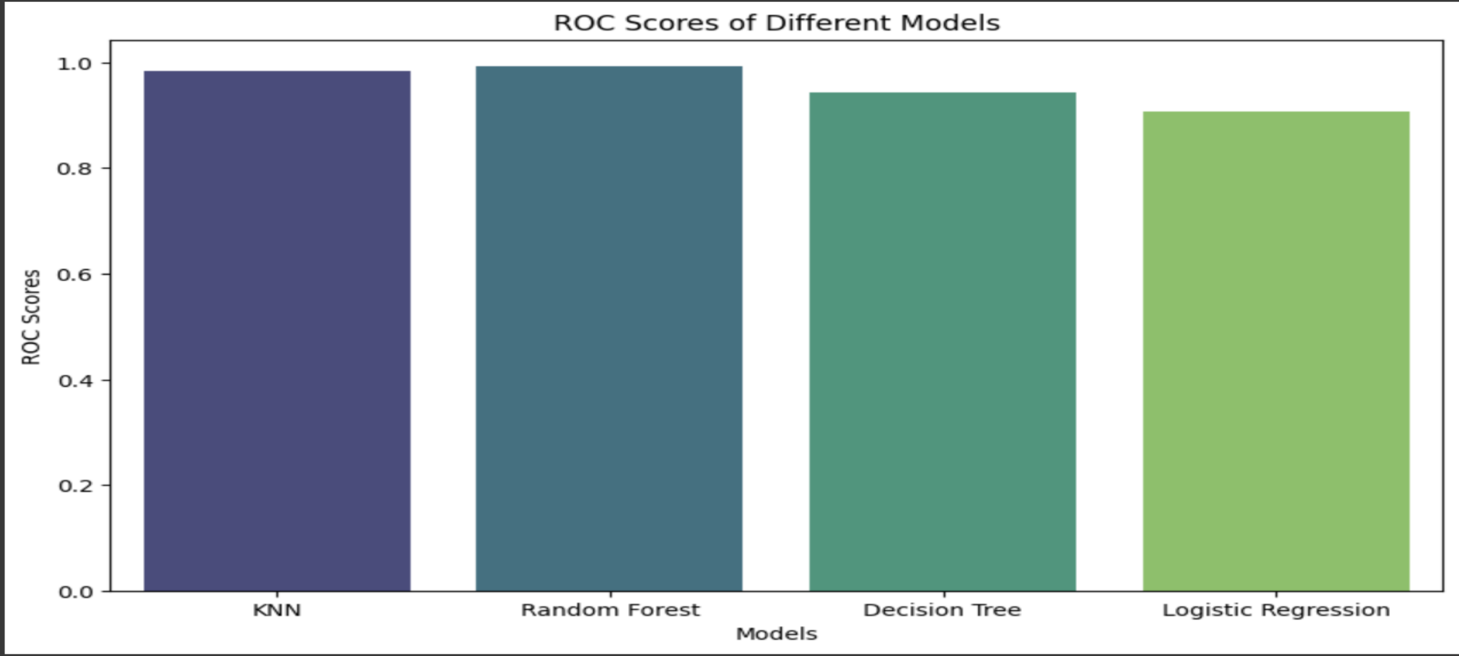
Accuracy Scores of Different Models

```
# Plottint Accuracy of each model
model_vs_metric(model_list, accuracy_list, 'Accuracy Scores')
```



Roc Scores of Different Models

```python
# Plottint ROC of each model
model_vs_metric(model_list, roc_curve_list, 'ROC Scores')
```



Utilization of Analysis

Data-Driven Decision-Making:
instead of randomly guessing a mushroom to be edible or not, you can take a data backed decision. The model generates valuable insights into the factors influencing the classification of mushrooms. These insights can be leveraged for data-driven decision-making, such as optimizing cultivation practices or adjusting product portfolios.

Market Expansion and Consumer Education:
A reliable classification model can facilitate market expansion by assuring consumers of the safety of the products. It also provides an opportunity for businesses to educate consumers about the importance of choosing safe mushrooms.

Market Differentiation:
A business that can guarantee the safety of its mushroom products through an advanced classification system may gain a competitive edge in the market. It can be positioned as a brand that prioritizes consumer well-being.

Crisis Management:
In the event of a contamination or poisoning incident, having a classification model can enable businesses to quickly identify affected batches, initiate recalls, and communicate with consumers, mitigating the impact on both public safety and the brand.

Thanks!

Any Questions?