MUSHROOM CLASSIFICATION USING MACHINE LEARNING

Jason Adika Tanuwijaya
Makhabat Zhyrgalbekova
Naima Dzhunushova
Devanshi Rhea Aucharaz

BDS 23 YEAR 2 SEM 4

SPJ GLOBAL
S P Jain School of Global Management
DUBAI • MUMBAI • SINGAPORE • SYDNEY

Project Scope and Objectives

Project Scope:
> Predict **mushroom edibility (edible vs poisonous)** using a synthetic dataset based on a UCI Dataset (dataset ID 848) — Secondary Mushroom Dataset.

Overall Objective:
> Support biological education and toxicology analysis by leveraging over 61,000 records of artificially generated mushroom data.

Project Goals:
> Build robust classification models.
> Evaluate and compare multiple machine learning approaches.
> Test the final model on an unseen test set to assess real-world generalizability.

Dataset Overview

[UCI DATASET](#) Inspired by the Mushroom Data Set of J. Schlimmer

Based on 173 species (353 mushrooms /-)

~61,000 rows used for initial modeling

class – whether the mushroom is edible (e) or poisonous (p)

20 independent variables (mix of categorical and numerical)

➤ **Categorical**

- cap-shape
- cap-surface
- cap-color
- gill-spacing
- veil-type
- habitat
- season
- gill-attachment
- gill-color
- stem-root
- stem-surface
- stem-color
- veil-color
- has-ring
- ring-type
- spore-print-color
- does-bruise-or-bleed

➤ **Numerical**

- cap-diameter
- stem-height
- stem-width

Preprocessing & Data Cleaning

Initial Analysis of the Training Data

Dataset Import:

- Loaded from UCImlrepo (ID: 848), including 61,000+ synthetic mushroom records split into features (X) and labels (y).
- Combined into a single dataframe for analysis.

```
df.head()
```

	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	stem-width	...	stem-surface	stem-color	veil-type	veil-color	has-ring	ring-type	spore-print-color	habitat	season	class
0	15.26	x	g	o	f	e	NaN	w	16.95	17.09	...	y	w	u	w	t	g	NaN	d	w	p
1	16.60	x	g	o	f	e	NaN	w	17.99	18.19	...	y	w	u	w	t	g	NaN	d	u	p
2	14.07	x	g	o	f	e	NaN	w	17.80	17.74	...	y	w	u	w	t	g	NaN	d	w	p
3	14.17	f	h	e	f	e	NaN	w	15.77	15.98	...	y	w	u	w	t	p	NaN	d	w	p
4	14.64	x	h	o	f	e	NaN	w	16.53	17.20	...	y	w	u	w	t	p	NaN	d	w	p

5 rows × 21 columns

Dataset Preview:

Shows the first 5 rows with 21 columns describing mushroom features.

```
df.info()
```

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

Dataset Info:

61,069 entries, 21 columns. Mix of numeric and categorical data, with some missing values.

Checking Null Values/Handling Missing Values

```
null_df = pd.DataFrame(df[df.columns[df.isna().sum() > 0]].isna().sum())
null_df.columns = ["Null counter"]

# Add percentage column
null_df["Missing Percentage (%)"] = (null_df["Null counter"] / len(df)) * 100

# Sort by number of nulls
null_df = null_df.sort_values(by="Null counter", ascending=False)
null_df
```

	Null counter	Missing Percentage (%)
veil-type	57892	94.797688
spore-print-color	54715	89.595376
veil-color	53656	87.861272
stem-root	51538	84.393064
stem-surface	38124	62.427746
gill-spacing	25063	41.040462
cap-surface	14120	23.121387
gill-attachment	9884	16.184971
ring-type	2471	4.046243

Checked for missing values and calculated their percentages.

Dropped columns with over 80% missing data (veil-type, spore-print-color, etc.).

```
[ ]  df.drop(columns=['veil-type', 'spore-print-color', 'veil-color', 'stem-root'],inplace = True)
```

Filled remaining missing values using mode (most frequent value), since most features are categorical.

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

```
df.isna().sum() # check if the dataset is clean
```

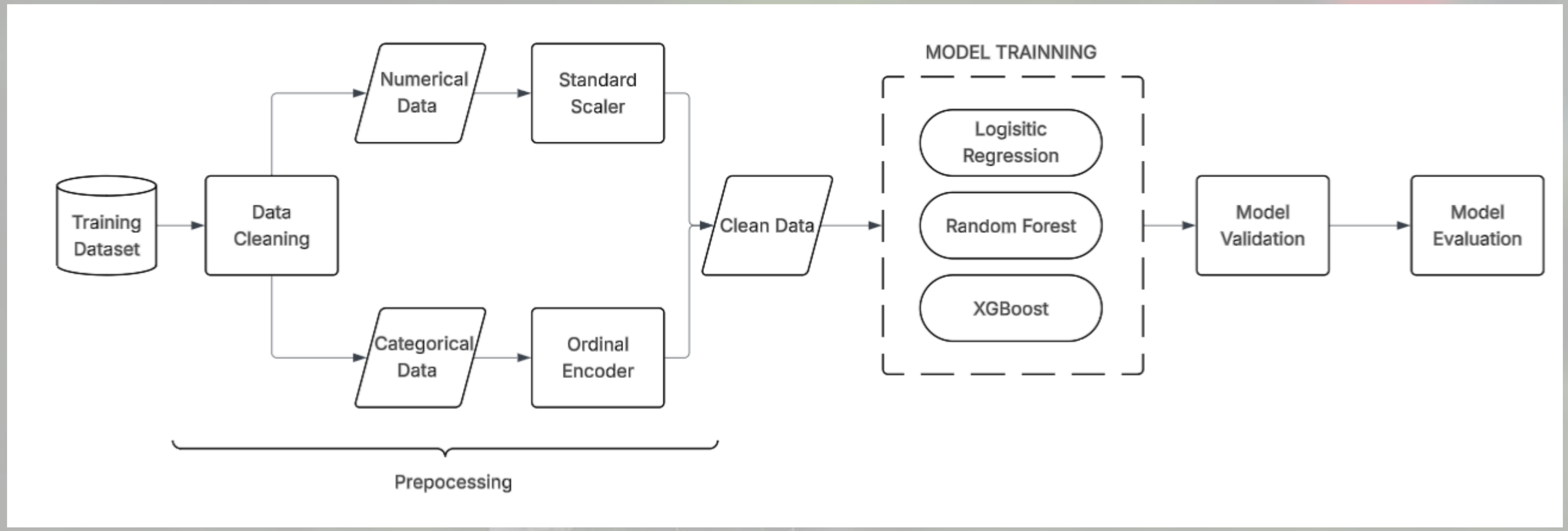
	0
cap-diameter	0
cap-shape	0
cap-surface	0
cap-color	0
does-bruise-or-bleed	0
gill-attachment	0
gill-spacing	0
gill-color	0
stem-height	0
stem-width	0
stem-surface	0
stem-color	0
has-ring	0
ring-type	0
habitat	0
season	0
class	0

Now that our dataset is clean and free of missing values, we're ready to move on to Round 1

Round 1

- Logistic Engineering
- Random Forest Classifier
- XGBoost
- Simple Cross-Validation

Round 1


Training Dataset
Data Cleaning
Numerical Data
Standard Scaler
Categorical Data
Ordinal Encoder
Prepocessing
Clean Data
MODEL TRAINNING
Logisitic Regression
Random Forest
XGBoost
Model Validation
Model Evaluation

Preparation, before training

```
#ENDCODING
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define categorical and numerical columns
categorical_cols = [
    'cap-shape', 'cap-surface', 'cap-color', 'does-bruise-or-bleed',
    'gill-attachment', 'gill-spacing', 'gill-color',
    'stem-surface', 'stem-color', 'has-ring',
    'ring-type', 'habitat', 'season'
]

numerical_cols = [
    'cap-diameter', 'stem-height', 'stem-width'
]

X = df.drop(columns = ['class'])
y = df['class']
```

We split the dataset into categorical and numerical columns to prepare for encoding:

- **Categorical:** cap-shape, gill-color, habitat, etc.
- **Numerical:** cap-diameter, stem-height, stem-width
- Also, the **target variable** class is separated from the feature set for model training.

Logistic Regression

solver	penalty	cv_accuracy
sag	l2	0.651609
liblinear	l1	0.651586
newton-cg	l2	0.651586
lbfgs	none	0.651586
newton-cg	none	0.651586
saga	l1	0.651563
saga	none	0.651563
saga	l2	0.651539
sag	none	0.651516
newton-cholesky	none	0.651516
newton-cholesky	l2	0.651516
lbfgs	l2	0.651493
liblinear	l2	0.651399

Logistic Regression

- Built a pipeline with OrdinalEncoder for categorical features and StandardScaler for numerical ones.
- Used Logistic Regression with class_weight='balanced' to address class imbalance.
- Model trained on a 70/30 train-test split.

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.preprocessing import StandardScaler, OrdinalEncoder
    from sklearn.compose import ColumnTransformer
    from sklearn.pipeline import Pipeline
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

```
# Create the logistic regression pipeline

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OrdinalEncoder(), categorical_cols)  # Using OrdinalEncoder instead of LabelEncoder
    ])

logreg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(
        penalty='l2',           # Ridge regularization
        C=1.0,                  # Inverse of regularization strength
        solver='lbfgs',         # Good for multiclass and small datasets
        max_iter=1000,          # Increased for convergence
        random_state=42,
        class_weight='balanced' # Handles imbalanced classes
    ))
])

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model
logreg_pipeline.fit(X_train, y_train)
```

```
# Make predictions
y_pred = logreg_pipeline.predict(X_test)
y_pred_proba = logreg_pipeline.predict_proba(X_test)[:, 1]  # Probability estimates

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6515474046176519

Confusion Matrix:
[[5200 2908]
 [3476 6737]]

Classification Report:
              precision    recall  f1-score   support

           e       0.60      0.64      0.62      8108
           p       0.70      0.66      0.68     10213

    accuracy                           0.65     18321
   macro avg       0.65      0.65      0.65     18321
weighted avg       0.65      0.65      0.65     18321
```

Model Evaluation

- **Accuracy: ~65%**
- Confusion Matrix shows class distribution of predictions.
- Classification Report: Precision and recall are fairly balanced, with room for improvement in both classes

Random Forest

Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OrdinalEncoder(), categorical_cols)  # Using OrdinalEncoder instead of LabelEncoder
    ])

# Create the complete pipeline with preprocessing and model
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(
        n_estimators=100,
        max_depth=10,
        random_state=42,
        class_weight='balanced'  # Useful if classes are imbalanced
    ))
])

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
pipeline.fit(X_train, y_train)

pipeline
```

- Created a pipeline with Ordinal Encoding and Standard Scaling.
- Used Random Forest with n_estimators=100, max_depth=10, and class_weight='balanced'.
- Dataset was split 80/20 for training and testing.

```python
# Make predictions
y_pred = pipeline.predict(X_test)
```

```python
from sklearn.metrics import classification_report, accuracy_score

# Evaluate the model
print("Model Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Model Accuracy: 0.9807597838545931

Classification Report:
              precision    recall  f1-score   support

           e       0.96      1.00      0.98      5374
           p       1.00      0.97      0.98      6840

    accuracy                           0.98     12214
   macro avg       0.98      0.98      0.98     12214
weighted avg       0.98      0.98      0.98     12214
```

- **Accuracy: ~98%**
- Strong precision, recall, and F1-scores for both edible (e) and poisonous (p) classes.
- Model shows significant improvement over logistic regression.

XGBoost

XGBoost

```python
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

# Encode the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # y = df['class']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)

# XGBoost pipeline
xgb_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(n_estimators=100, max_depth=10, use_label_encoder=False, eval_metric='mlogloss'))
])

# Fit the model
xgb_pipeline.fit(X_train, y_train)
```

- Encoded the target variable using LabelEncoder.
- Built a pipeline with preprocessing and an XGBClassifier.
- 70/30 train-test split.

```
Accuracy: 0.9993995960919164

Classification Report:
              precision    recall  f1-score   support

           e       1.00      1.00      1.00      8108
           p       1.00      1.00      1.00     10213

    accuracy                           1.00     18321
   macro avg       1.00      1.00      1.00     18321
weighted avg       1.00      1.00      1.00     18321
```

- **~99.93% accuracy**
- Indicates excellent performance on the test set

Hyperparameter Tuning (GridSearchCV)

```
[ ] #Hyperparameter Tuning (GridSearchCV)
    from sklearn.model_selection import GridSearchCV

    param_grid = {
        'classifier__n_estimators': [100, 200],
        'classifier__max_depth': [10, 15, 20]
    }
    grid = GridSearchCV(pipeline, param_grid, cv=3)
    grid.fit(X_train, y_train)
    print("Best params:", grid.best_params_)
```

```
Best params: {'classifier__max_depth': 20, 'classifier__n_estimators': 200}
```

```
#Update Your XGBoost Pipeline with Best Params
xgb_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(n_estimators=200, max_depth=20, use_label_encoder=False, eval_metric='mlogloss'))
])
xgb_pipeline.fit(X_train, y_train)
```

- Tuned n_estimators and max_depth for XGBoost using GridSearchCV with 3-fold cross-validation.
- Best parameters: n_estimators = 200, max_depth = 20
- Retrained the model with the optimal settings.

✅ Final Model Accuracy: 0.9991812673980678

📋 Final Classification Report:

	precision	recall	f1-score	support
e	1.00	1.00	1.00	8108
p	1.00	1.00	1.00	10213
accuracy			1.00	18321
macro avg	1.00	1.00	1.00	18321
weighted avg	1.00	1.00	1.00	18321

- **~99.94% accuracy**
- Perfect precision, recall, and F1-score for both classes
- Final model is highly accurate and generalizes well.

Model Comparison

Model	accuracy
logistic regression	0.651609
random forest	0.999789
xgboost	0.999454

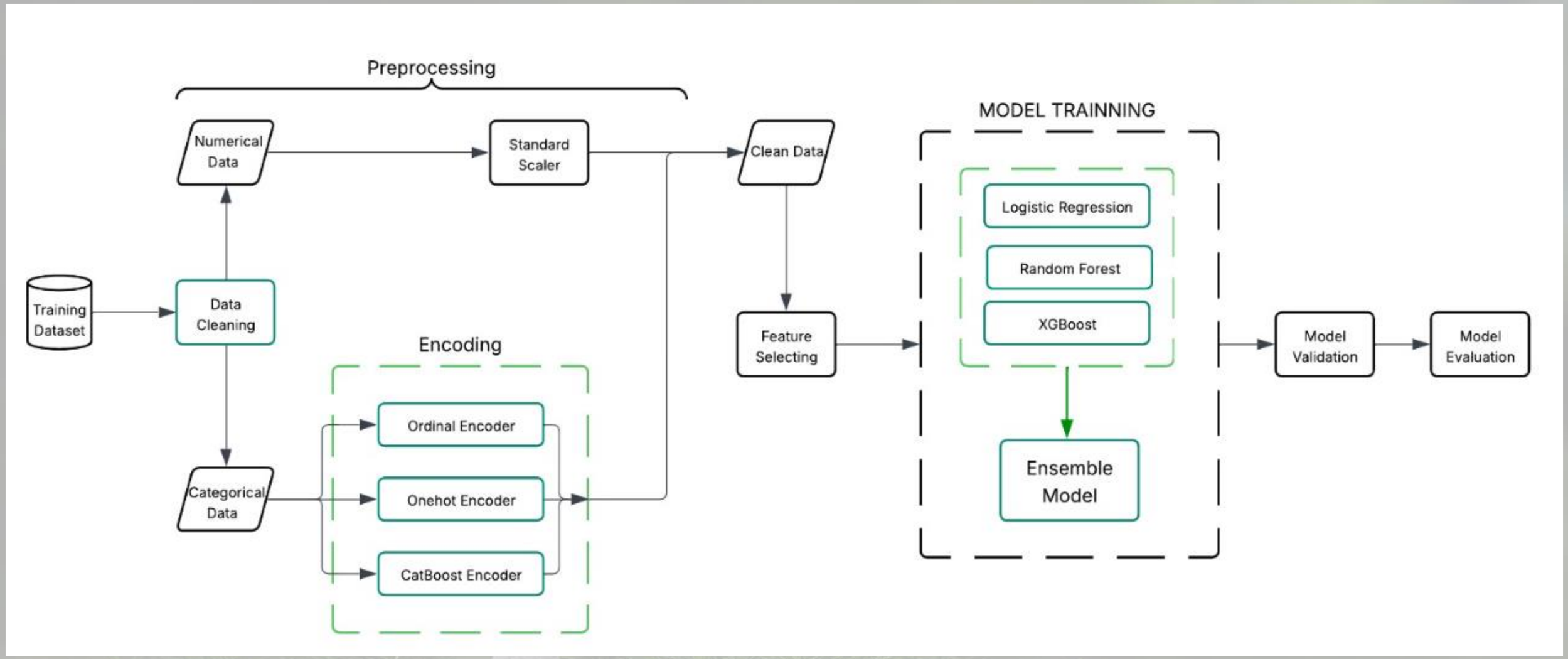
Round 1 -Recommendations

- Consider OneHotEncoding for categorical features for models sensitive to label encoding (like logistic regression, SVC).
- Include exploratory data analysis (EDA): class balance check, histograms, correlation heatmaps.

Round 2

- Advanced Data Imputation
- Enhanced Preprocessing
- Cross-Validation
- Ensemble Methods & Stacking
- Model Interpretability

Round 2


Preprocessing
Numerical Data
Standard Scaler
Clean Data
MODEL TRAINNING
Logistic Regression
Random Forest
XGBoost
Training Dataset
Data Cleaning
Encoding
Feature Selecting
Model Validation
Model Evaluation
Ordinal Encoder
Categorical Data
Onehot Encoder
CatBoost Encoder
Ensemble Model

Iterative Imputer

Iterative Imputer

```
# Apply Iterative Imputer using DecisionTreeClassifier
# Create the imputer pipeline
cat_imputer = Pipeline(steps=[
    ('imputer', IterativeImputer(
        estimator=DecisionTreeClassifier(),
        initial_strategy='most_frequent',
        max_iter=10,
        random_state=0
    )),
    ('rounder', FunctionTransformer(lambda x: np.round(x).astype(int)))
])

# Fit the pipeline
df_imputed_encoded = pd.DataFrame(
    cat_imputer.fit_transform(df_encoded[cat_cols]),
    columns=cat_cols
)

# Decode back to original categories
df[cat_cols] = encoder.inverse_transform(df_imputed_encoded)
df.head()
```

- Performing Iterative Imputation and not dropping any features
  - LogReg does worse than Round 1

    `Accuracy: 0.6454342011898914`
  - RandForest does better than Round 1 and 2

    `Accuracy: 0.9857540527263796`
  - XGBoost does better than Round 1 and 2

    `Accuracy: 0.9994541782653785`

```
#dropping columns with 80+%missing values
df.drop(columns=['veil-type', 'spore-print-color',
                 'veil-color', 'stem-root'],inplace = True)
```

- Removing features with 80+% missing values and then performing iterative permutation
  - LogReg performs better but still worse than in Round 1

    `Accuracy: 0.6476174881283773`
  - Random Forest performs even better

    `Accuracy: 0.9886196168331423`
  - XGBoost performs even better

```
Accuracy: 0.9993995960919164

Classification Report:
              precision    recall  f1-score   support

           e       1.00      1.00      1.00      8108
           p       1.00      1.00      1.00     10213

    accuracy                           1.00     18321
   macro avg       1.00      1.00      1.00     18321
weighted avg       1.00      1.00      1.00     18321
```

Model	accuracy	imputer1_accuracy	imputer2_accuracy_80%
logistic regression	0.651609	0.652202	0.647290
random forest	0.999789	0.999891	0.999945
xgboost	0.999454	0.999509	0.999509

Other Encoding Methods

Other Encoding Methods

```
# Encode the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # y = df['class']

# Build the preprocessor using OneHotEncoder for categoricals
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])
```

- Using OneHotEncoder Instead of OrdinalEncoder.
  - Best performance for Logistic Regression so far. Random Forest gives the worst results. Despite that we get the BEST results for XGBoost as well.
  - **XGBoost** performance:

```
Accuracy: 0.9992904317449921

Classification Report:
              precision    recall  f1-score   support

           e       1.00      1.00      1.00      8108
           p       1.00      1.00      1.00     10213

    accuracy                           1.00     18321
   macro avg       1.00      1.00      1.00     18321
weighted avg       1.00      1.00      1.00     18321
```

```
cat_encoder = CatBoostEncoder(cols=categorical_cols)
X_train_cat = cat_encoder.fit_transform(X_train[categorical_cols], y_train)
X_test_cat = cat_encoder.transform(X_test[categorical_cols])

# Scale numerical features
scaler = StandardScaler()
X_train_num = pd.DataFrame(scaler.fit_transform(X_train[numerical_cols]), columns=numerical_cols)
X_test_num = pd.DataFrame(scaler.transform(X_test[numerical_cols]), columns=numerical_cols)

# Reset index before concat to avoid misalignment
X_train_num = X_train_num.reset_index(drop=True)
X_train_cat = X_train_cat.reset_index(drop=True)
X_test_num = X_test_num.reset_index(drop=True)
X_test_cat = X_test_cat.reset_index(drop=True)

# Merge encoded features
X_train_encoded = pd.concat([X_train_num, X_train_cat], axis=1)
X_test_encoded = pd.concat([X_test_num, X_test_cat], axis=1)
```

- Considering CatBoostEncoder which lowers the accuracy score

Cross-Validation with XGBoost

Cross-Validation with XGBoost

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold

# 5-Fold Stratified Cross-Validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(xgb_pipeline, X, y_encoded, cv=cv, scoring='accuracy')

# Results
print("Cross-validation scores (accuracy):", scores)
print("Mean accuracy:", scores.mean())
print("Standard deviation:", scores.std())
```

```
Cross-validation scores (accuracy): [0.99975438 0.99967251 0.99950876 0.99950876 0.99975436]
Mean accuracy: 0.9996397536328819
Standard deviation: 0.0001110541423415606
```

- Performing 5-fold cross validation on our XGBoost model tells us that we don't have overfitting or underfitting issues here.
- The mean accuracy is **~99.97%**, which is outstanding.

Model	accuracy	imputer1_accuracy	imputer2_accuracy_80%	Onehot_accuracy
logistic regression	0.651609	0.652202	0.647290	0.798810
random forest	0.999789	0.999891	0.999945	0.999945
xgboost	0.999454	0.999509	0.999509	0.999891

Ensemble with StackingClassifier

Ensemble with StackingClassifier

```python
# Base models
estimators = [
    ('lr', LogisticRegression(penalty='l2',           # Ridge regularization
                              C=1.0,                  # Inverse of regularization strength
                              solver='lbfgs',         # Good for multiclass and small datasets
                              max_iter=10000,         # Increased for convergence
                              random_state=42,
                              class_weight='balanced' # Handles imbalanced classes
                              )),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')),
    ('xgb', XGBClassifier(n_estimators=200, max_depth=20, eval_metric='mlogloss'))
]

# Meta-classifier (can be a simple Logistic Regression)
stacking_clf = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(),
    cv=5
)

# Create a full pipeline including preprocessor
pipeline_stacking = Pipeline(steps=[
    ('preprocessor', preprocessor),  # reuse preprocessor from above
    ('stacking', stacking_clf)
])

pipeline_stacking.fit(X_train, y_train)
print("Stacking Classifier Accuracy:", pipeline_stacking.score(X_test, y_test))
```

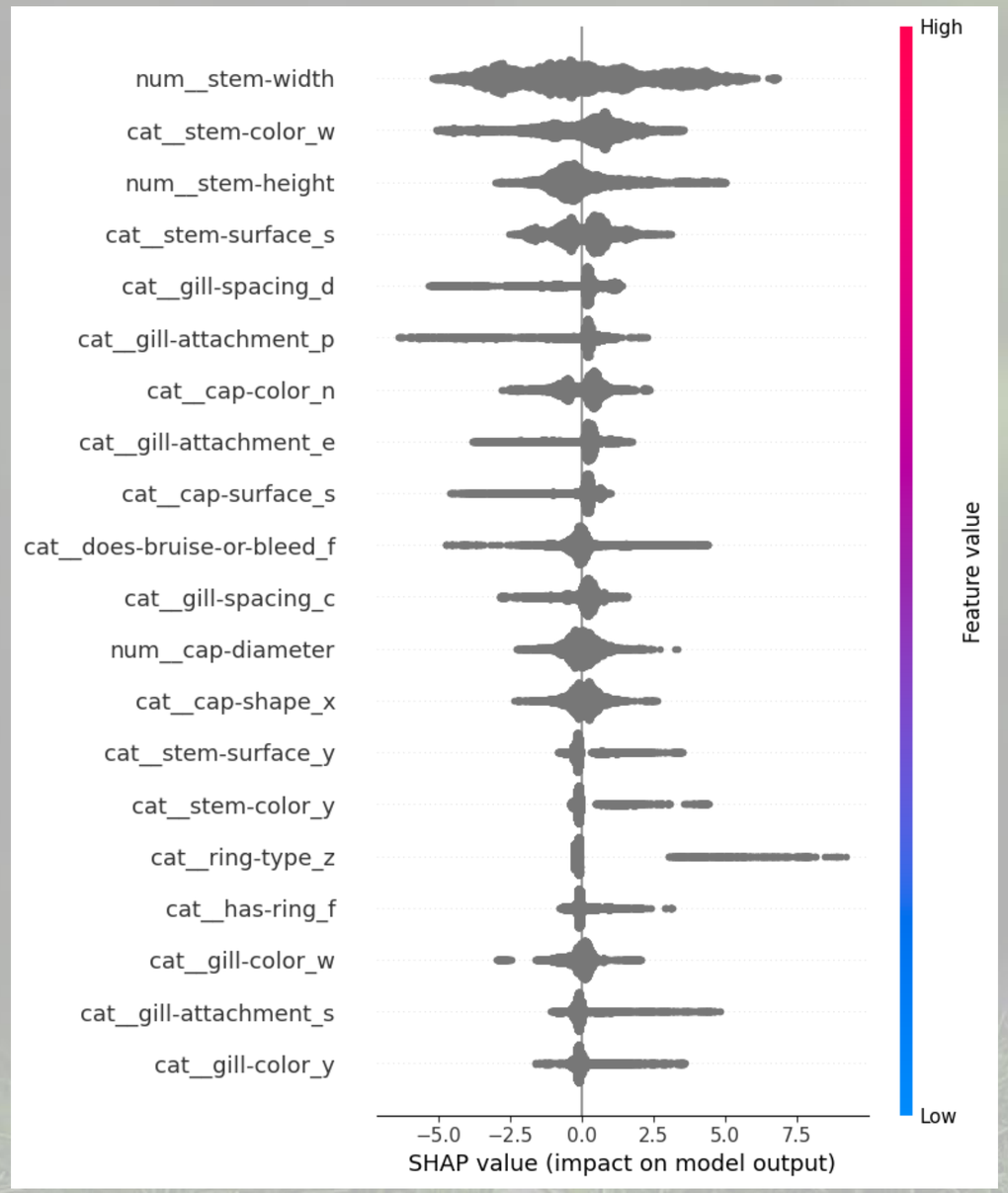
```
Stacking Classifier Accuracy: 0.9998362534796136
```

- building an Ensemble with StackingClassifier using Logistic Regression, Random Forest and XGBoost. It gives us the best accuracy so far: ~99.99%
- I also do cross-validation and it is not overfitting or underfitting

```
Mean accuracy: 0.9998359854417841
Standard deviation: 0.00013641295959334083
```

Model Interpretability - SHAP

Model Interpretability - SHAP



- Feature importance (which features impact the model most)
- Direction of impact (whether a high or low value of the feature increases or decreases prediction)

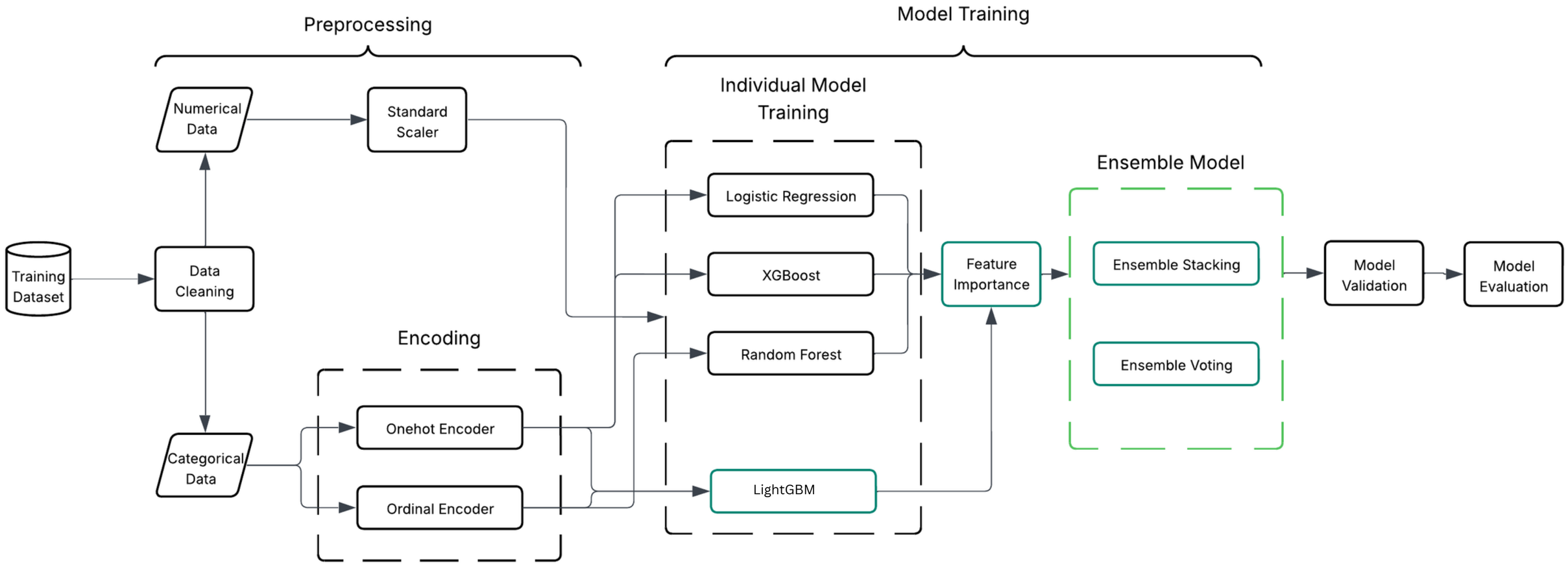
Round 2 -Recommendations

- Try different meta-learners (e.g., LightGBM).
- Perform feature selection to reduce redundancy.
- Introduce model weighting in stacking.

Round 3

- Reordering Base Models
- Light GBM
- Working With Top Features
- Stacking Classifier
- ROC Curve + AUC

Round 3


Preprocessing
Model Training
Individual Model Training
Numerical Data
Standard Scaler
Training Dataset
Data Cleaning
Logistic Regression
XGBoost
Random Forest
LightGBM
Feature Importance
Ensemble Model
Ensemble Stacking
Ensemble Voting
Model Validation
Model Evaluation
Encoding
Categorical Data
Onehot Encoder
Ordinal Encoder

Reordering Base Models

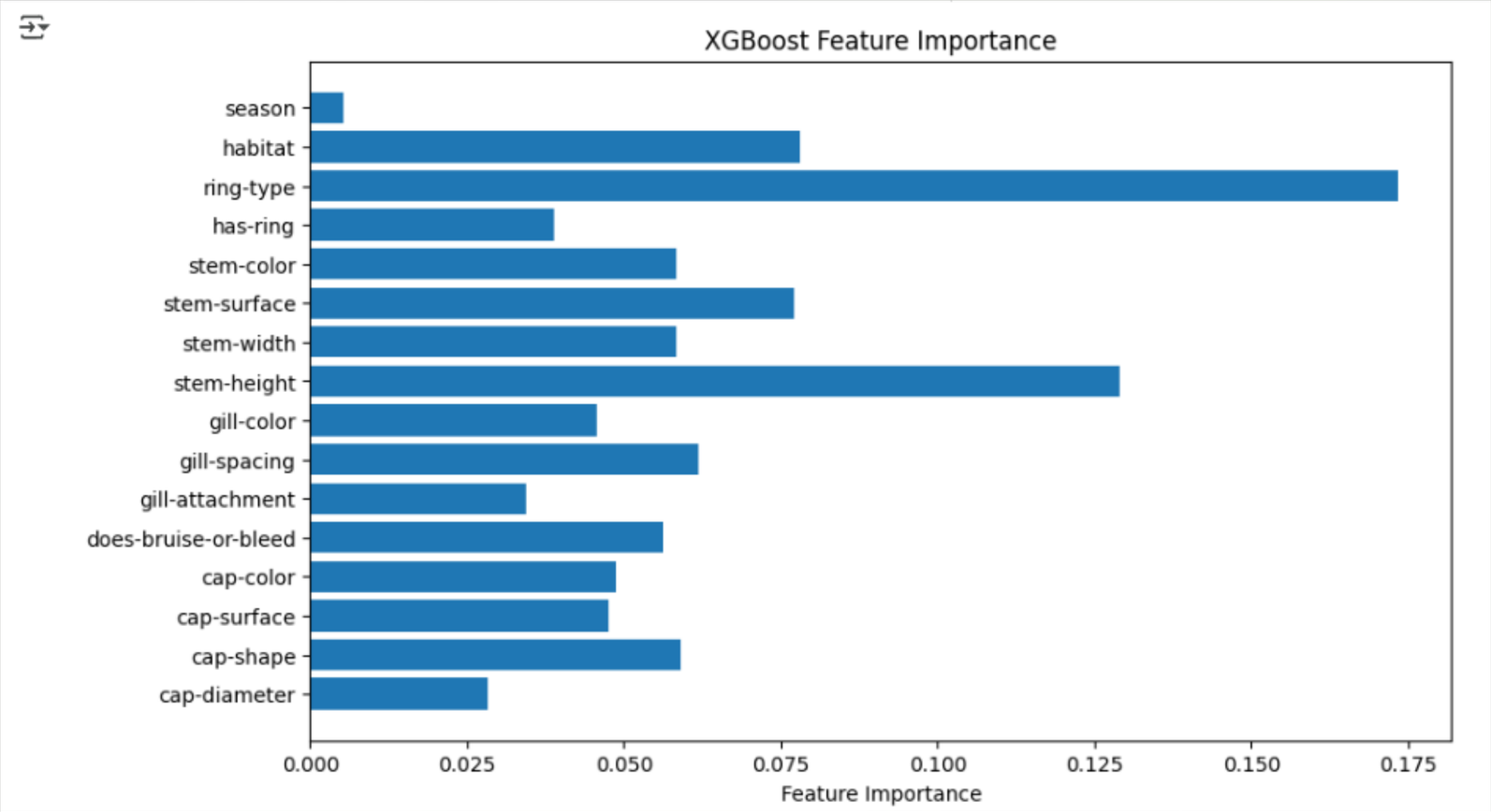
Reordering Base Models

	Model	Accuracy
0	LR First	0.999945
1	XGB First	0.999945
2	RF First	0.999945
3	RF First + Ridge Meta	0.999945

- Changing the order of models in the ensemble (e.g., XGB → LR → RF or Different Orders) does not affect the final accuracy.
- This is because most ensemble techniques (like voting or averaging) are order-independent—they treat each model equally, regardless of position.
- The consistent results support that imputation and encoding strategy is robust.

Feature Importance

Feature Importance

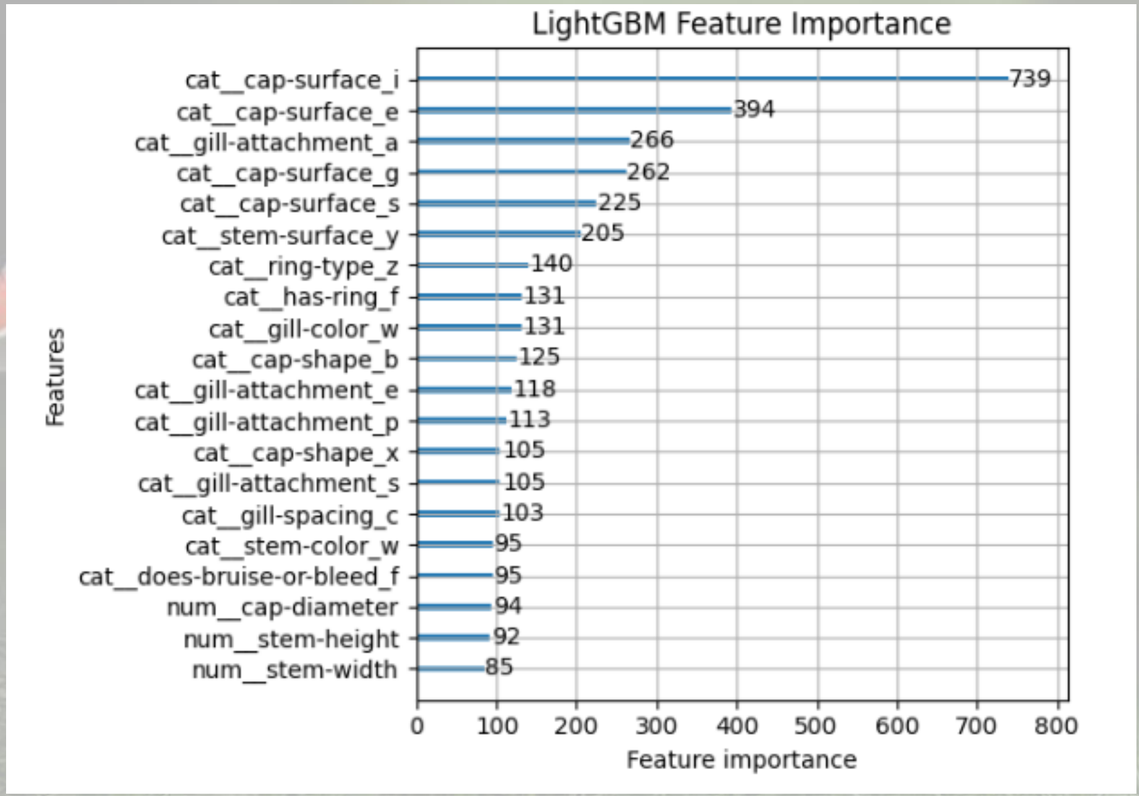


- Visualized feature importance from the trained XGBoost model.
- Top contributors: ring-type, stem-height, and habitat.
- Helps identify which features have the most influence on predicting whether a mushroom is edible or poisonous.

Light GBM

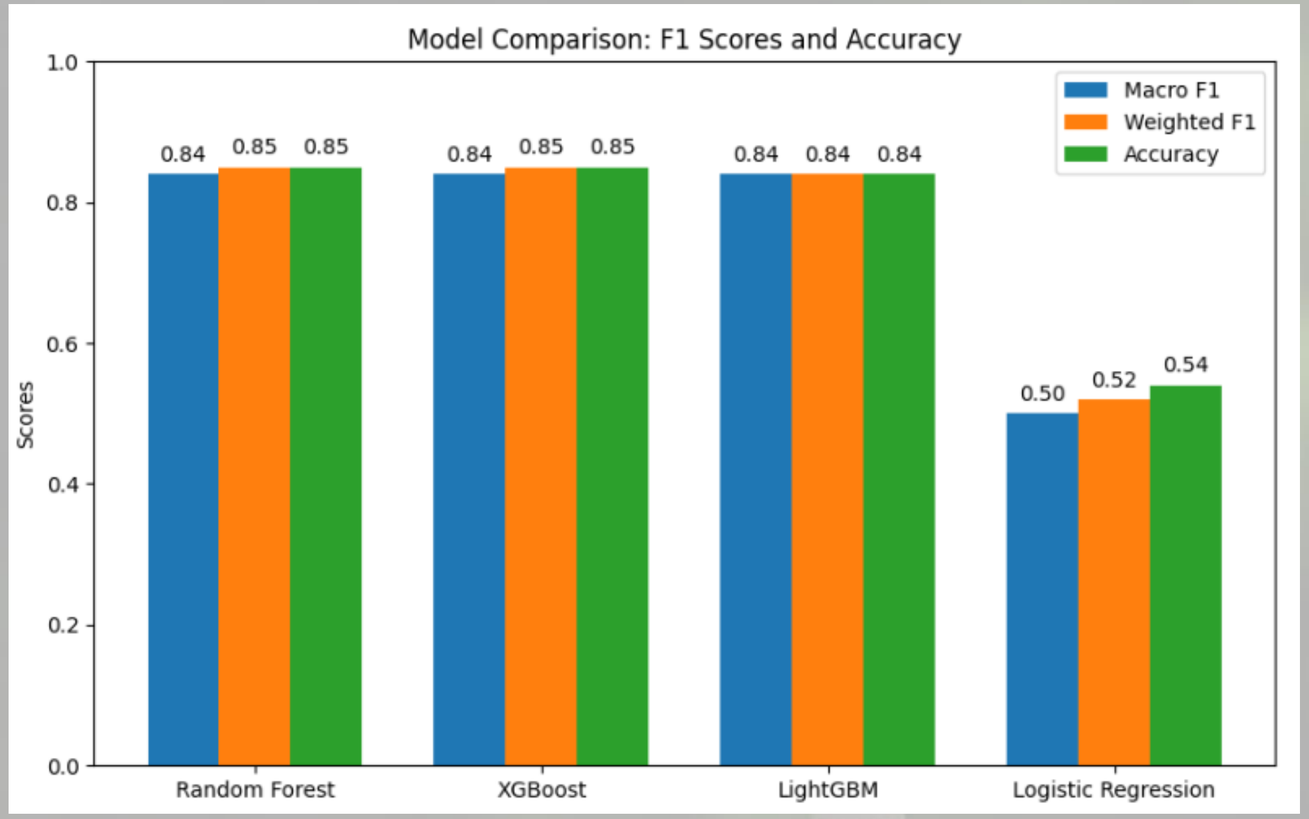
Light GBM

- **Light GBM** accuracy is at 0.99858 **(99.86%).** This is slightly lower than the stacking models (99.99%). —> still exceptionally high and likely overkill in most practical settings.
- **LightGBM + Stacking Classifier** accuracy is **99.99%** reduced error by more than 80%



Working With Top Features

Working With Top Features



Accuracy of **Random Forest Classifier:** 0.8469515856121391
Accuracy of **Logistic Regression:** 0.5407455924894929
Accuracy of **XGBoost Classifier:** 0.8468424212652148
Accuracy of **LightGBM Classifier**: 0.8434583265105616

Model	Accuracy
Random Forest	0.85
XGBoost	0.85
LightGBM	0.84
Logistic Regression	0.54

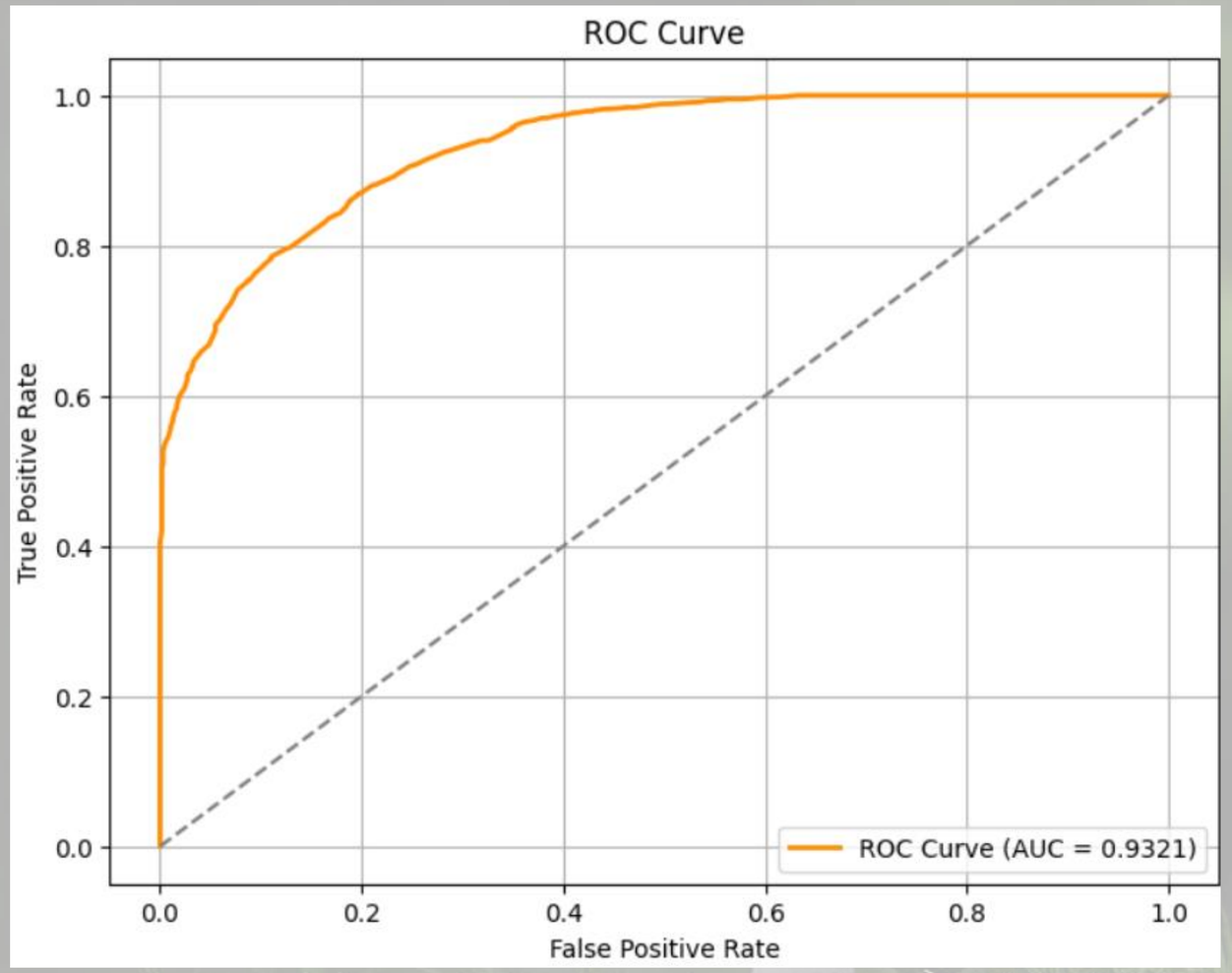
- XGBoost and Random Forest are the top-performing models.
- LightGBM performs very closely and is also an excellent choice.
- Logistic Regression is not suitable for this dataset — likely due to non-linearity and complex feature interactions
- Tree-based models maintain the best precision/recall balance across both classes.

Stacking Classifier with Top Features

```python
from sklearn.ensemble import StackingClassifier

# Define base learners
base_learners = [
    ('rf', RandomForestClassifier(random_state=42)),
    ('lr', LogisticRegression(random_state=42)),
    ('xgb', XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')),
    ('lgbm', LGBMClassifier(random_state=42))
]

# Define the meta-learner (final estimator)
meta_learner = LogisticRegression(random_state=42)

# Create the stacking classifier
stacking_clf = StackingClassifier(
    estimators=base_learners,
    final_estimator=meta_learner,
    cv=5,
    passthrough=False,  # Set True if you want raw features + predictions for meta-learner
    n_jobs=-1
)

# Create pipeline
stacking_pipeline = Pipeline([
    ('preprocessing', preprocessor),
    ('stacking_classifier', stacking_clf)
])

# Fit the model
stacking_pipeline.fit(X_train, y_train)

# Evaluate
stacking_accuracy = stacking_pipeline.score(X_test, y_test)
print("Accuracy of Stacking Classifier:", stacking_accuracy)
```

Accuracy of Stacking Classifier: **0.8466240925713662** —> Choosing only top features does not improve the accuracy of the model.

ROC Curve + AUC



- The model balances sensitivity and specificity well.
- An AUC of **0.9321** confirms that the classifier is reliable and robust.
- This performance is particularly suitable for applications where accurate class separation is critical.

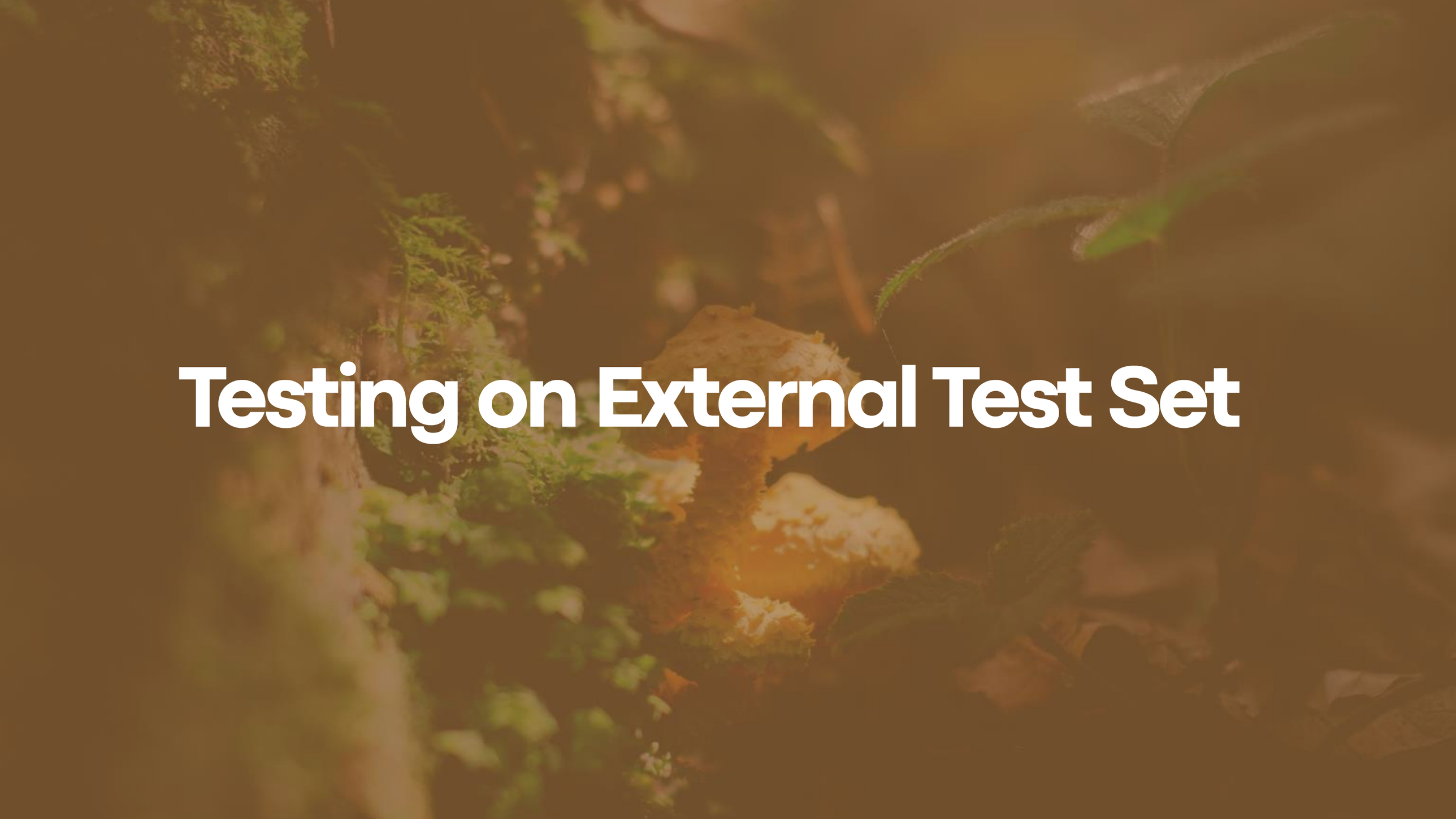
Round 3 -Recommendations

After 3 rigorous rounds of experimentation and tuning, the best performing model is:

- Stacking Classifier
  - > Components: Logistic Regression, Random Forest, XGBoost, LightGBM
  - > Achieved 99.99% accuracy on validation data- 511 900 samples

To ensure the model performs well outside of the training environment:

> Test the final model on an unseen Kaggle external test set


Testing on External Test Set

Kaggle Dataset

Dataset Source:

- Name: One Million Mushrooms
- Kaggle Link: Tertiary Mushroom Dataset – Kaggle
- Records: Over 1 million mushroom samples
- Purpose: Used exclusively for final model evaluation — not used during training or tuning

Why This Dataset?

- Realistic high-volume data
- Same 21 features as UCI dataset
- Ideal for final benchmarking and testing generalizability

Test Objective:

To confirm the final stacking classifier performs reliably on unseen, real-world-style data

Testing on External Test Set

```python
# Get the stacking model
stacking_model = pipeline_lgbm_stack.named_steps['stacking']

# Predict
y_pred_external = stacking_model.predict(X_external_processed)
#y_pred_external_labels = label_encoder.inverse_transform(y_pred_external)

# Evaluate
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Accuracy
y_external_test_encoded = label_encoder.transform(y_external_test)
accuracy_ext = accuracy_score(y_external_test_encoded, y_pred_external)
print("External Test Set Accuracy:", accuracy_ext)

# Classification Report
print("Classification Report:")
print(classification_report(y_external_test_encoded, y_pred_external))

# Confusion Matrix
cm = confusion_matrix(y_external_test_encoded, y_pred_external)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',
            xticklabels=['Edible', 'Poisonous'], yticklabels=['Edible', 'Poisonous'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix: External Test Set')
plt.show()
```

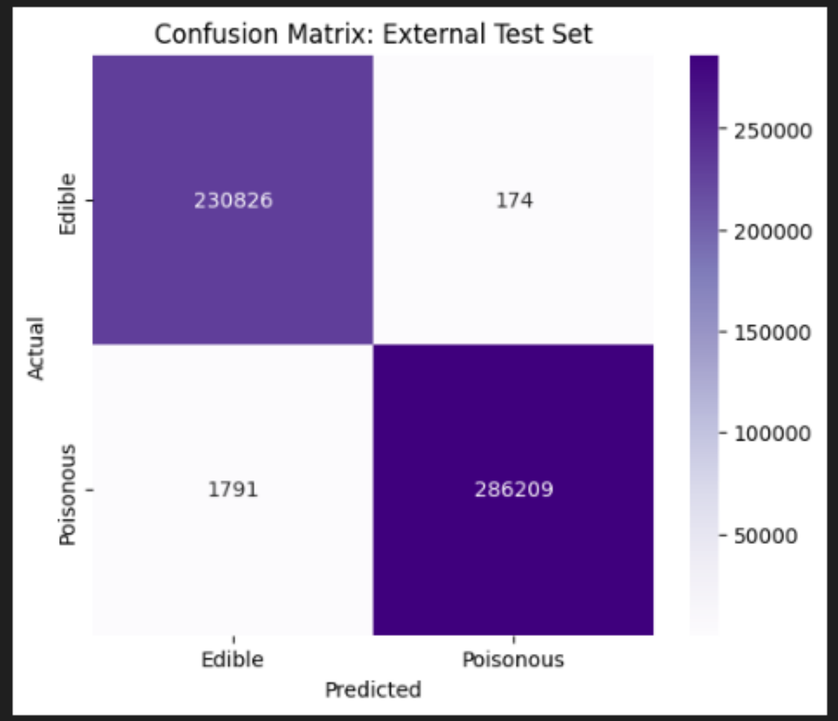
- Tested the external set on the best performing model- **StackingClassifier incorporating Logistic Regression, Random Forest, XGBoost, and LightGBM**

- **Accuracy**: **~99.62%**
  > Shows strong generalization capability.

Testing on External Test Set

External Test Set Accuracy: 0.99621387283237
Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	231000
1	1.00	0.99	1.00	288000
accuracy			1.00	519000
macro avg	1.00	1.00	1.00	519000
weighted avg	1.00	1.00	1.00	519000



- Class-wise Performance:
  > Class 0 – Precision: 0.99, Recall: 1.00, F1-score: 1.00
  > Class 1 – Precision: 1.00, Recall: 0.99, F1-score: 1.00

- Averaged Metrics:
  > Macro & weighted averages ≈ 1.00, indicating balanced performance.
  > No bias observed despite Class 1 having more samples (288k vs. 231k).


SPJ GLOBAL
S P Jain School of Global Management
DUBAI • MUMBAI • SINGAPORE • SYDNEY
THANK YOU FOR YOUR ATTENTION