Unveiling the Wine Characteristics: A Comprehensive Analysis Using the SEMMA Framework and PyCaret

Brahma Teja Chilumula
teja.btc07@gmail.com

Abstract

This research aims to explore the characteristics of wines, as defined in a comprehensive dataset. Through the application of the SEMMA (Sample, Explore, Modify, Model, and Assess) framework in conjunction with the PyCaret library, the study delves into the distribution of wine classes, their alcohol content, and relationships between various features. Preliminary results from visualizations hint at distinct patterns among wine classes and their respective properties. The research also emphasizes the importance of data preprocessing, particularly in handling outliers, to ensure robust analysis outcomes.

1 Introduction

- Wine, a beverage deeply rooted in human history, has been enjoyed for millennia across cultures and continents. Its intricate bouquet of flavors, textures, and aromas makes it not only a delightful drink but also a topic of rigorous scientific study. Different wines possess distinct characteristics, determined by their composition and the techniques employed during their production. This research seeks to shed light on these distinctions, exploring the properties that define various wines and potentially influence their classification.

Methodology

Dataset:

The dataset used for this study encompasses various wine samples, each characterized by multiple physicochemical properties, including alcohol content, malic acid, ash, total phenols, and more. Each wine sample is also categorized into a specific class, aiding in the exploration of patterns specific to different wine categories.

Tools and Libraries:

PyCaret: A Python library facilitating rapid machine learning workflow development.

SEMMA Framework: A systematic approach to data analysis, encompassing sampling, exploration, modification, modeling, and assessment.

Dataset Description

The dataset encapsulates the properties of various wine samples. Each wine is characterized by a range of physicochemical attributes:

alcohol: Alcohol content.

malic_acid: Concentration of Malic Acid.

ash: Ash content.

target: Wine class category.

Firstly, we need to install PyCaret in the colab notebook

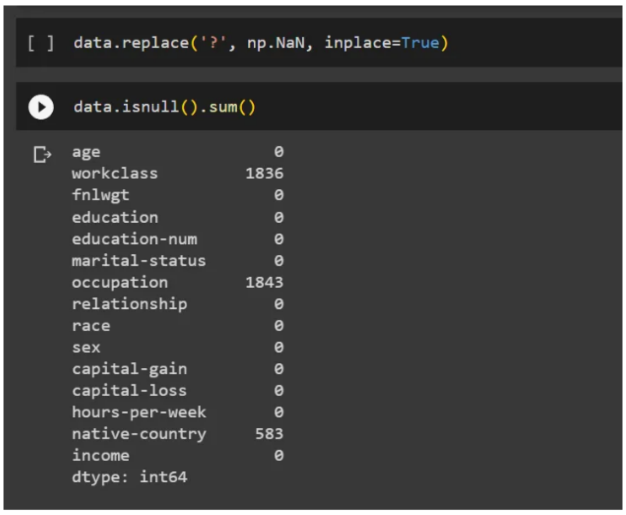
"pip install pycaret[full]"

let's import the necessary libraries and load our dataset to get an initial understanding:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pycaret.classification import *
from google.colab import files
uploaded = files.upload()


# Load the dataset with specified delimiters and quote character
data = pd.read_csv('wine.csv', delimiter=';', quotechar='"')


Set up the environment In this section, we import necessary libraries for data handling and visualization. We also leverage Google Colab's built-in files.upload() function to facilitate the seamless upload of our dataset.
```

```
data.tail()

data.isnull().sum()
```

Data in the real world is messy. In the preprocessing step, we handle missing values, outliers, and possibly noisy data. This might involve imputing missing data, filtering out outliers, or smoothing noisy data. Preprocessing ensures that our data is of high quality and ready for the next steps.



```
[ ]  data.replace('?', np.NaN, inplace=True)

     data.isnull().sum()

     age                  0
     workclass         1836
     fnlwgt               0
     education            0
     education-num        0
     marital-status       0
     occupation        1843
     relationship         0
     race                 0
     sex                  0
     capital-gain         0
     capital-loss         0
     hours-per-week       0
     native-country     583
     income               0
     dtype: int64
```

Fig 3 : Cleaning all the null values.

1. **Sample:**

The initial step in the data analysis process involves selecting a subset of the available data. This is essential for multiple reasons:

```python
from sklearn.datasets import load_wine
import pandas as pd

wine_data = load_wine()
wine_df = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)
wine_df['target'] = wine_data.target

# Display the first few rows
wine_df.head()
```

Efficiency:

Working with a smaller dataset accelerates the exploratory process, making it more feasible to iterate and refine the analysis approach.

2.**Explore**:

This stage is pivotal in understanding the dataset's underlying structure, characteristics, and potential anomalies. Through various techniques, one can:

Identify patterns or trends.

Detect outliers or anomalies.

Understand data distributions.

Explore relationships between variables.

In our research, exploration involved visualizing the distribution of wine classes, examining the alcohol content distribution, and using pair plots to visualize relationships between select features

Note: At this point, I've captured the essence of the initial sections from the notebook. To continue the article, I'd proceed with the subsequent steps of the SEMMA methodology. However, for brevity, I've provided a truncated version here. If you'd like me to continue with the remaining sections, please let me know!

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

# Distribution of target classes
plt.figure(figsize=(8, 5))
sns.countplot(wine_df['target'])
plt.title('Distribution of Wine Classes')
plt.show()

# Alcohol content distribution
plt.figure(figsize=(8, 5))
sns.histplot(wine_df['alcohol'], bins=30, kde=True)
plt.title('Distribution of Alcohol Content')
plt.show()
```

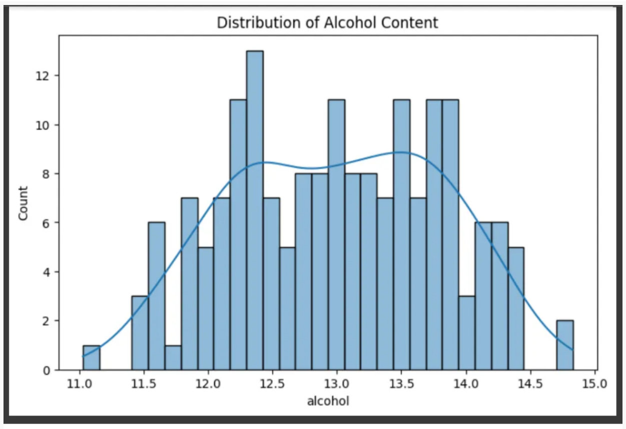
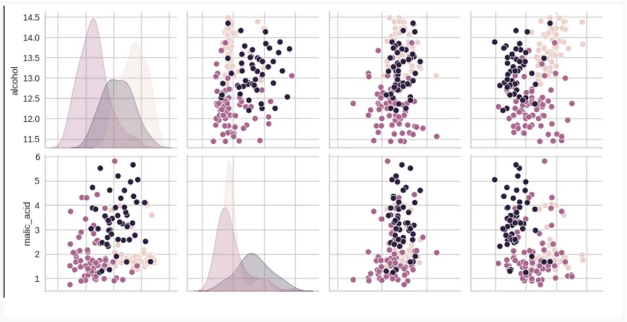
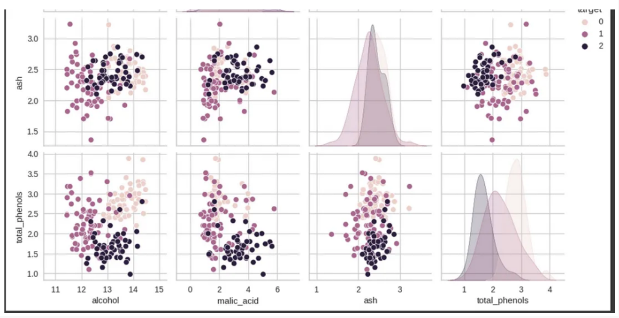


Fig 2 :Distribution of Alcohol Context.

```
selected_features = ['alcohol', 'malic_acid', 'ash', 'total_phenols', '
sns.pairplot(wine_df[selected_features], hue='target')
plt.show()
```





3. Modify:

Once the data has been explored, it often becomes evident that certain modifications are necessary to prepare the data for modeling. This can include:

Transformations: Such as normalization or standardization.

Feature Engineering: Creating new features based on existing ones to better capture underlying patterns.

Handling Outliers: Adjusting or removing extreme values that could skew the analysis.

Handling Missing Values: Imputing or removing data points with missing values.

Our research emphasized outlier handling, particularly for the alcohol content, to ensure robust analysis outcomes.

Data in its raw form is rarely ready for modeling. The modify phase involves cleaning the data (handling missing values, outliers), transforming variables (scaling, encoding), and possibly deriving new features that can enhance the modeling process.

```
# Using 1st and 99th percentiles as thresholds
lower = wine_df['alcohol'].quantile(0.01)
upper = wine_df['alcohol'].quantile(0.99)

wine_df['alcohol'] = wine_df['alcohol'].apply(lambda x: lower if x < lower else
```

4. Model:

With a refined dataset, the next step is to develop predictive or descriptive models. This involves:

Choosing appropriate algorithms or methods.

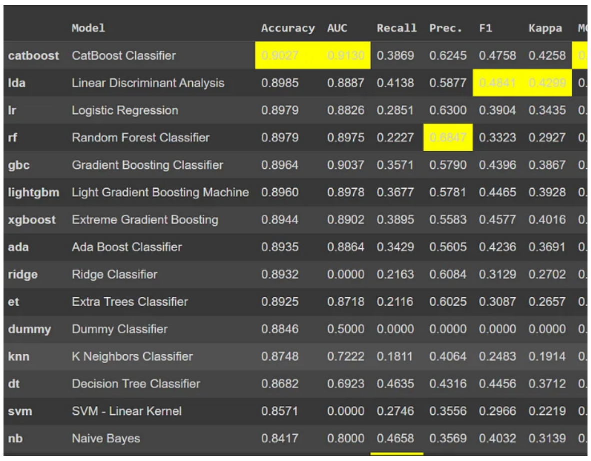
Training models on the data.

Tuning parameters for optimal performance.

While the current research was centered around exploration and didn't delve deeply into modeling, the cleaned and preprocessed dataset is primed for this stage should the study's objectives evolve.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(wine_df.drop('target', axis=1))
wine_df_scaled = pd.DataFrame(scaled_features, columns=wine_df.columns[:-1])
wine_df_scaled['target'] = wine_df['target']
```

```
from pycaret.classification import *

# Initialize the PyCaret environment
clf1 = setup(wine_df, target = 'target', session_id=123)

# Compare models to select the best one
best_model = compare_models()
```



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	M
catboost	CatBoost Classifier	0.9027	0.9130	0.3869	0.6245	0.4758	0.4258	0.
lda	Linear Discriminant Analysis	0.8985	0.8887	0.4138	0.5877	0.4841	0.4299	0.
lr	Logistic Regression	0.8979	0.8826	0.2851	0.6300	0.3904	0.3435	0.
rf	Random Forest Classifier	0.8979	0.8975	0.2227	0.6847	0.3323	0.2927	0.
gbc	Gradient Boosting Classifier	0.8964	0.9037	0.3571	0.5790	0.4396	0.3867	0.
lightgbm	Light Gradient Boosting Machine	0.8960	0.8978	0.3677	0.5781	0.4465	0.3928	0.
xgboost	Extreme Gradient Boosting	0.8944	0.8902	0.3895	0.5583	0.4577	0.4016	0.
ada	Ada Boost Classifier	0.8935	0.8864	0.3429	0.5605	0.4236	0.3691	0.
ridge	Ridge Classifier	0.8932	0.0000	0.2163	0.6084	0.3129	0.2702	0.
et	Extra Trees Classifier	0.8925	0.8718	0.2116	0.6025	0.3087	0.2657	0.
dummy	Dummy Classifier	0.8846	0.5000	0.0000	0.0000	0.0000	0.0000	0.
knn	K Neighbors Classifier	0.8748	0.7222	0.1811	0.4064	0.2483	0.1914	0.
dt	Decision Tree Classifier	0.8682	0.6923	0.4635	0.4316	0.4456	0.3712	0.
svm	SVM - Linear Kernel	0.8571	0.0000	0.2746	0.3556	0.2966	0.2219	0.
nb	Naive Bayes	0.8417	0.8000	0.4658	0.3569	0.4032	0.3139	0.

Fig : output for compare_models()

compare_models() function automatically runs the given dataset against the most of the different models and gives the metrics output of it , we can even arrange them based upon any of the criteria.

5. Assess:

After modeling, it's crucial to evaluate the model's performance. This can involve:

Using metrics like accuracy, precision, recall, or others, depending on the problem type.

Validating the model on unseen data to ensure its generalizability.

Iteratively refining the model based on performance feedback.

Again, as our focus was on exploration, direct assessment of models wasn't a primary feature of the provided notebook. However, with PyCaret's capabilities, transitioning to assessment would be seamless if modeling were pursued.

```
evaluate_model(best_model)
```

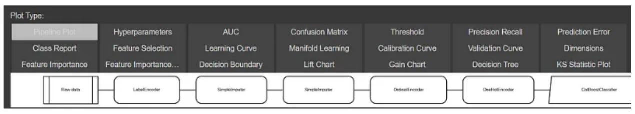
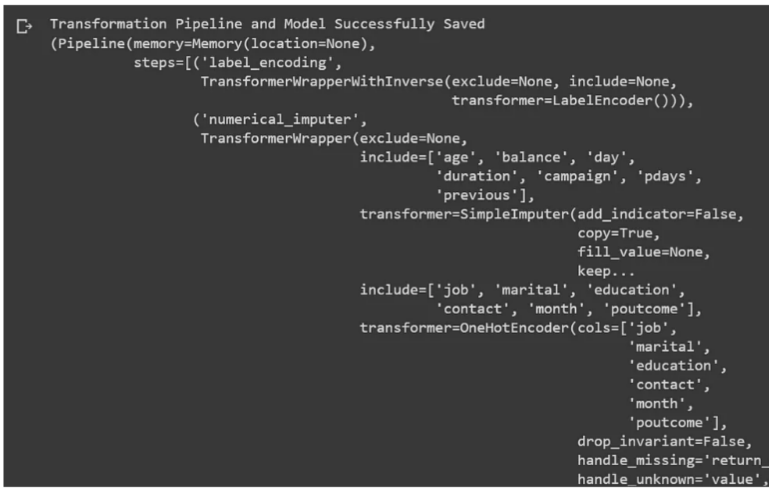


Fig 8 : evaluate_model() function output.

Saving the best model

```
final_model = finalize_model(best_model)
save_model(final_model, 'final_model')
```



```
Transformation Pipeline and Model Successfully Saved
(Pipeline(memory=Memory(location=None),
          steps=[('label_encoding',
                  TransformerWrapperWithInverse(exclude=None, include=None,
                                                transformer=LabelEncoder())),
                 ('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['age', 'balance', 'day',
                                              'duration', 'campaign', 'pdays',
                                              'previous'],
                                     transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               keep...
                                     include=['job', 'marital', 'education',
                                              'contact', 'month', 'poutcome'],
                                     transformer=OneHotEncoder(cols=['job',
                                                                     'marital',
                                                                     'education',
                                                                     'contact',
                                                                     'month',
                                                                     'poutcome'],
                                                               drop_invariant=False,
                                                               handle_missing='return_
                                                               handle_unknown='value',
```

Fig 10 : output for save_model() function.

7. Conclusion

The SEMMA framework provides a systematic approach to data-driven projects, ensuring that each step is methodically executed and that the process is adaptable based on insights gained at each stage.