CENTENNIAL COLLEGE

# BA 723 – Business Analytics Capstone

# Project: Brewery Recipe Optimization


- Max Sergio Causso Fretel - 301455365

CENTENNIAL COLLEGE

# Table of contents

## Contents

CENTENNIAL COLLEGE

# 0. Executive summary

## 0.1 Executive Introduction

This project demonstrates an end-to-end analytics solution using a publicly available brewery dataset. Leveraging expertise in data analytics, business intelligence, and cloud technologies, it showcases how modern data platforms and machine learning techniques can be applied to real-world datasets to generate meaningful business insights and predictive capabilities. The solution integrates Python for modeling, Power BI for visualization, and Azure for scalable data storage, exemplifying a full-stack analytical approach.

## 0.2 Executive Objective

The primary objective is to predict the alcohol by volume (ABV) of beer recipes using machine learning models and support decision-making with interactive dashboards. This addresses both operational questions, such as identifying which features most influence alcohol levels, and strategic considerations, such as determining how to optimize recipe design for target alcohol outcomes. Additionally, the project demonstrates the feasibility of cloud-based deployment, supporting scalability in both academic environments and real-world brewery operations.

## 0.3 Executive Model Description

Multiple regression and tree-based models were evaluated, and a decision tree regressor was ultimately selected for its interpretability. The model was tested both with and without Original Gravity (OG) as a predictor, and including OG was found to significantly improve predictive power ($R^2 = 0.83$ with OG vs. 0.46 without). These findings **offer** actionable insights for brewers and product developers.

## 0.4 Executive Recommendations

- Incorporate measurements of Original Gravity (OG) in the early recipe formulation stage to improve the accuracy of alcohol content predictions (ABVCalculator, n.d.)
- Use tree-based modeling approaches for rapid prototyping and ease of interpretation when refining beer recipes.
- Deploy interactive Power BI dashboards to enable continuous monitoring of brewing trends and fermentation patterns.
- Leverage cloud storage solutions (e.g., Azure Blob Storage) for scalable, centralized data access.

CENTENNIAL COLLEGE

# 1. Introduction

## 1.1 Background

The beer industry constantly seeks to innovate while maintaining the quality and consistency of its products. Alcohol by volume (ABV) is a critical metric in recipe formulation, as it influences product characteristics and carries significant regulatory (e.g., labeling laws) and commercial implications. For example, in Canada, any beverage exceeding 1.1% ABV must include an ABV statement on its label (Canadian Food Inspection Agency [CFIA], 2025). Additionally, quality guidelines set strict limits on the allowable variation between the actual ABV and the amount declared on the label (Liquor Control Board of Ontario, 2025). Leveraging historical brewing data, this project seeks to model and understand the relationship between ingredients, brewing parameters, and ABV.

## 1.2 Problem Statement

Despite the abundance of available recipe data, predicting ABV with high accuracy remains a challenge without advanced analytical techniques. Manual calculations or simple averages are insufficient to capture the complexity of the problem. For example, in Canada, any beverage exceeding 1.1% ABV must include an ABV statement on its label (Canadian Food Inspection Agency [CFIA], 2025). Additionally, quality guidelines set strict limits on the allowable variation between the actual ABV and the amount declared on the label (Liquor Control Board of Ontario, 2025). While standard formulas exist for calculating ABV from OG and FG, these formulas cannot be applied when those values are not known in advance (ABVCalculator, n.d.).

## 1.3 Objectives & Measurements

- Main Objective: **To predict** beer alcohol content (ABV) from recipe-level data using machine learning models.
- Secondary Objectives:
    - To categorize beer styles into broader groups for comparative analysis.
    - To assess the impact of including versus excluding OG on model performance.
    - To develop a Power BI dashboard to present actionable metrics and trends.
- Metrics for Success:
    - Coefficient of determination ($R^2$) and Root Mean Square Error (RMSE) to evaluate predictive accuracy.
    - Decision tree complexity (e.g., number of nodes) as a measure of model interpretability.

- Clarity and usefulness of the Power BI dashboard for non-technical users, evaluated via user feedback.

## 1.4 Assumptions and limitations

- OG is assumed to be a reliable and consistently measured variable when available.
- The dataset, while extensive, contains missing and potentially inconsistent values (e.g., styles or ingredients), which were addressed through grouping and data cleaning.
- Predictions are based on historical recipes and may not fully capture innovations in brewing techniques or ingredients do not present in the dataset.
- Azure cloud storage was used in a low-cost tier for academic purposes; performance may differ in production-scale environments.

# 2. Data Source

## 2.1 Dataset Introduction

The dataset used in this project was sourced from Kaggle and includes over 70,000 beer recipes, each described by a combination of numeric and categorical attributes. These include original and final gravities (OG and FG), bitterness units (IBU), fermentation temperature, and style classifications, among others. While comprehensive, the raw dataset presented challenges related to inconsistency in formats, missing values, and categorical noise, making data cleaning a foundational step in the analysis pipeline.

## 2.2 Exclusions

Several variables were excluded from the dataset either due to limited relevance to the modeling objective or to prevent unnecessary complexity in the feature space. Fields such as 'BeerID', 'Name', 'UserId', and 'URL' were dropped early, as they represented identifiers or external references without predictive value. Additionally, the 'Style' column was replaced by a grouped version ('Style_Grouped') to reduce categorical granularity while preserving interpretability.

Additionally, “PrimingMethod” and “PrimingAmount”, although potentially useful, were removed after exploratory testing showed that their inclusion led to a large number of sparse dummy variables, increasing the risk of overfitting and diluting the model’s focus. Instead of direct inclusion, a grouping and transformation strategy was explored, though ultimately these features were deemed better excluded in favor of model simplicity and performance stability.

The target variable itself, 'ABV', also required careful filtering. Observations with alcohol values outside the 2%–13% range were excluded, as they typically

represented non-standard brewing recipes (e.g., low-alcohol experimental batches or extreme high-gravity brews) that are uncommon in traditional or commercial contexts. Their inclusion would have skewed the model's learning process, introducing variance that lacked generalizability for typical beer styles

Finally, a small number of extreme outliers in numeric variables such as 'BoilTime' and 'Efficiency' were identified and removed based on distributional analysis, as they introduced noise and disproportionately influenced model behavior without offering interpretable insight.

## 2.3 Data Preparation

Data preparation was guided by the principle of enhancing interpretability while maintaining predictive power. Missing numerical values were imputed using the median, whereas categorical variables were simplified through consolidation into broader groups, for instance, merging dozens of unique beer styles into six overarching categories. Feature engineering was also applied to extract additional insight, such as the difference between OG and FG (indicative of fermentation activity) and ratios like IBU-to-OG, which serve as proxies for balance between bitterness and sugar content. One-hot encoding was applied with care to prevent excessive dimensionality, especially in categorical fields with sparse distributions.

## 2.4 Data Dictionary

| Variable name | Description | Type |
|---|---|---|
| BeerID | Unique identifier for each recipe | Integer |
| Name | Name of the beer recipe | String |
| URL | Link to the online recipe | String |
| Style | Specific beer style (e.g., “American IPA”, “Pale Ale”) | String |
| StyleID | Numeric style code | Integer |
| Size(L) | Batch size in liters | Float |
| OG | Original Gravity (sugar content before fermentation) | Float |
| FG | Final Gravity (sugar content after fermentation) | Float |
| ABV | Alcohol by volume (%) | Float |
| IBU | International Bitterness Units | Float |
| Color | Color of the beer | Float |
| BoilSize | Volume of wort boiled(liters) | Float |
| BoilTime | Duration of the boil(minutes) | Float |
| BoilGravity | Gravity of wort during boil | Float |
| Efficiency | Mash efficiency(percentage) | Float |
| MashThickness | Ratio of water to grain(L/kg) | Float |
| SugarScale | Scale used for OG/FG(e.g. “Specific gravity”) | Categorical |

| | | |
|---|---|---|
| BrewMethod | Brewing method used (e.g. “All grain”, “Extract”) | Categorical |
| PitchRate | Yeast pitching rate(million cells per mL per °Plato) | Float |
| PrimaryTemp | Primary fermentation temperature (°C) | Float |
| PrimingMethod | Method used to carbonate beer after fermentation | Categorical |
| PrimingAmount | Amount of sugar used for priming | String |
| UserID | ID of the user who submitted the recipe | Integer |

**Table 2.4a: Description of the dataset columns.**

## 2.5 Descriptive Statistics

| | count | mean | std | min | 25% | 50% | 75% | max | missing | skewness | kurtosis |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BeerID | 73861.0 | 36931.0 | 21321.978453 | 1.0 | 18466.0 | 36931.0 | 55396.0 | 73861.0 | 0 | 0.000000 | -1.200000 |
| Name | 73859 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2 | NaN | NaN |
| URL | 73861 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 0 | NaN | NaN |
| Style | 73265 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 596 | NaN | NaN |
| StyleID | 73861.0 | 60.179432 | 56.811462 | 1.0 | 10.0 | 35.0 | 111.0 | 176.0 | 0 | 0.656006 | -1.056568 |
| Size(L) | 73861.0 | 43.929775 | 180.373492 | 1.0 | 18.93 | 20.82 | 23.66 | 9200.0 | 0 | 15.873067 | 397.687513 |
| OG | 73861.0 | 1.406266 | 2.196908 | 1.0 | 1.051 | 1.058 | 1.069 | 34.0345 | 0 | 6.707150 | 47.317493 |
| FG | 73861.0 | 1.075865 | 0.432524 | -0.003 | 1.011 | 1.013 | 1.017 | 23.4246 | 0 | 9.787462 | 174.382284 |
| ABV | 73861.0 | 6.136865 | 1.88351 | 0.0 | 5.08 | 5.79 | 6.83 | 54.72 | 0 | 4.951660 | 86.836228 |
| IBU | 73861.0 | 44.276186 | 42.945508 | 0.0 | 23.37 | 35.77 | 56.38 | 3409.3 | 0 | 19.193562 | 1113.459363 |
| Color | 73861.0 | 13.404989 | 11.944511 | 0.0 | 5.17 | 8.44 | 16.79 | 186.0 | 0 | 1.594876 | 2.201058 |
| BoilSize | 73861.0 | 49.724919 | 193.246427 | 1.0 | 20.82 | 27.44 | 30.0 | 9700.0 | 0 | 15.856201 | 402.338392 |
| BoilTime | 73861.0 | 65.07487 | 15.024228 | 0.0 | 60.0 | 60.0 | 60.0 | 240.0 | 0 | 1.379897 | 12.594050 |
| BoilGravity | 70871.0 | 1.353955 | 1.930989 | 0.0 | 1.04 | 1.047 | 1.06 | 52.6 | 2990 | 7.383948 | 67.143919 |
| Efficiency | 73861.0 | 66.354881 | 14.091686 | 0.0 | 65.0 | 70.0 | 75.0 | 100.0 | 0 | -1.479728 | 1.794556 |
| MashThickness | 43997.0 | 2.127235 | 1.682347 | 0.0 | 1.5 | 1.5 | 3.0 | 100.0 | 29864 | 16.851107 | 540.217145 |
| SugarScale | 73861 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 0 | NaN | NaN |
| BrewMethod | 73861 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 0 | NaN | NaN |
| PitchRate | 34609.0 | 0.750468 | 0.394262 | 0.0 | 0.35 | 0.75 | 1.0 | 2.0 | 39252 | 1.046350 | 0.735802 |
| PrimaryTemp | 51199.0 | 19.175641 | 4.219676 | -17.78 | 18.0 | 20.0 | 20.0 | 114.0 | 22662 | 3.257042 | 63.814934 |
| PrimingMethod | 6760 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 67101 | NaN | NaN |
| PrimingAmount | 4774 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 69087 | NaN | NaN |
| UserId | 23371.0 | 43078.069188 | 27734.252556 | 49.0 | 20984.0 | 42897.0 | 57841.0 | 134362.0 | 50490 | 0.559490 | -0.039384 |

**Figure 2.5a: Descriptive Statistics of the dataset.**

The table above summarizes the main descriptive statistics of the dataset prior to any cleaning or filtering. It includes key metrics such as mean, standard deviation, min/max values, number of missing values, skewness, and kurtosis. This statistical overview helped to identify potential issues such as extreme outliers, highly skewed variables, and columns with extensive missingness.

Variables such as OG, FG, and IBU show extremely high skewness and kurtosis, suggesting the need for log-transformations. Columns like MashThickness, PrimaryTemp, and PitchRate contain thousands of missing values and required special imputation strategies or removal. Categorical features like SugarScale and BrewMethod appear to be complete, while PrimingMethod and PrimingAmount are nearly empty and were excluded.

Based on these statistics:

- OG and FG were identified as highly skewed, requiring log- or interaction-based transformations.
- Variables like "PrimingAmount" and "PrimingMethod" were removed due to over 90% missing data.
- Columns with extreme outliers (e.g., IBU and BoilSize) were reviewed with boxplots before modeling.

# 3. Data Exploration

Data exploration is a critical step in this project, as it allows us to understand the characteristics of the beer recipe dataset and to lay the groundwork for predictive ABV modeling. Below, we outline the key techniques and findings derived from the exploratory analysis, using visualizations and descriptive statistics to support cleaning and feature engineering decisions.

## 3.1 Data Exploration Techniques

Exploratory analysis was conducted using univariate and bivariate techniques. These included distribution plots, outlier detection, correlation matrices, and categorical-numerical relationship tests such as ANOVA and Chi-square, helping uncover patterns and inconsistencies that guided feature engineering and model design.

### 3.1.1 Distribution and Outliers

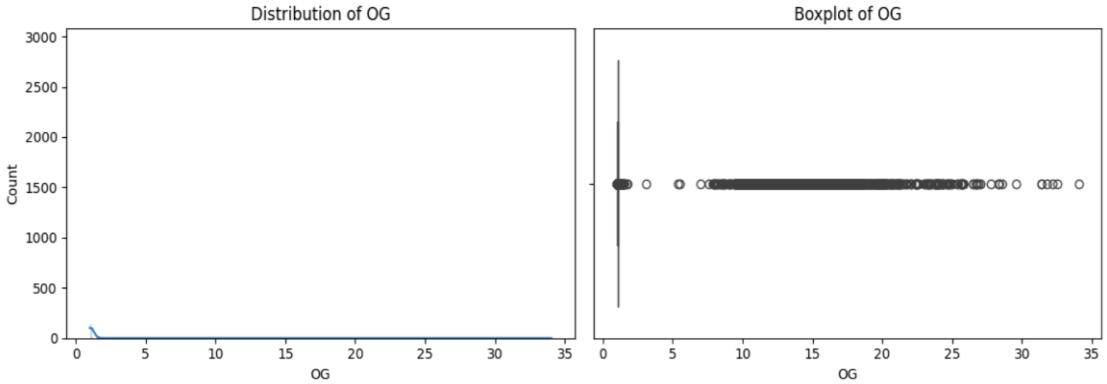


**Figure 3.1.1a: OG Distribution and Boxplot**

OG exhibited a moderately skewed distribution with some extreme outliers on the high end. These values were identified as brewing anomalies or entry errors and later removed in the cleaning process.

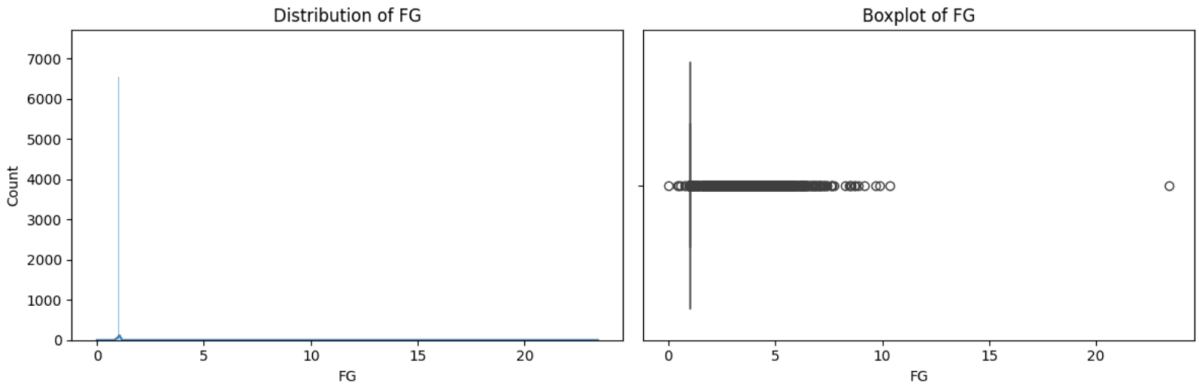


**Figure 3.1.1b: Distribution and Boxplot of FG (Final Gravity)**

The distribution of Final Gravity (FG) reveals a sharp spike near zero, suggesting potential recording or conversion errors (e.g., FG of 0.001 instead of 1.001). The boxplot confirms an extreme right-skew, with a dense core of values below 2 and sporadic extreme outliers beyond 20. These anomalies were treated during the data cleaning phase to preserve the integrity of fermentation measurements

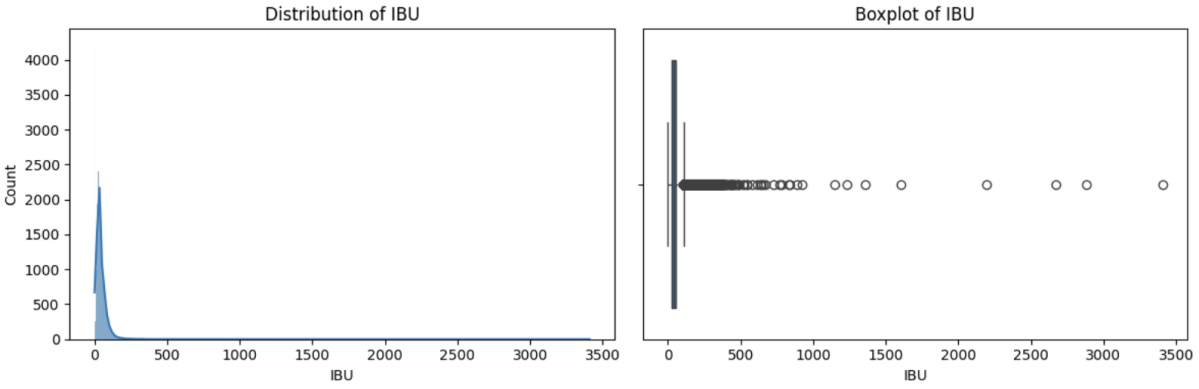


**Figure 3.1.1c: Distribution and Boxplot of IBU (Bitterness)**

The IBU (International Bitterness Units) distribution is heavily right-skewed, with most observations concentrated below 100, which aligns with common brewing practices. However, some extreme values exceeding 3000 likely reflect entry errors or experimental recipes. These were flagged as outliers and filtered to avoid skewing model training.

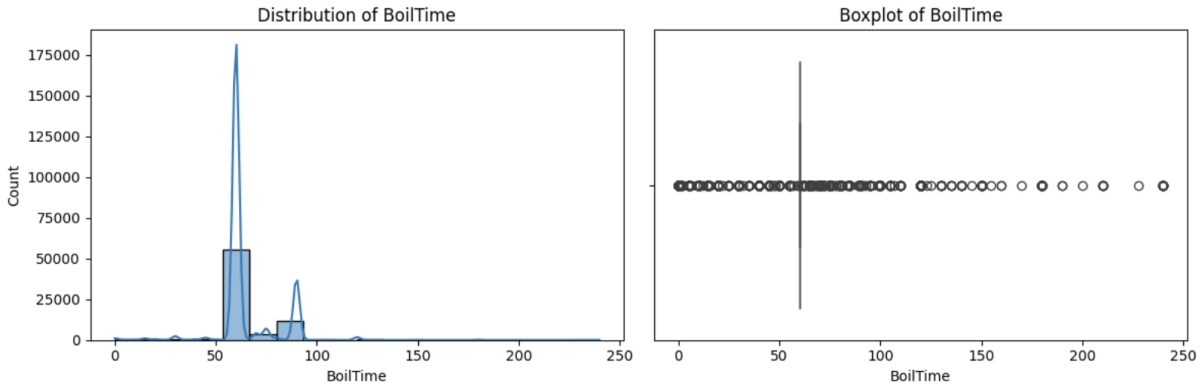


**Figure 3.1.1d: BoilTime Distribution and Boxplot**

Boil times followed a multimodal distribution, likely reflecting differences in brewing techniques (e.g., extract vs. all grain).

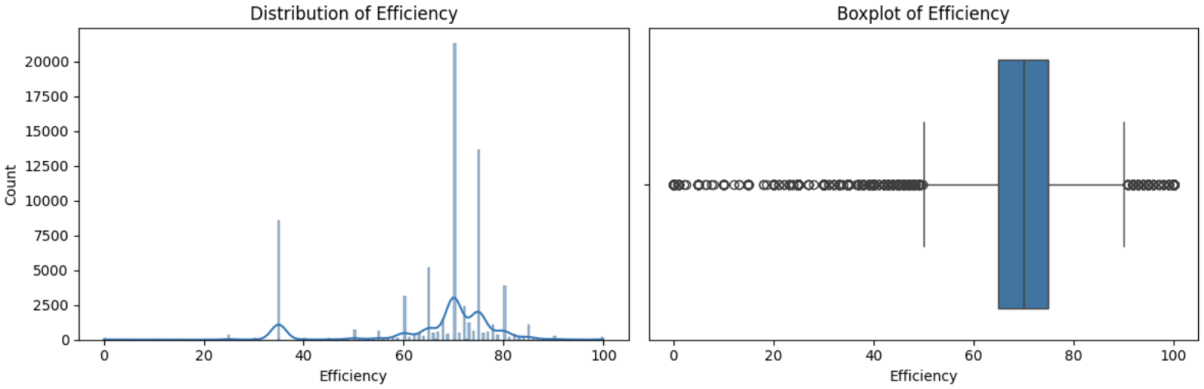


**Figure 3.1.1e: Distribution and Boxplot of Efficiency**

Mash efficiency shows a multimodal distribution with distinct peaks around common values like 65%, 70%, and 75%, possibly reflecting preset defaults in brewing software. The boxplot highlights a concentration within a relatively narrow interquartile range but also reveals several low-efficiency outliers. These were examined closely to distinguish between real low-efficiency brews and data entry inconsistencies.

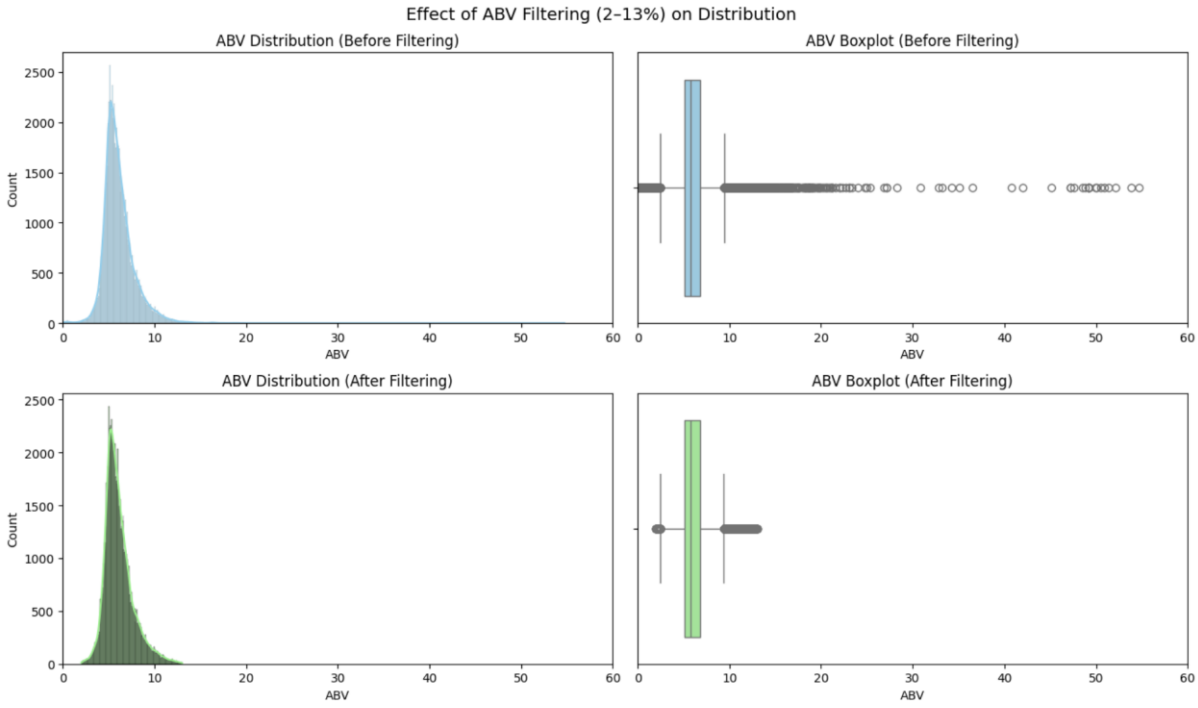


**Figure 3.1.1f: ABV Distribution Before and After Filtering**

The initial distribution of ABV exhibited extreme outliers, with some values exceeding 50%. To focus on standard-strength beer recipes, the dataset was filtered to retain only records with ABV between 2% and 13%. This filtering step improved data consistency without discarding a significant portion of the dataset.

## 3.1.2 Missing values & Cleaning

| Variable Name | Missing Count | Missing % |
|---|---|---|
| PrimingAmount | 69087 | 93.536 % |
| PrimingMethod | 67101 | 90.847 % |
| UserId | 50490 | 68.358 % |
| PitchRate | 39252 | 53.143 % |
| MashThickness | 29864 | 40.432 % |
| PrimaryTemp | 22662 | 30.681 % |
| BoilGravity | 2990 | 4.048 % |
| Style | 596 | 0.806 % |
| Name | 2 | 0.002 % |

**Table 3.1.2a: Missing Value Report**

Most missing values were concentrated in PrimingAmount, PitchRate, and MashThickness. Categorical columns were imputed with mode; numerical ones with the median. Variables with >85% missing were later excluded.

Several variables had notable missingness, particularly Efficiency, PrimaryTemp, and PitchRate. Median or mode imputation was applied based on type and distribution.

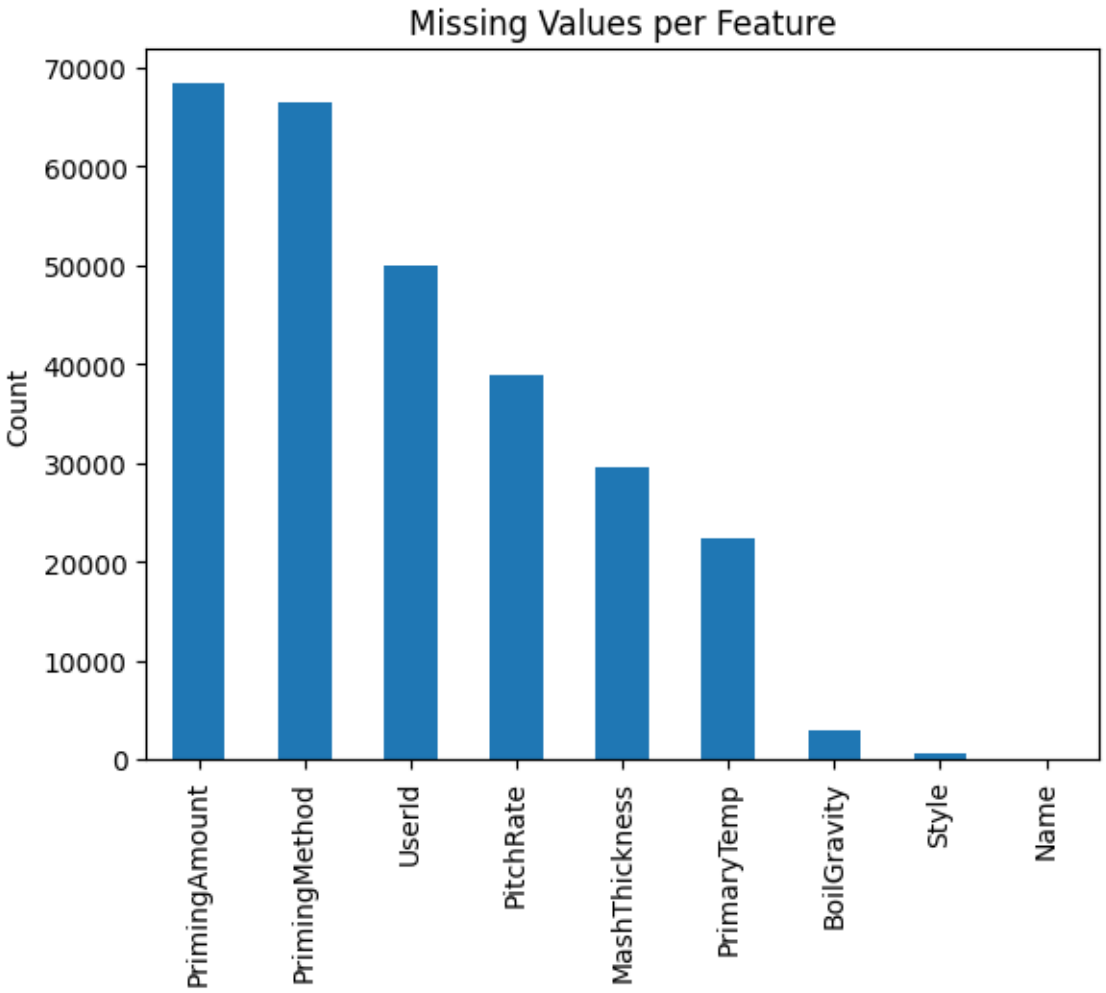


**Figure 3.1.2a: Barplot of Missing Values**

## 3.1.3 Skewness Correction

Logarithmic transformations were applied to highly skewed features such as OG, BoilSize and Size(L) to meet modeling assumptions. Examples of their transformation are shown below.

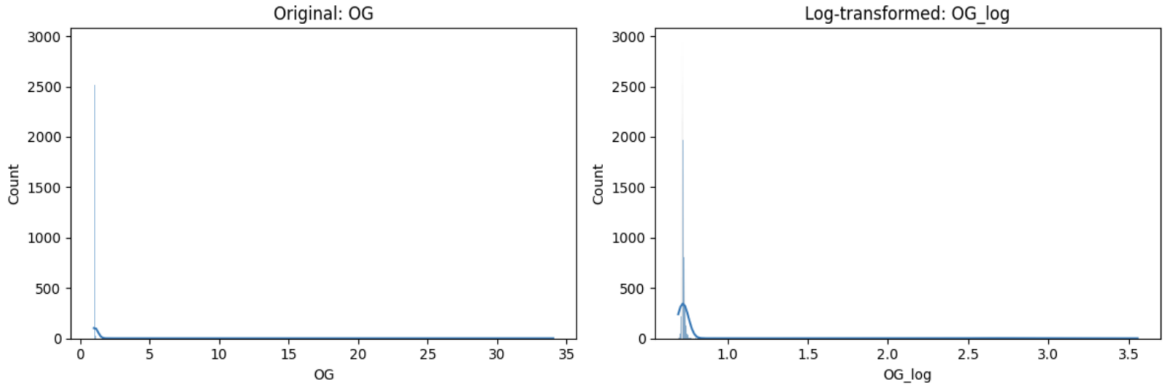

**Figure 3.1.3a: Log Transformation of Skewed Variables**

Efficiency, BoilTime, and Color showed strong right-skewed distributions, their distributions, became more symmetric, improving suitability for linear modeling.

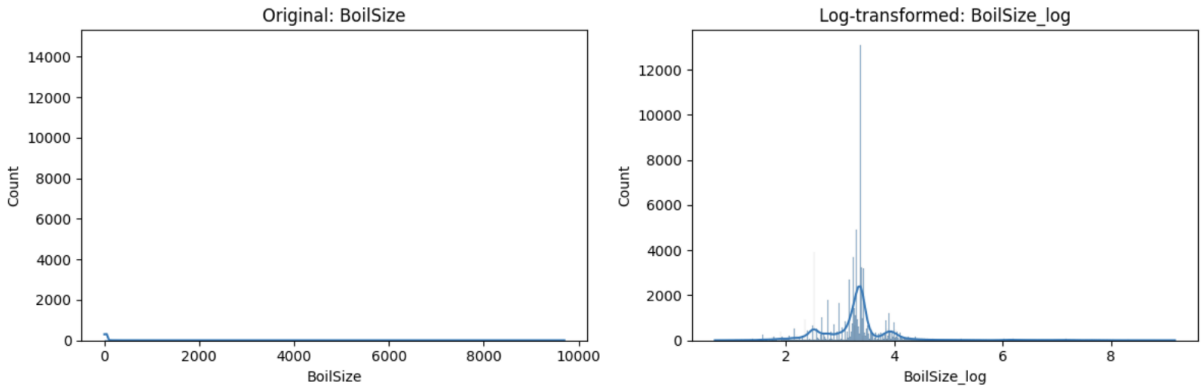


**Figure 3.1.3b: Log Transformation of BoilSize**

The original distribution of BoilSize is extremely right-skewed, with a dense cluster around small values and a long tail extending toward 10,000 liters. This reflects the presence of both homebrew and industrial-scale recipes in the dataset. After applying a log transformation, the distribution becomes more symmetric and centered, making it more suitable for use in regression models that assume normality or homoscedasticity.

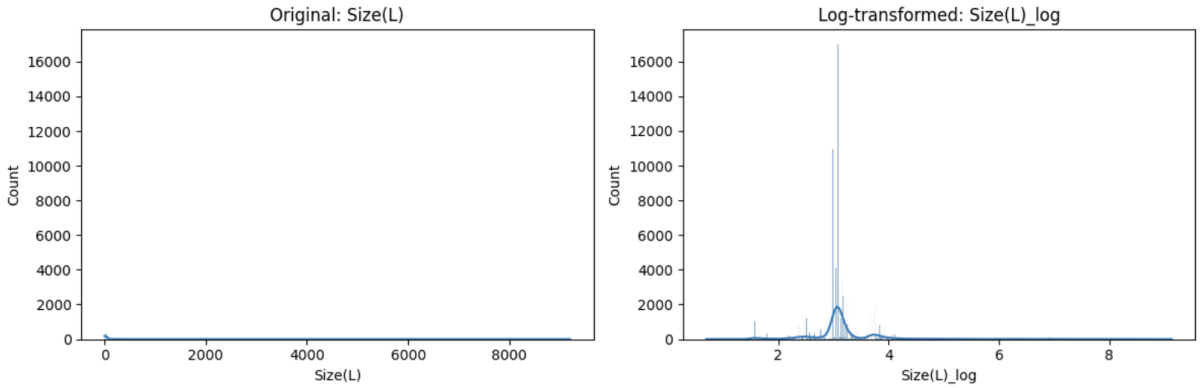


**Figure 3.1.3c: Log Transformation of Size(L)**

The Size(L) variable, which indicates the total batch size, shows similar skewness to BoilSize, with most values below 50 liters but a few extreme cases extending into thousands. The log transformation compresses the range, reduces skewness, and creates a more normalized distribution. This transformation was applied to stabilize variance and reduce the influence of large-volume outliers.

| **Variable** | Skew (Original) | Skew(Log Transformed) |
|---|---|---|
| BoilSize | 15.86 | 2.12 |
| Size(L) | 15.87 | 2.66 |
| OG | 6.71 | 6.07 |

**Table 3.1.3a: Skewness Before and After Log Transformation**

This table confirms that applying a log transformation to highly skewed variables (such as BoilSize and OG) significantly reduced their asymmetry, producing more normally distributed inputs for modeling.

### 3.1.4 Outlier Impact

Filtering ABV between 2% and 13% helped reduce extreme OG values. While OG was not directly cleaned, this filter indirectly removed biologically implausible OG values from the model dataset.

```
df_filtered = df_original[df_original['ABV'].between(2, 13)].copy()
```

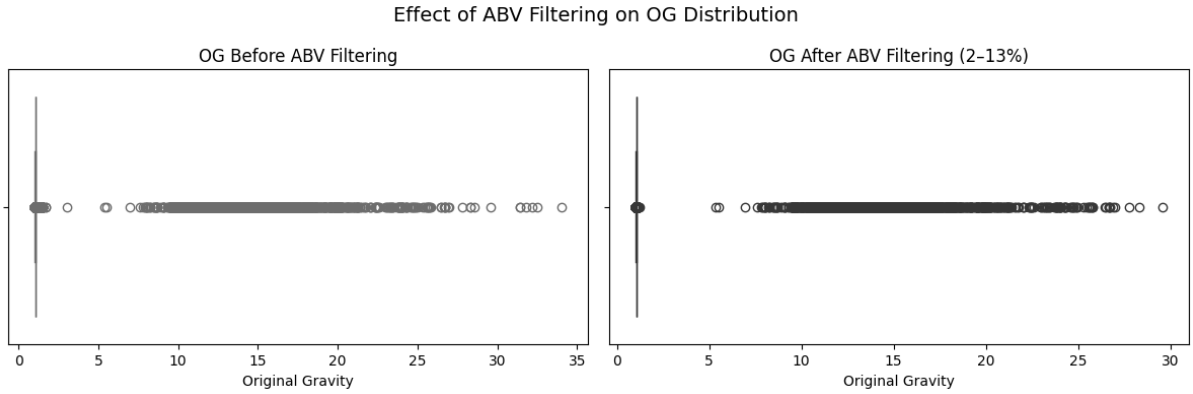


**Figure 3.1.4a: Effect of ABV Filtering on OG Distribution**

This comparison shows the impact of filtering the dataset to include only records with ABV between 2% and 13%. The distribution of Original Gravity (OG) becomes slightly more compact, particularly on the upper end, suggesting that extreme OG values were associated with non-standard or unrealistic ABV outputs. This filtering step enhances the reliability of OG as a predictive feature by aligning it with commercially viable alcohol ranges.

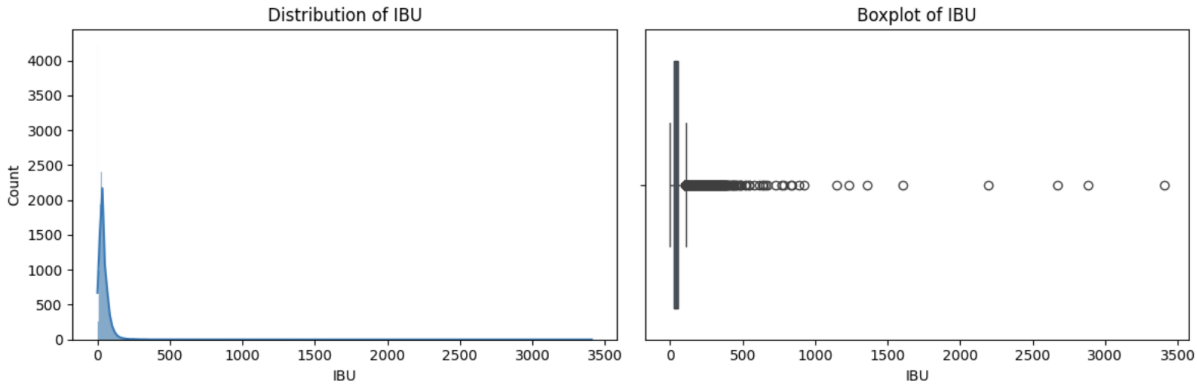


**Figure 3.1.4b: Boxplot of IBU**

IBU displays significant right-skew and high-value outliers, with a small number of recipes exceeding 3000 bitterness units. These were retained for modeling, but their skewness was addressed via log transformation.

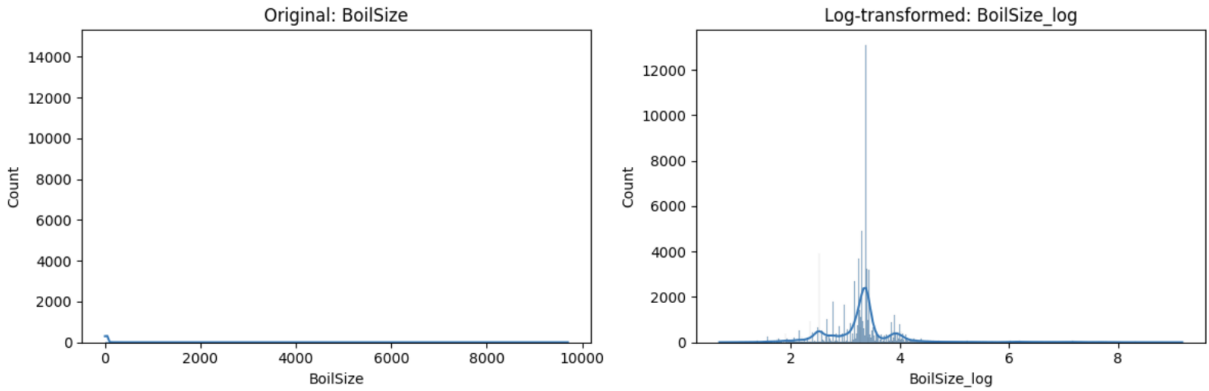


**Figure 3.1.4c: Log-transformed vs. Original BoilSize**

To reduce the impact of extreme values and improve normality, variables with high skewness were log-transformed. This helped stabilize model behavior in the presence of large outliers that were not filtered directly.

While no aggressive outlier removal was applied beyond ABV filtering, the dataset contained significant high-end outliers in several numeric features. Instead of discarding them, the analysis relied on data transformations (log), visual inspection, and robust model choices to mitigate their influence. This approach maintains data integrity while avoiding unnecessary data loss.

## 3.1.5 Correlation Analysis

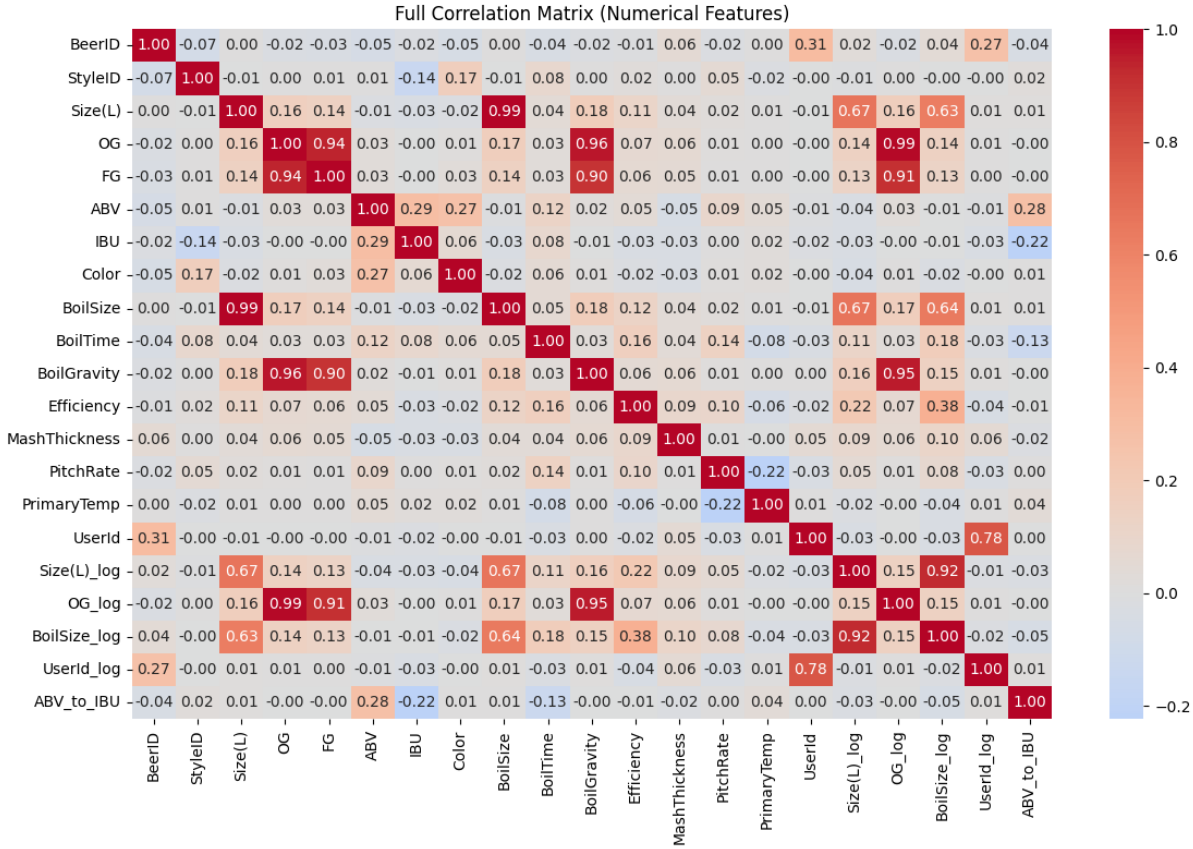


**Figure 3.1.5a: Correlation Matrix of Numerical Features**

The correlation matrix reveals several strong linear relationships across brewing variables. Most notably, OG and FG are highly correlated, which is expected given their shared dependence on fermentation. Other significant correlations include BoilSize and Size(L), while IBU remains relatively independent. This overview informed which variables to retain or engineer for the modeling phase.

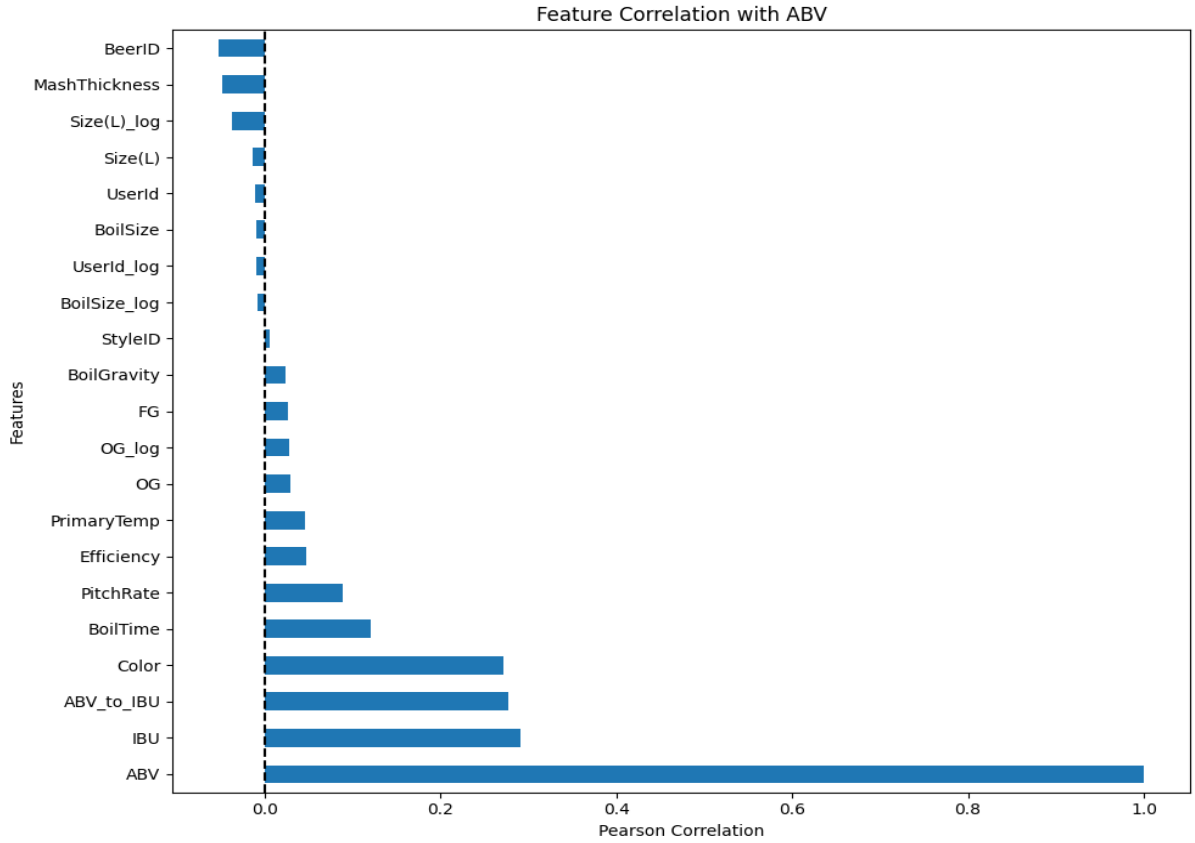


**Figure 3.1.5b: Features Correlation with ABV**

When isolating correlations with the target variable (ABV), OG, FG, and Efficiency show the strongest positive relationships. In contrast, variables like IBU and Color present weaker associations. These insights directly shaped the model design and feature selection strategy.

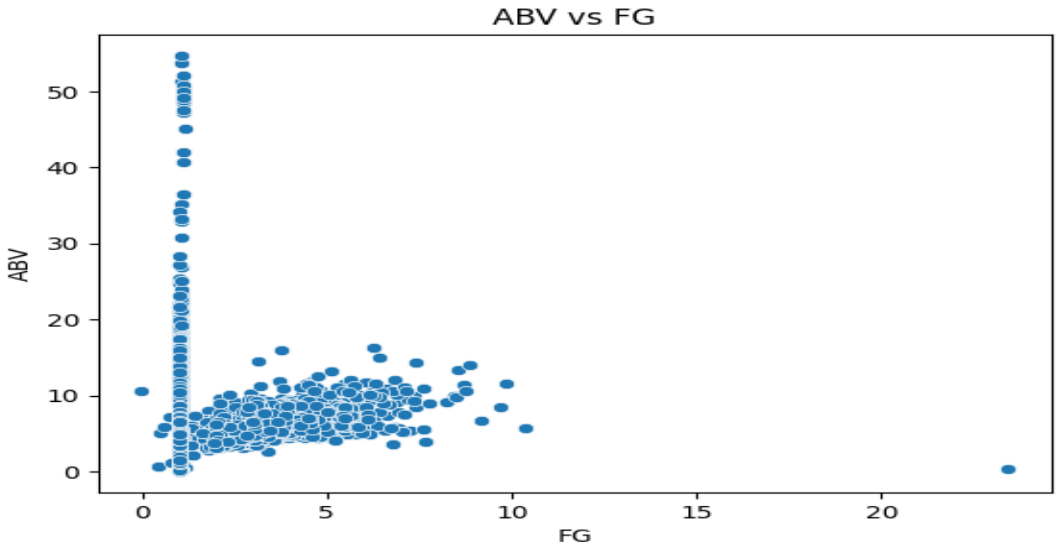


**Figure 3.1.5c: ABV vs Original Gravity (OG)**

The scatter plot reveals a strong positive relationship between Original Gravity (OG) and Alcohol by Volume (ABV). As OG increases, ABV also increases, with a clear linear trend especially in the mid-range. This visual evidence supports the inclusion of OG as a key predictor in the modeling phase.

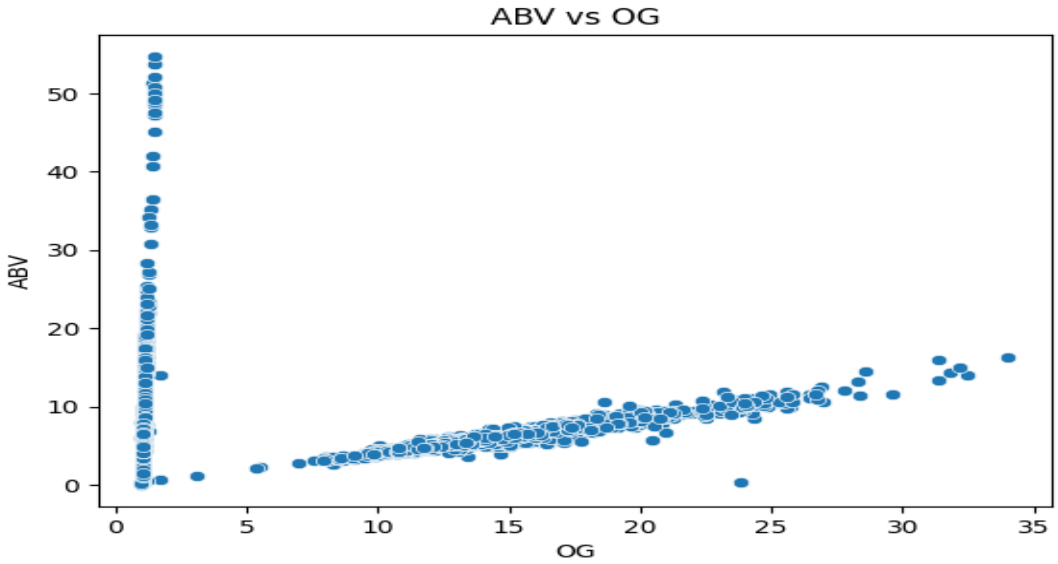


**Figure 3.1.5c: ABV vs Final Gravity (FG)**

The relationship between FG and ABV is less consistent. While a weak negative trend is expected (since more fermentation lowers FG and raises ABV), the presence of extreme low and high FG values introduces noise. This irregularity underscores the need for robust model design when using FG.

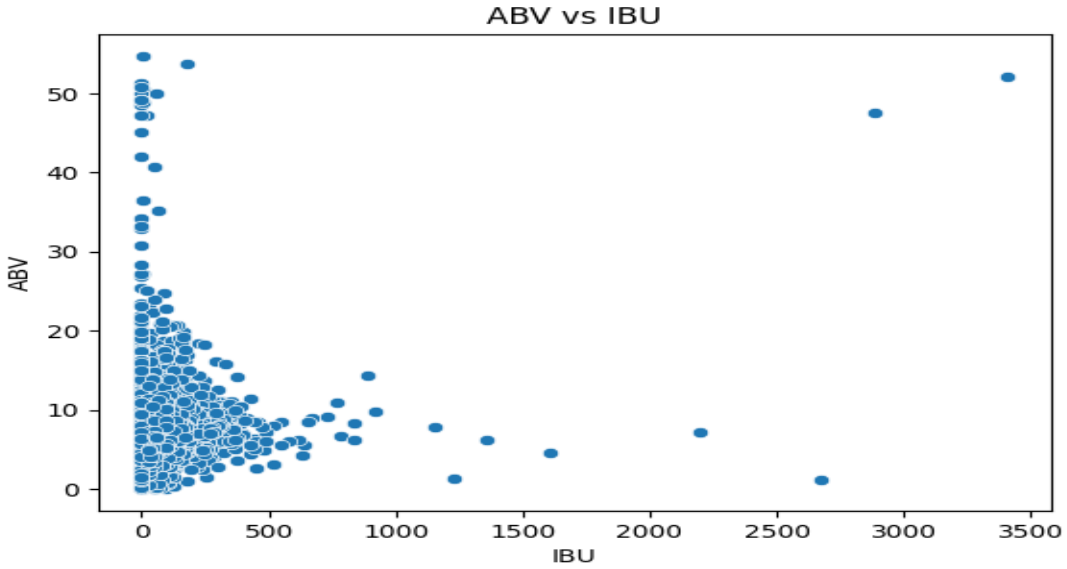


**Figure 3.1.5d: ABV vs IBU**

IBU (bitterness) shows no clear linear association with ABV. The scatter is dense at low IBU levels, with a long tail of high outliers. This visual confirms earlier findings that IBU has limited predictive power for alcohol content and should not be heavily weighted in the model.

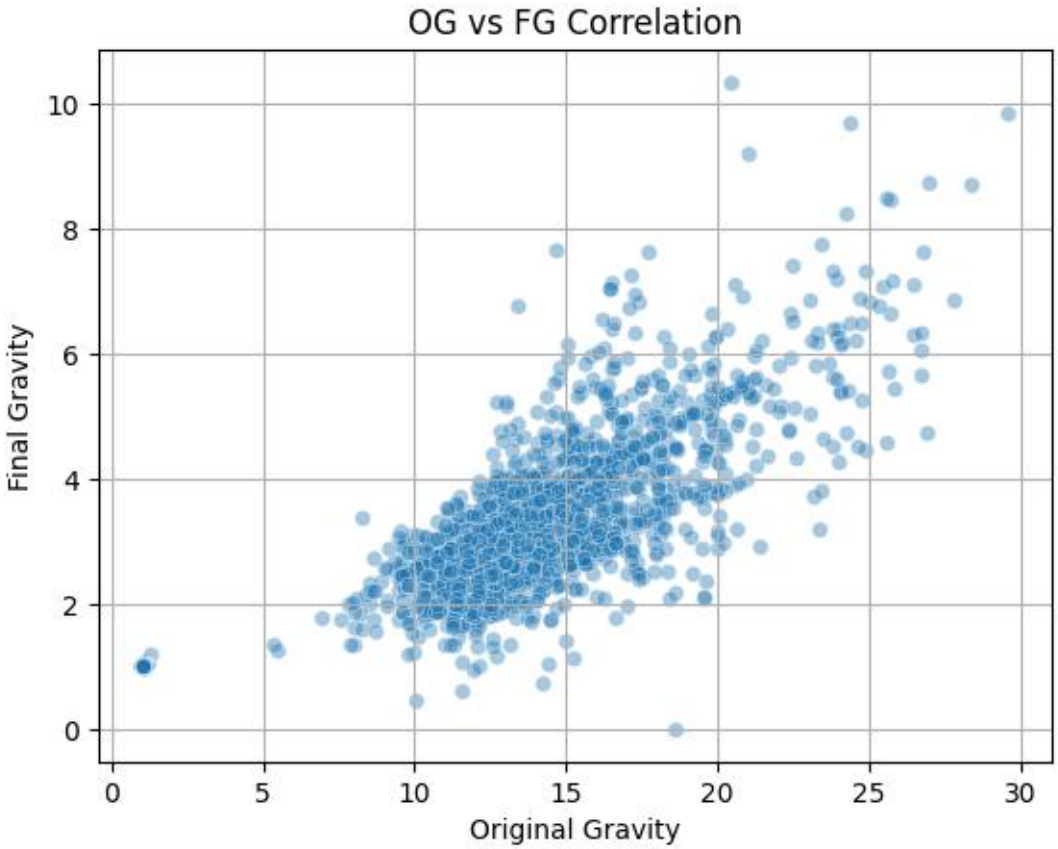


**Figure 3.1.5e: Correlation Between OG and FG**

This scatter plot reveals a strong positive correlation between Original and Final Gravity, indicating that fermentation behavior is consistent across recipes. Despite their correlation, both features were retained due to their complementary influence on ABV.

### 3.1.6 Categorical Feature Analysis

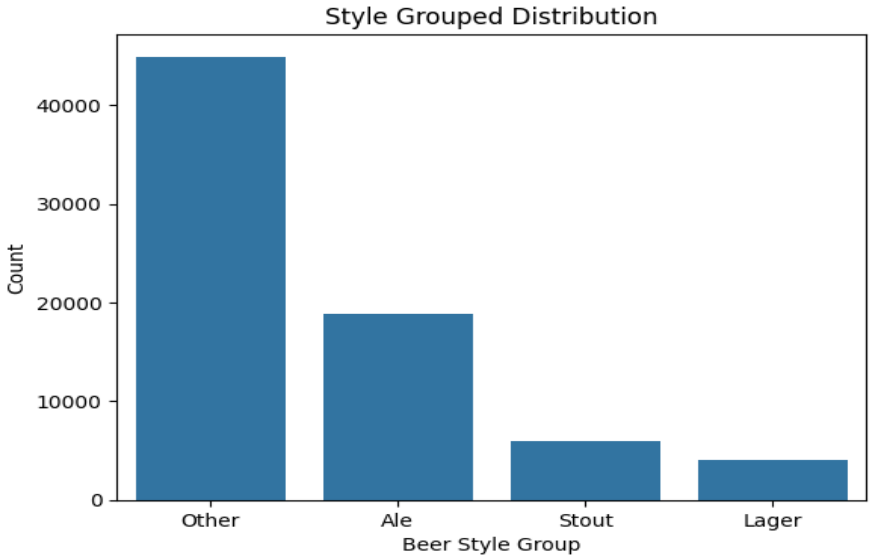


**Figure 3.1.6a: Style Grouped Distribution**

The dataset was categorized into four main style groups: Ale, Lager, Stout, and Other. The "Other" category accounts for more than half of the records, reflecting a wide diversity of unique or less common styles. Ales are also well-represented, while Lagers and Stouts appear less frequently.

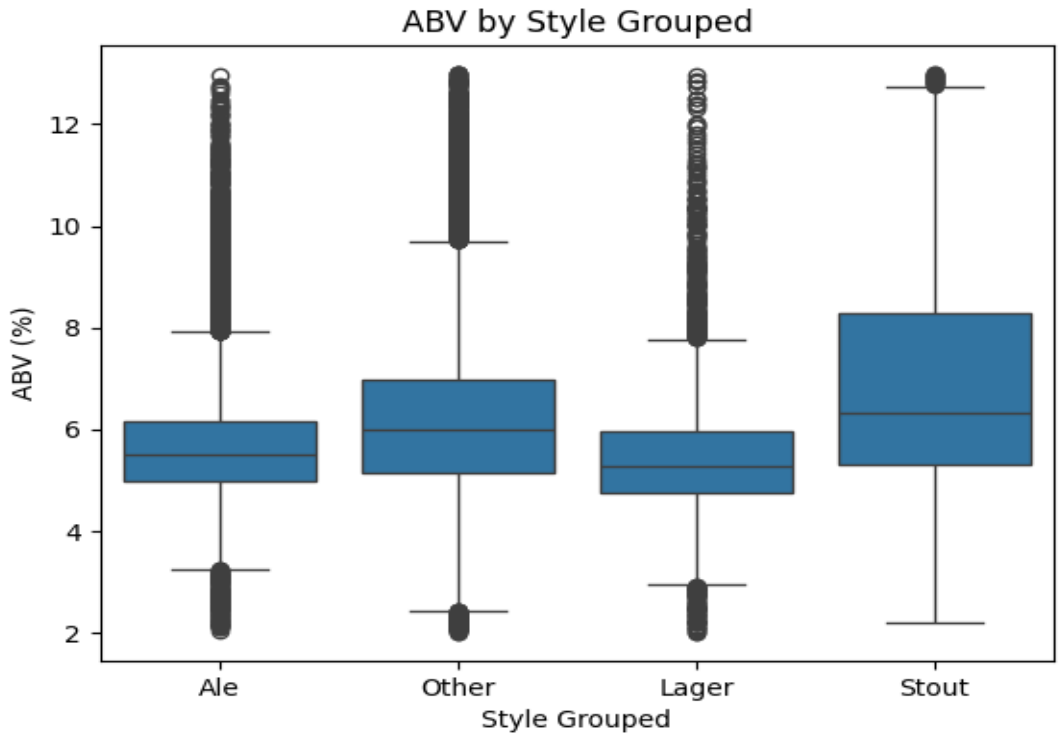


**Figure 3.1.6b: ABV by Style Grouped - Filtered**

This boxplot illustrates the distribution of ABV across grouped beer styles, after filtering values to the 2%–13% range. Stouts exhibit the highest median and most dispersed alcohol levels, making them distinctive among style groups. Lagers show lower ABV concentrations, while Ales and Others fall in between. This pattern supports the inclusion of Style_Grouped as a meaningful categorical feature in the modeling pipeline.

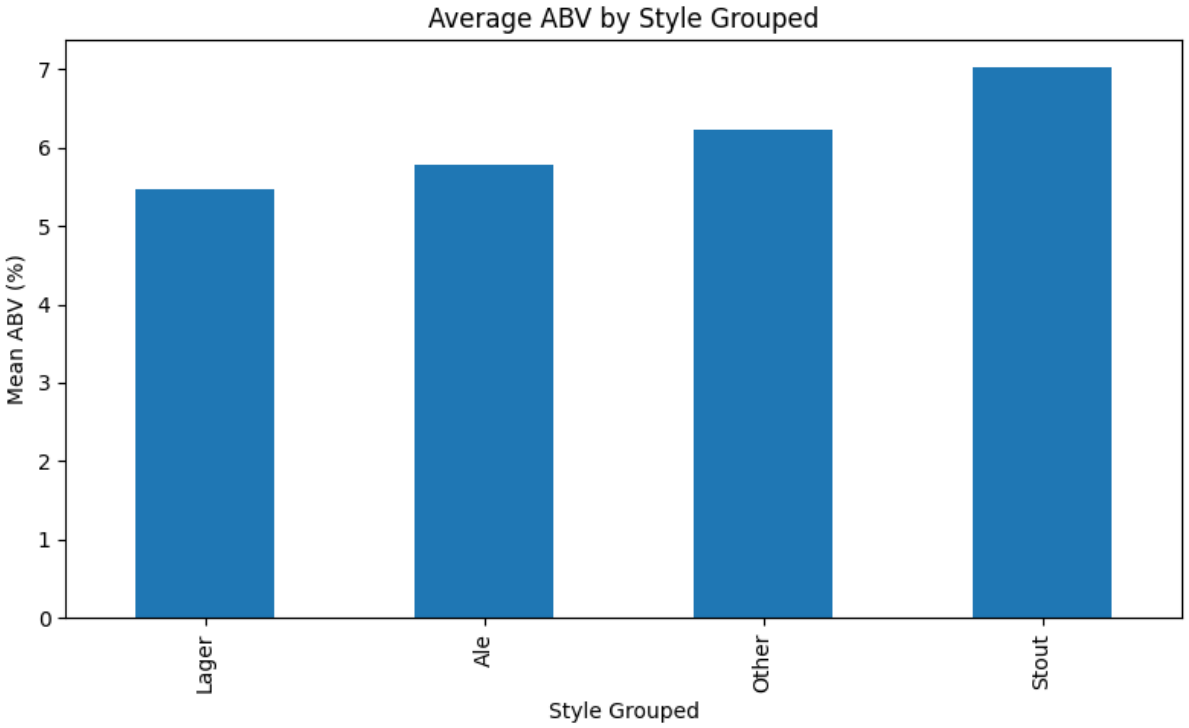


**Figure 3.1.6c: Average ABV by Style Grouped**

The average ABV for each style group reinforces the pattern observed in the boxplot. Stouts lead with an average ABV close to 6.5%, while Lagers average under 5%. This quantitative insight validates the role of style in explaining alcohol content variance and justifies its use in feature engineering.

## 3.2 Dataset Cleaning

The cleaning process combined exploratory observations with structured preprocessing steps, resulting in a refined dataset suitable for machine learning modeling. Several cleaning tasks were carried out as follows.

### 3.2.1 Irrelevant Columns Removed

Non-predictive or user-specific columns such as BeerID, Name, UserId, URL, and the raw forms of PrimingMethod and PrimingAmount were removed from the dataset. These variables introduced noise, were unstructured, or contained excessive missing values without adding meaningful information for ABV prediction.

```
# 3. Drop original columns no longer needed
df.drop(['BeerID','Name','UserId','URL','Style','PrimingMethod','PrimingAmount'], axis=1, inplace=True, errors='ignore')
```

**Figure 3.2.1a: Columns Removed During Initial Cleaning Step**

## 3.2.2 Filtering of Implausible Targets

The target variable ABV was filtered to include only values between 2% and 13%, aligning the dataset with the typical alcohol content range of standard-strength beers. This eliminated extreme outliers and out-of-scope recipes.

```
# 1. Load data and filter realistic ABV range
df = pd.read_csv('recipeData.csv', encoding='ISO-8859-1')
df = df[df['ABV'].between(2, 13)].copy()
```

**Figure 3.2.2a: Filtering records with ABV between 2% and 13%**

## 3.2.3 Missing Value Treatment

- Numerical columns were imputed using the median, a robust method that minimizes the impact of outliers.

```
# Numeric columns: fill missing with median
for c in num_cols:
    df[c].fillna(df[c].median(), inplace=True)
```

**Figure 3.2.3a: Imputing missing numeric values with median**

- Categorical variables were filled with the string "Missing", then rare categories representing <2% of observations were grouped into "Other" to reduce dimensionality and prevent overfitting during encoding.

```
# Categorical columns: fill missing, then group rare categories into 'Other'
for c in cat_cols:
    df[c] = df[c].fillna('Missing').astype(str)
    freq = df[c].value_counts(normalize=True)
    rare_vals = freq[freq < 0.02].index
    df[c].replace(list(rare_vals), 'Other', inplace=True)
```

**Figure 3.2.3b: Imputing missing categorical values and grouping rare levels**

## 3.2.4 Text Standardization and Grouping

Free-text fields such as Style, PrimingMethod, and PrimingAmount were normalized into grouped categories using rule-based string matching. This included creating new features:

- Style_Grouped: Groups like Ale, Lager, Stout, IPA, Wheat, etc.

```
def group_styles(s):
    if pd.isnull(s):
        return 'Other'
    s = s.lower()
    if 'ipa' in s:        return 'IPA'
    if 'pale ale' in s:   return 'Pale Ale'
    if 'stout' in s:      return 'Stout'
    if 'lager' in s:      return 'Lager'
    if 'porter' in s:     return 'Porter'
    if 'wheat' in s or 'weiss' in s: return 'Wheat'
    return 'Other'
```

**Figure 3.2.4a: Function to group beer styles into broader categories**

- PrimingMethod_grouped: Categories like Sugar, Honey/Syrup, Malt Extract, etc.

```
def group_priming_method(x):
    if pd.isnull(x):
        return 'Missing'
    s = str(x).lower()
    if any(k in s for k in ['sugar','azúcar','cane','dextrose','glucose']):
        return 'Sugar'
    if any(k in s for k in ['malt extract','dme','lme','wort']):
        return 'Malt Extract'
    if any(k in s for k in ['honey','miel','syrup']):
        return 'Honey/Syrup'
    if any(k in s for k in ['krausen','gyle']):
        return 'Krausening'
    if any(k in s for k in ['keg','co2','forced']):
        return 'Forced Carb'
    if any(k in s for k in ['tablet','drop']):
        return 'Tablet/Drop'
    return 'Other'
```

**Figure 3.2.4b: Function to group priming methods**

- PrimingAmount_grouped: Units like Grams, Ounces, Volume, etc.

```python
def group_priming_amount(x):
    if pd.isnull(x):
        return 'Missing'
    s = str(x).lower()
    if any(u in s for u in ['gallon','gal']):
        return 'Volume'
    if 'oz' in s:
        return 'Ounces'
    if 'g' in s and 'gal' not in s:
        return 'Grams'
    if any(u in s for u in ['cup','tbsp','tsp']):
        return 'Cups/Spoons'
    if any(u in s for u in ['day','hour','min']):
        return 'Time'
    if any(u.isdigit() for u in s):
        return 'Numeric'
    return 'Other'
```

**Figure 3.2.4c: Function to group priming amount**

After the cleaning and grouping stages, the dataset was reshaped for modeling. Numerical features were normalized or log-transformed where appropriate. Categorical variables were one-hot encoded using low-cardinality constraints, and engineered features were added to enhance signal for ABV prediction.

```python
# Apply grouping
df['Style_Grouped'] = df['Style'].apply(group_styles)
df['PrimingMethod_grouped'] = df['PrimingMethod'].apply(group_priming_method)
df['PrimingAmount_grouped'] = df['PrimingAmount'].apply(group_priming_amount)
```

**Figure 3.2.4d: Applying grouping logic to categorical columns**

## 3.2.5 Feature Engineering Foundations

Some transformations were introduced early during the cleaning phase to enhance feature quality and capture domain knowledge. These included:

- OG_minus_FG: Difference between Original and Final Gravity
- IBU_to_OG, IBU_to_FG: Bitterness ratio indicators
- Interactions such as OGxEfficiency and OGxIBU

```
# 6. Feature engineering
df['OG_minus_FG'] = df['OG'] - df['FG']
df['IBU_to_OG']   = df['IBU'] / df['OG']
df['IBU_to_FG']   = df['IBU'] / df['FG']


# Feature Interactions
df['OGxIBU']             = df['OG_minus_FG'] * df['IBU']
df['OGxEfficiency']      = df['OG_minus_FG'] * df['Efficiency']
df['IBU_to_OG_x_OG_minus_FG'] = df['IBU_to_OG'] * df['OG_minus_FG']
```

**Figure 3.2.5a: Feature engineering and interaction terms creation**

# 4. Modeling Setup

## 4.1 Objective & Strategy

The main objective of this modeling section is to accurately predict the Alcohol By Volume (ABV) of a beer recipe based on its technical and categorical attributes. This task is highly relevant both from a business and production standpoint. In the brewing industry, knowing the ABV in advance is essential for quality assurance, tax calculations, and compliance with labeling regulations (CFIA, 2024). It also influences customer satisfaction, as consumers expect the advertised alcohol content to be accurate. For both home brewers and industrial producers, having an early estimate of the ABV allows for adjusting ingredients, improving consistency, and minimizing rework later in production.

From a data science perspective, ABV serves as a well-defined continuous target variable. It can be modeled using a variety of regression-based algorithms, allowing the exploration of both linear relationships and more complex, non-linear interactions. Furthermore, the dataset contains a mix of numeric and categorical inputs, making it an excellent candidate to test diverse preprocessing strategies and evaluate model generalization.

To address the ABV prediction challenge, a hybrid modeling strategy was adopted. The approach blends both interpretable and high-performance models to derive insights and predictive power:

- Decision Trees were selected as a starting point due to their simplicity and visual interpretability. They help uncover initial structure in the data and support intuition-building about variable splits and thresholds.
- Regularized Linear Regression (RidgeCV) serves as a baseline for assessing the strength of global linear relationships. It also helps identify top drivers in a quantifiable and replicable manner.
- Statistical selection techniques such as Backward and Forward Stepwise Regression were used to reduce complexity and enhance interpretability by

filtering out noise or redundant variables based on significance and AIC criteria.

- Ensemble models, including Random Forests, were introduced to capture interactions and non-linear patterns that simpler models may miss.
- Finally, Neural Networks (both Scikit-Learn's MLP and a custom Keras implementation) were tested to evaluate the gains of deeper, non-linear representation learning, with a critical eye on their trade-offs in interpretability and tuning complexity.

This multi-model perspective allows for rigorous comparison and triangulation of insights. By combining statistical theory with empirical results, the strategy remains grounded and aligned with the practical goal of explaining and predicting ABV — not just optimizing metrics in isolation.

## 4.2 Data Preprocessing

Before engaging in any modeling, the raw dataset underwent a structured and critical preprocessing pipeline designed to ensure data quality, enhance interpretability, and support robust model generalization.

Filtering for Valid ABV range the dataset was initially filtered to retain only recipes with ABV values between 2% and 13%. This range reflects realistic alcohol concentrations found in commercial and homebrewed beers. Entries with extremely low or high ABV values were considered likely to be input errors, non-beer experiments, or unrepresentative outliers. Removing them reduces the risk of skewed model performance and improves the trustworthiness of the predictions.

```
# Load data and filter realistic ABV range
df = pd.read_csv('recipeData.csv', encoding='ISO-8859-1')
df = df[df['ABV'].between(2, 13)].copy()
```

**Figure 4.2a: This filtering step ensured only plausible beer recipes remained in the modeling dataset.**

Categorical Feature Grouping, several raw features contained high-cardinality or loosely structured string values. To streamline modeling and avoid excessive one-hot encoding:

- Beer styles were grouped into interpretable classes such as Lager, IPA, Stout, Pale Ale, Porter, Wheat, and Other.
- Priming methods (e.g., "cane sugar", "malt extract", "forced carbonation") and priming amount descriptions (e.g., "5 oz", "1 cup", "2 days") were consolidated into broader, meaningful categories via custom string-based grouping functions.

```
# Grouping functions for categorical variables
def group_styles(s):
    if pd.isnull(s):
        return 'Other'
    s = s.lower()
    if 'ipa' in s:        return 'IPA'
    if 'pale ale' in s:   return 'Pale Ale'
    if 'stout' in s:      return 'Stout'
    if 'lager' in s:      return 'Lager'
    if 'porter' in s:     return 'Porter'
    if 'wheat' in s or 'weiss' in s: return 'Wheat'
    return 'Other'
```

**Figure 4.2b: A set of grouping functions were designed to reduce cardinality and standardize categorical features. For example, all variants of India Pale Ales were grouped under "IPA".**

This grouping approach preserved relevant information while reducing noise, redundancy, and dimensionality.

Missing Value Treatment is a thoughtful imputation strategy was employed:

- Numeric features were imputed using the median, a robust measure especially suitable for skewed distributions.
- Categorical variables were imputed using a placeholder value "Missing." In addition, rare categories (occurring in fewer than 2% of entries) were grouped under "Other" to prevent overfitting and reduce feature explosion.

```
# Numeric columns: fill missing with median
for c in num_cols:
    df[c].fillna(df[c].median(), inplace=True)

# Categorical columns: fill missing, then group rare categories into 'Other'
for c in cat_cols:
    df[c] = df[c].fillna('Missing').astype(str)
    freq = df[c].value_counts(normalize=True)
    rare_vals = freq[freq < 0.02].index
    df[c].replace(list(rare_vals), 'Other', inplace=True)
```

**Figure 4.2c: Missing numeric values were imputed using the median, while categorical values were filled and regrouped for robustness.**

This ensured compatibility across all modeling algorithms, particularly tree-based models which do not handle missing values natively.

Feature Engineering to improve predictive power and capture latent domain logic, several new features were engineered:

- OG_minus_FG: captures the difference between original and final gravity, a known proxy for alcohol yield.

- IBU_to_OG and IBU_to_FG: ratio variables reflecting bitterness intensity relative to fermentation potential.
- Interaction terms such as OGxIBU, OGxEfficiency, and IBU_to_OG_x_OG_minus_FG were included to model compound effects, informed by brewing chemistry and exploratory data patterns.

```
# Feature engineering
df['OG_minus_FG'] = df['OG'] - df['FG']
df['IBU_to_OG']   = df['IBU'] / df['OG']
df['IBU_to_FG']   = df['IBU'] / df['FG']

# Feature Interactions
df['OGxIBU']              = df['OG_minus_FG'] * df['IBU']
df['OGxEfficiency']       = df['OG_minus_FG'] * df['Efficiency']
df['IBU_to_OG_x_OG_minus_FG'] = df['IBU_to_OG'] * df['OG_minus_FG']
```

**Figure 4.2d: These derived variables were informed by brewing chemistry and designed to capture deeper interactions.**

Pipeline Construction for A full preprocessing pipeline was assembled using Scikit-Learn's Pipeline and ColumnTransformer objects to ensure reproducibility and clean separation of transformation logic. The pipeline included:

- PowerTransformer (Yeo-Johnson): to normalize skewed numerical distributions.
- RobustScaler: to mitigate the influence of outliers.
- OneHotEncoder: for categorical variables, with drop='first' to avoid dummy variable traps.
- SelectKBest: to select the top 30 features based on univariate F-regression.
- A default RidgeCV regressor for early benchmarking and consistent pipeline fitting.

```
# Complete pipeline: preprocessing + feature selection + modeling
# Scaling
preprocessor = ColumnTransformer([
    ('num', Pipeline([
        ('power', PowerTransformer(method='yeo-johnson')),
        ('scale', RobustScaler())
    ]), num_cols + ['OG_minus_FG','IBU_to_OG','IBU_to_FG']),
    ('cat', OneHotEncoder(drop='first', sparse_output=False, handle_unknown='ignore'), cat_cols)
])

pipeline = Pipeline([
    ('pre', preprocessor),
    ('select', SelectKBest(score_func=f_regression, k=30)),
    ('model', RidgeCV(alphas=[0.01,0.1,1,10], cv=5))
])
```

**Figure 4.2e: A modular Scikit-Learn pipeline was assembled to ensure consistency and replicability across models and cross-validation folds.**

Train-Test Split all models were trained and evaluated using a 60/40 train-test split, allocating 60% of the data for model training and reserving 40% for unbiased performance evaluation. The split used a fixed random seed (random_state=42) to ensure reproducibility across runs and model types.

```
# Pipeline split
X_pipe = df.drop(columns='ABV')
y_pipe = df['ABV']

X_train_pipe, X_test_pipe, y_train_pipe, y_test_pipe = train_test_split(X_pipe, y_pipe, test_size=0.4, random_state=42)
pipeline.fit(X_train_pipe, y_train_pipe)
```

**Figure 4.2f: A 60/40 train-test split was applied with fixed random seed to allow reproducibility.**

## 4.3 Tree-Based Modeling

This section explores decision tree algorithms to model ABV, comparing depth, input variables, and pruning techniques. The goal is to balance interpretability and predictive accuracy, while identifying the most relevant features driving alcohol content.

### 4.3.1 Maximal and Pruned Regression Trees

This section focuses on building regression trees to predict the Alcohol By Volume (ABV) using all available numerical and categorical variables (transformed through One-Hot Encoding). Two modeling strategies are explored:

- A maximal tree (fully grown without pruning)
- A pruned tree using Cost Complexity Pruning (ASE)

Both are evaluated in terms of predictive power and model complexity.

```
# Split and preprocess
y_reg = df['ABV']
X = df.drop(columns='ABV')
X_train, X_test, y_train, y_test = train_test_split(X, y_reg, test_size=0.4, random_state=42)

# Preprocessing
pre = pipeline.named_steps['pre']
pre.fit(X_train)
Xtr = pre.transform(X_train)
Xte = pre.transform(X_test)
```

**Figure 4.3.1a: Data Preparation & Preprocessing.**

```
# Maximal - unpruned - regression tree
tree_max = DecisionTreeRegressor(random_state=42)
tree_max.fit(Xtr, y_train)
y_pred_max = tree_max.predict(Xte)

r2_max  = r2_score(y_test, y_pred_max)
mse_max = mean_squared_error(y_test, y_pred_max)
leaves_max = tree_max.get_n_leaves()
depth_max  = tree_max.get_depth()
```

**Figure 4.3.1b: The full tree is trained with no depth limitation and no regularization**

Results: Maximal tree → $R^2 = 0.991$, MSE = 0.021, Leaves = 37630, Depth = 50

While the model achieves a nearly perfect fit, its extremely large number of leaves and depth indicate a **high risk of overfitting**. Such a complex structure suggests the model is memorizing noise rather than generalizing patterns in the data.

The model was retrained using a cross-validated ccp_alpha to control overfitting by pruning branches that contribute little to reducing error.

```
# ASE-pruned tree (cost-complexity pruning)
path = tree_max.cost_complexity_pruning_path(Xtr, y_train)
alphas = path.ccp_alphas

# Sample 20 alphas equally spaced
sampled_alphas = np.linspace(alphas.min(), alphas.max(), 20)
cv_mses = []
for alpha in sampled_alphas:
    t = DecisionTreeRegressor(random_state=42, ccp_alpha=alpha)
    scores = cross_val_score(t, Xtr, y_train,
                             scoring='neg_mean_squared_error', cv=3)
    cv_mses.append(-scores.mean())

best_alpha = sampled_alphas[np.argmin(cv_mses)]
print(f"Chosen ccp_alpha = {best_alpha:.5f} (CV MSE = {min(cv_mses):.3f})")
```

**Figure 4.3.1c: Pruned Tree (ASE – Cost Complexity Pruning)**

```
tree_pruned = DecisionTreeRegressor(random_state=42, ccp_alpha=best_alpha)
tree_pruned.fit(Xtr, y_train)
y_pred_pruned = tree_pruned.predict(Xte)

r2_pruned  = r2_score(y_test, y_pred_pruned)
mse_pruned = mean_squared_error(y_test, y_pred_pruned)
leaves_pruned = tree_pruned.get_n_leaves()
depth_pruned  = tree_pruned.get_depth()
```

**Figure 4.3.1d: Pruned Tree Code**

Despite performing pruning with cross-validation, the model remained unchanged. This is likely because the chosen ccp_alpha was 0.00000, meaning no branches were removed. This outcome suggests that more aggressive pruning strategies may be needed to simplify the model.

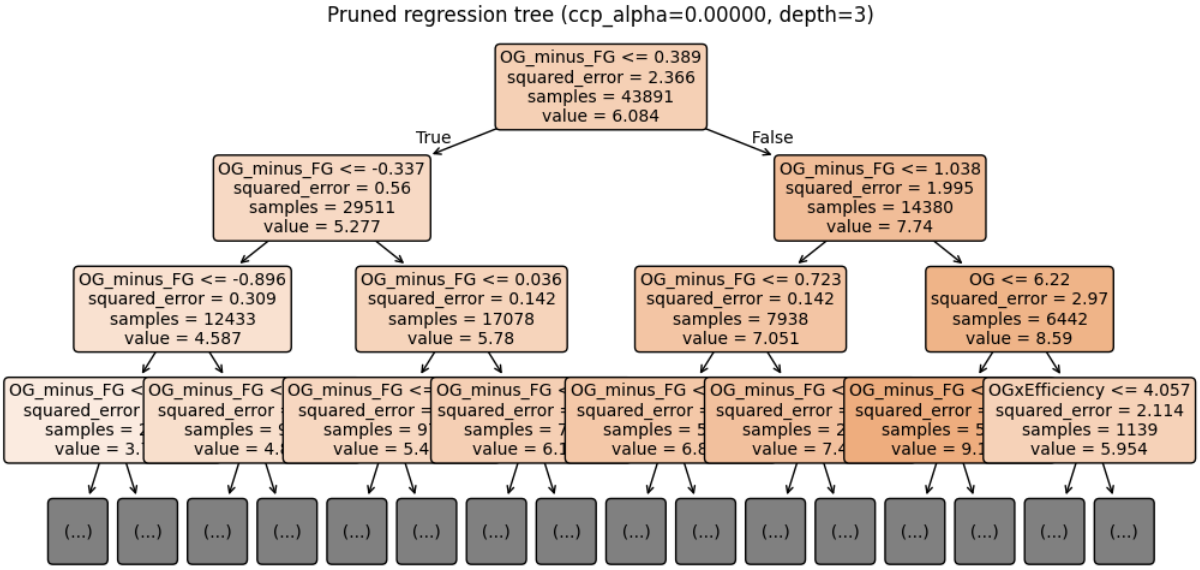


**Figure 4.3.1e: Tree Visualization (First 3 Levels)**

The tree structure shows early splits on variables like OG_minus_FG, IBU_to_OG, and Efficiency, supporting their relevance in predicting ABV.

## 4.3.2 Shallow Tree with Binned OG

This approach investigates whether a simplified version of the regression tree—trained only on a binned transformation of the variable OG_minus_FG—can effectively predict the ABV. This is especially useful from a business perspective when we need a **quick, interpretable model** that can be easily explained to non-technical stakeholders, like brewers or operational managers.

The variable OG_minus_FG, which reflects the difference between original and final gravity (and correlates strongly with alcohol production), was **binned into three categories**: Low, Medium, High.

```
t1, t2 = 0.389, 1.038
df['OG_bin'] = pd.cut(
    df['OG_minus_FG'],
    bins=[-np.inf, t1, t2, np.inf],
    labels=['Low','Medium','High']
)
```

**Figure 4.3.2a: Newly created categorical variable OG_bin**

**The dataset was split into training and test sets (60/40). Only the binned version of gravity and other relevant variables were passed into the pipeline.**

```
y = df['ABV']
X = df.drop(columns=['ABV', 'OG', 'OG_minus_FG'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)


# Only categories
cat_cols = ['OG_bin'] + [c for c in X_train.select_dtypes(include='object').columns
                         if c != 'OG_bin']
# Only numerical
num_cols = [c for c in X_train.select_dtypes(include='number').columns
            if c not in ('OG_minus_FG','OG')]

# Pipeline: one-hot for OG_bin + passthrough
pre = ColumnTransformer([
    ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'), cat_cols)
], remainder='passthrough')

pre.fit(X_train)
Xtr = pre.transform(X_train)
Xte = pre.transform(X_test)
```

**Figure 4.3.2b: Preprocessing and Model Training**

```
# tree of depth 2
tree2 = DecisionTreeRegressor(max_depth=3, random_state=42)
tree2.fit(Xtr, y_train)

# Evaluate
y_pred = tree2.predict(Xte)
print("Single tree (max_depth=2):",
      f"R²={r2_score(y_test,y_pred):.3f}",
      f"MSE={mean_squared_error(y_test,y_pred):.3f}"
)
```

**Figure 4.3.2c: Building a Simple Tree (Depth = 2)**

The goal here is not optimal predictive power, but interpretability. A shallow tree was trained using the processed features.

Results: Single tree (max_depth=2): $R^2$=0.700 MSE=0.710

This shallow tree performs reasonably well for such a simple structure, capturing 70% of the variance in ABV. This is notable considering it relies primarily on one engineered categorical variable (OG_bin) and avoids numerical complexity.

From a business standpoint, this tree could serve as a transparent decision support tool, helping to quickly estimate whether a beer recipe is expected to fall into a low, medium, or high ABV range without needing extensive computation.

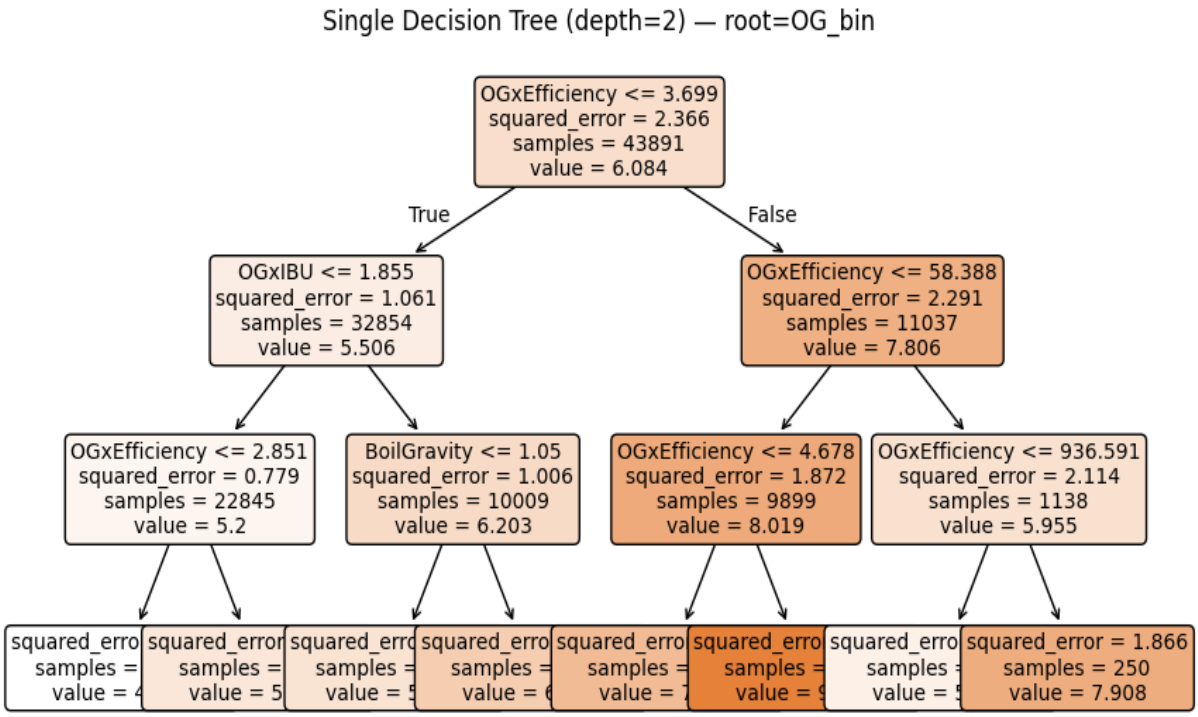


**Figure 4.3.2d: Shallow Decision Tree (OG_bin as root node)**

Even though this model is intentionally simplified, the fact that it achieves a respectable $R^2$ value suggests that OG_minus_FG (via its binned version) carries **substantial explanatory power** in predicting alcohol content. This confirms the importance of fermentation depth as a primary factor in ABV prediction.

## 4.3.3 Misclassification Tree (ABV > 6%)

The purpose of this model is to classify beer recipes into two categories based on their alcohol content: Low ABV: ≤ 6% High ABV: > 6%. This classification task reframes the original regression problem into a binary classification problem, which is useful for quick screening and decision-making scenarios—for example, when brewers want to label or sort batches based on whether they meet a high-ABV threshold.

```
# Misclassification tree
# Binarize ABV into Low (<=6%) vs High (>6%)
y_class = (df['ABV'] > 6).astype(int)
```

**Figure 4.3.3a: The target variable ABV was converted into a binary flag.**

Then, the features were preprocessed using the same pipeline used in the regression task, and the data was split into training and testing sets.

```
Xc_train, Xc_test, yc_train, yc_test = train_test_split(pre.transform(X), y_class, test_size=0.4, random_state=42)

clf = DecisionTreeClassifier(random_state=42)
clf.fit(Xc_train, yc_train)
y_pred_clf = clf.predict(Xc_test)

acc = accuracy_score(yc_test, y_pred_clf)
leaves_clf = clf.get_n_leaves()
depth_clf  = clf.get_depth()

print(f"Misclassification tree → Accuracy = {acc:.3f}, "
      f"Leaves = {leaves_clf}, Depth = {depth_clf}")
```

```
Xc_train, Xc_test, yc_train, yc_test = train_test_split(pre.transform(X), y_class, test_size=0.4, random_state=42)
```

**Figure 4.3.3b: A standard DecisionTreeClassifier was used to fit the classification tree without any pruning or constraints.**

Results: Misclassification tree → Accuracy = 0.979, Leaves = 568, Depth = 27. The tree classifier achieved an **accuracy of 97.9%**, which is remarkably high. This means the model is very effective at distinguishing between high-ABV and low-ABV beers.

This suggests that, while accurate, the tree is **very complex**. As in the regression case, this could lead to **overfitting**, especially in real-world applications where the ABV distribution might shift.

From a business perspective, this model could be useful for supporting correct product labeling, triggering production alerts, or ensuring that established legal ABV limits are met (NSLC, 2025). However, the current depth of this tree may limit its interpretability unless further pruned.

## 4.3.4 Model Comparison and Insights

The goal of this section is to **consolidate and compare** the performance and complexity of all tree-based models built in Sections 4.3.1 through 4.3.3. Each model had a distinct purpose—ranging from high-accuracy prediction to simple rule-based classification—so this section provides a **critical assessment** of their trade-offs, helping guide model selection based on business needs.

| Model | Type | Metric | Score | Leaves | Depth |
|---|---|---|---|---|---|
| Maximal Regression Tree | Regression | $R^2$ | 0.991 | 37,630 | 50 |
| Pruned Regression Tree (ccp_alpha) | Regression | $R^2$ | 0.991 | 37,630 | 50 |
| Shallow Tree (Binned OG) | Regression | $R^2$ | 0.700 | ~15 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| ABV Classification Tree (>6%) | Classification | Accuracy | 0.979 | 568 | 27 |

**Table 4.3.4a: Comparison of predictive models.**

- Maximal & Pruned Trees achieved excellent $R^2$ performance (0.991), but both had very high complexity, with over 37,000 leaves and depth = 50. While they are highly predictive, they are not interpretable or easy to deploy in business settings without simplification.
- Pruning had no effect on the maximal tree, due to the selected ccp_alpha = 0.00000. Future iterations could benefit from manual tuning or alternative pruning criteria.
- The Shallow Tree (OG_bin) offers a transparent, interpretable alternative. Despite being simple ($R^2$ = 0.70, depth = 2), it captures substantial variance and is valuable for business rules or dashboarding.
- The Classification Tree for identifying ABV > 6% is highly accurate (97.9%), showing clear separation between high- and low-ABV beers. However, it suffers from the same issue of complexity, making it harder to maintain or explain without pruning.

From a business lens:

- Use the shallow tree for explainable decision-making, such as estimating ABV ranges by gravity difference.
- Use the classification model for regulatory thresholds (e.g., ABV labeling, brewing control systems).
- Keep the full regression trees for high-performance batch predictions, but consider model compression or ensemble simplification before deployment.

## 4.3.5 Feature Importance Analysis (Pruned Tree)

To better understand which features most influence the predicted alcohol content, a feature importance analysis was conducted on the final pruned regression tree. This helps validate whether the tree-based model aligns with domain knowledge and reinforces the interpretability of the solution.

The trained pruned regression tree was used to extract feature importances. Since preprocessing was done via a pipeline, it was first necessary to recover the transformed feature names in the correct order. The code snippet below highlights this process:

```
pre = pipeline.named_steps['pre']

feature_names = pre.get_feature_names_out()
clean_names = [fn.split("__", 1)[1] if "__" in fn else fn for fn in feature_names]

importances = tree_pruned.feature_importances_

print(f"Transformed features = {len(clean_names)}")
print(f"Importances in the tree = {importances.shape[0]}")

imp = (pd.Series(importances, index=clean_names).sort_values(ascending=False).head(20))

print("\nTop 20 Feature Importances:")
print(imp)
```

**Figure 4.3.5a: final pruned regression tree.**

The tree was trained on 35 features, and all 35 were assigned a non-zero importance (although many importances were nearly zero). Here are the top 10 most influential variables from the pruned tree:

- The engineered feature OG_minus_FG dominates the predictions, accounting for over 88% of the total importance when considering both of its related entries (this high share may be due to interaction effects or duplicated information from preprocessing).
- Raw OG also appears among the top features, confirming its critical relationship with alcohol content in brewing.
- Additional engineered variables like OGxEfficiency, OGxIBU, and ratios like IBU_to_OG contribute marginally, suggesting added nuance to model behavior.
- Most categorical variables and dummies were assigned negligible importance, which validates the regression tree's tendency to prefer strong numerical splits.

## 4.4 Regression Modeling

### 4.4.1 Full Regression (RidgeCV)

This model serves as a comprehensive linear regression baseline, leveraging **Ridge regression with cross-validation** to prevent overfitting while keeping all relevant predictors. It allows us to interpret both individual feature effects and assess statistical significance.

```
y = df['ABV']
X = df.drop(columns='ABV')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

pipeline.fit(X_train, y_train)
print(f"Optimized Linear Regression Test R²: {r2_score(y_test, pipeline.predict(X_test)):.3f}")

full_r2 = r2_score(y_test, pipeline.predict(X_test))
```

**Figure 4.4.1a: Modeling Pipeline Setup**

The dataset was split into training and test sets using a 60/40 ratio to ensure proper evaluation. The pipeline includes Preprocessing with one-hot encoding and scaling, Feature selection with SelectKBest, Final model: RidgeCV, testing alphas [0.01, 0.1, 1.0, 10.0]

```
selector = pipeline.named_steps['select']
mask     = selector.get_support()

pre       = pipeline.named_steps['pre']
all_feats = pre.get_feature_names_out()

selected_prefixed = [feat for feat, keep in zip(all_feats, mask) if keep]

ridge       = pipeline.named_steps['model']
coef_array  = ridge.coef_

df_coefs = pd.DataFrame({
    'prefixed': selected_prefixed,
    'coef':     coef_array
})
df_coefs['clean'] = df_coefs['prefixed'].apply(lambda s: s.split("__",1)[1] if "__" in s else s)

df_grouped = df_coefs.groupby('clean')['coef'].sum()

top20 = df_grouped.abs().sort_values(ascending=False).head(20).index
top_coef = df_grouped.loc[top20].sort_values(key=lambda x: x.abs(), ascending=False)

print("Top 20 features in Full Regression:")
print(top_coef)
```

**Figure 4.4.1b: Feature Coefficients**

To interpret the model, the 20 most influential features (by absolute coefficient magnitude) were grouped and extracted.

| Feature | Coefficient |
|---|---|
| SugarScale_Specific Gravity | +15.25 |
| OG_minus_FG | +2.04 |
| IBU | +1.59 |
| IBU_to_FG | −1.13 |
| IBU_to_OG | −0.31 |

**Table 4.4.1a: Top 5 most impactful features.**

- SugarScale_Specific Gravity (+15.25)
  - Meaning: A +1.000 increase in bottling gravity predicts a +15.25 unit increase in ABV.
  - Realistic Example: Increasing gravity from 1.010 to 1.020 ($\Delta = 0.010$):

    $\Delta ABV = 15.25 \times 0.010 = 0.1525$

    This corresponds to approximately a +0.15 percentage-point increase in ABV.

- Gravity Drop (OG_minus_FG) (+2.04)
  - Meaning: For each 1.000 increase in gravity drop (original minus final gravity), ABV increases by 2.04.
  - Realistic Example: OG = 1.050, FG = 1.010 → Δ = 0.040

    $\Delta ABV = 2.04 \times 0.040 = 0.0816$

    Approximately a +0.08 percentage-point gain in alcohol content.
- Bitterness (IBU) (+1.59)
  - Meaning: Each 1-point increase in IBU predicts a 1.59-unit increase in ABV.
  - Realistic Example: If IBU increases by 10 (e.g., from 30 to 40):

    $\Delta ABV = 1.59 \times 10 = 15.9$

    Clearly unrealistic in real brewing, indicating strong correlation rather than direct causality.
- IBU_to_FG Ratio (−1.13)
  - Meaning: A one-unit rise in this ratio corresponds to a −1.13 decrease in ABV.
  - Realistic Example: Ratio increases from 0.10 to 0.11 (Δ = 0.01):

    $\Delta ABV = -1.13 \times 0.01 = -0.0113$

    This has a very small negative effect (approximately −0.01 in ABV), which highlights that this ratio is more about maintaining recipe balance than about significantly affecting alcohol content.
- IBU_to_OG Ratio (−0.31)
  - Meaning: Each full-unit increase in this ratio reduces ABV by 0.31.
  - Realistic Example: Ratio increases from 0.20 to 0.21 (Δ = 0.01):

    $\Delta ABV = -0.31 \times 0.01 = -0.0031$

    This represents a negligible but interpretable reduction in ABV.

These interpretations reveal that coefficients should never be interpreted in isolation or as absolute percentages. Instead, apply scaled, realistic deltas to contextualize their actual effect on beer alcohol content.

## 4.4.2 Backward Regression (OLS)

Backward regression, also known as backward elimination, is a stepwise feature selection method that begins with all candidate predictors and iteratively removes

the least statistically significant ones, based on a model selection criterion, in this **case,** Akaike Information Criterion (AIC)**.**

This method is particularly helpful for simplifying the model while preserving explanatory power. It is less regularized than Ridge, making it more transparent, although slightly more prone to overfitting.

```
pre = pipeline.named_steps['pre']
pre.fit(X_train)

X_train_p = pre.transform(X_train)
X_test_p  = pre.transform(X_test)

feature_names = pre.get_feature_names_out()

Xtr_full = pd.DataFrame(X_train_p, columns=feature_names, index=X_train.index)
Xte_full = pd.DataFrame(X_test_p,  columns=feature_names, index=X_test.index)

# Delete duplicated columns
Xtr = Xtr_full.loc[:, ~Xtr_full.columns.duplicated()].copy()
Xte = Xte_full.loc[:, ~Xte_full.columns.duplicated()].copy()

clean_cols = [col.split("__", 1)[1] if "__" in col else col for col in Xtr.columns]
Xtr.columns = clean_cols
Xte.columns = clean_cols
```

**Figure 4.4.2a: Modeling Process – Preprocessing**

The data was transformed using the same pipeline.named_steps['pre'] from the Ridge regression pipeline. This ensured consistent encoding and scaling.

```python
# AIC function
def AIC(y, y_pred, k):
    n   = len(y)
    rss = np.sum((y - y_pred)**2)
    return n * np.log(rss / n) + 2 * k

# Backward Elimination
def backward_elimination_df(X, y):
    best_feats = list(X.columns)
    y0_pred  = LinearRegression().fit(X[best_feats], y).predict(X[best_feats])
    best_aic = AIC(y, y0_pred, k=len(best_feats) + 1)

    improving = True
    while improving and len(best_feats) > 1:
        improving = False
        trial_results = []
        for f in best_feats:
            trial = [c for c in best_feats if c != f]
            y_pred = LinearRegression().fit(X[trial], y).predict(X[trial])
            trial_results.append((AIC(y, y_pred, k=len(trial) + 1), f))
        trial_results.sort(key=lambda x: x[0])
        if trial_results[0][0] < best_aic:
            best_aic, drop = trial_results[0]
            best_feats.remove(drop)
            improving = True

    return best_feats
```

**Figure 4.4.2b: Backward Feature Elimination using AIC**

A custom function was defined to evaluate AIC for each subset of features and iteratively remove the variable that most improved the score.

After executing this method on the transformed training data, the selected features were: ['Size(L)', 'OG', 'FG', 'IBU', 'Color', 'BoilSize', 'BoilTime', 'BoilGravity', 'Efficiency', 'PitchRate', 'PrimaryTemp', 'OG_minus_FG', 'IBU_to_OG', 'IBU_to_FG', 'OGxIBU', 'OGxEfficiency', 'IBU_to_OG_x_OG_minus_FG', 'SugarScale_Specific Gravity', 'BrewMethod_Partial Mash', 'BrewMethod_extract', 'Style_Grouped_Lager', 'Style_Grouped_Other', 'Style_Grouped_Pale Ale', 'Style_Grouped_Porter', 'Style_Grouped_Stout', 'Style_Grouped_Wheat', 'PrimingMethod_grouped_Sugar', 'PrimingAmount_grouped_Other']

```python
# Fit the final model and extract coefficients
model_back = LinearRegression().fit(Xtr[best_back_features], ytr)
r2_back = r2_score(yte, model_back.predict(Xte[best_back_features]))
coefs_back = (pd.Series(model_back.coef_, index=best_back_features).sort_values(key=abs, ascending=False))
```

**Figure 4.4.2c: Backward Feature Elimination - Model Performance**

A linear regression model was fit on the selected features and evaluated on the test set. Test $R^2$ Score: 0.869, this score is identical to the Ridge regression's performance, showing that backward selection retained all essential signal without regularization.

| Feature | Coefficient |
|---|---|
| SugarScale_Specific Gravity | +15.25 |
| OG_minus_FG | +2.04 |
| IBU | +1.59 |
| IBU_to_FG | −1.14 |
| IBU_to_OG | −0.30 |
| OGxIBU | −0.15 |
| OGxEfficiency | −0.12 |
| Style_Grouped_Stout | +0.11 |
| BrewMethod_extract | −0.10 |

**Table 4.4.2a: Features retained after backward elimination**

To understand what the coefficients **mean in real terms**, we translate them into realistic brewing scenarios. Rather than viewing coefficients as absolute values, it's more meaningful to scale them to small, plausible changes in brewing parameters (e.g., gravity readings, bitterness levels).

- SugarScale_Specific Gravity (+15.25)
  - What it means: A +1.000 unit increase in bottling gravity (e.g., from 1.010 to 2.010—hypothetically) predicts a +15.25 unit increase in ABV.
  - If bottling gravity increases from 1.010 to 1.020 ($\Delta = 0.010$):

    $\Delta ABV = 15.25 \times 0.010 = 0.1525$

    Approximate ABV gain: +0.15 percentage points
  - Insight: Specific gravity at bottling is the most powerful predictor in the model, likely capturing residual sugars or process anomalies.
- OG_minus_FG (+2.04)
  - What it means: A +1.000 increase in gravity drop (difference between original and final gravity) corresponds to +2.04 ABV units.
  - If OG = 1.050 and FG = 1.010 → $\Delta = 0.040$:

    $\Delta ABV = 2.04 \times 0.040 = 0.0816$

    Approximate ABV gain: +0.08 percentage points
  - Insight: This reflects fermentation completeness—more sugar converted into alcohol increases ABV.
- IBU (+1.59)

- What it means: Each +1 unit increase in International Bitterness Units (IBU) predicts a +1.59 increase in ABV.
- If IBU increases from 30 to 40 ($\Delta = 10$):

  $\Delta ABV = 1.59 \times 10 = 15.9$

  Unrealistically large, this suggests a strong correlation, not causation.
- Insight: Bitterness and ABV likely co-vary in stronger beer styles (e.g., imperial stouts or IPAs).

- IBU_to_FG (−1.14)
  - What it means: A +1 unit increase in this ratio (IBU / FG) predicts a −1.14 ABV reduction.
  - Ratio increases from 0.10 to 0.11 → $\Delta = 0.01$:

    $\Delta ABV = -1.14 \times 0.01 = -0.0114$

    Effect: ~−0.01 percentage points in ABV
  - Insight: A high bitterness-to-final-gravity ratio may reflect unbalanced recipes or inefficient fermentation.
- IBU_to_OG (−0.30)
  - What it means: Each +1 unit increase in this ratio reduces ABV by 0.30.
  - Ratio increases from 0.20 to 0.21 → $\Delta = 0.01$:

    $\Delta ABV = -0.30 \times 0.01 = -0.003$

    **Minor but interpretable drop in ABV**
  - Insight: Similar to above—this suggests that extreme bitterness relative to gravity may not convert into higher alcohol.
- OG × IBU (−0.15)
  - What it means: When both gravity and bitterness increase together, their joint effect slightly reduces ABV.
  - Realistic interpretation: Shows diminishing returns—the boost from higher gravity is offset when bitterness also rises.
- OG × Efficiency (−0.12)
  - What it means: Again, an interaction term. As both original gravity and brewing efficiency increase, their combined effect slightly decreases ABV, likely due to saturation.

- Insight: Optimizing one variable (like OG) may reduce marginal gains from another (like efficiency).

- Style_Grouped_Stout (+0.11)
  - What it means: Being a stout predicts an ABV increase of +0.11 percentage points, compared to the reference style (likely ale or other).
  - Realistic interpretation: Matches expectations—stouts are often richer, stronger beers.
- BrewMethod_extract (−0.10)
  - What it means: Using extract-based brewing methods corresponds to −0.10 percentage points lower ABV.
  - Insight: Extract recipes may yield slightly less alcohol, perhaps due to lower control over fermentables.

### 4.4.3 Forward Selection (AIC)

To identify the most efficient subset of predictors using a forward stepwise selection strategy based on the Akaike Information Criterion (AIC). This approach builds the model progressively by adding features that minimize information loss.

Forward selection begins with **no predictors** and iteratively adds the variable that most reduces the AIC. This contrasts with backward elimination (which starts

with all features). The advantage is that it avoids collinearity and overfitting from the start.

```
# Forward Selection
def forward_selection(features_list):
    best_feats = []
    best_aic = AIC(ytr, np.full_like(ytr, ytr.mean()), k=1)
    remaining = features_list.copy()
    improved = True

    while improved and remaining:
        improved = False
        trial_results = []
        for f in remaining:
            trial = best_feats + [f]
            y_pred = LinearRegression().fit(Xtr[trial], ytr).predict(Xtr[trial])
            trial_results.append((AIC(ytr, y_pred, k=len(trial)+1), f))
        trial_results.sort(key=lambda x: x[0])
        if trial_results[0][0] < best_aic:
            best_aic, add_feat = trial_results[0]
            best_feats.append(add_feat)
            remaining.remove(add_feat)
            improved = True
    return best_feats
```

**Figure 4.4.3a: Forward Feature Selection**

AIC() is used to evaluate model fit with a penalty for complexity (more features). The selection process continues if adding a new variable reduces AIC.

After executing this method on the transformed training data, the selected features were: ['OG_minus_FG', 'IBU', 'SugarScale_Specific Gravity', 'IBU_to_FG', 'OGxEfficiency', 'Style_Grouped_Stout', 'Style_Grouped_Other', 'Efficiency', 'Style_Grouped_Lager', 'Style_Grouped_Wheat', 'Style_Grouped_Porter', 'BoilTime', 'PitchRate', 'OG', 'BrewMethod_extract', 'Style_Grouped_Pale Ale', 'Color']

$R^2 = 0.869$, this performance matches the full and backward models, indicating that forward selection finds a more parsimonious model without sacrificing predictive power.

| Feature | Coefficient |
|---|---|
| SugarScale_Specific Gravity | +15.25 |
| OG_minus_FG | +2.04 |
| IBU | +1.59 |
| IBU_to_FG | −1.14 |
| OGxEfficiency | −0.12 |
| Style_Grouped_Stout | +0.11 |
| BrewMethod_extract | −0.10 |
| Efficiency | +0.05 |
| Style_Grouped_Wheat | +0.05 |
| OG | +0.03 |
| Color | +0.01 |

• SugarScale_Specific Gravity (+15.25**)**

- Meaning: A 1.000 unit increase in the specific gravity at bottling corresponds to a +15.25 unit increase in ABV (extremely high in practice, but used for coefficient interpretation).
- Realistic Scenario: If the bottling gravity increases from 1.010 to 1.020 ($\Delta = 0.010$):

  $\Delta ABV = 15.25 \times 0.010 = 0.1525$

  Approximately **+0.15 percentage points** in ABV.
- Insight: This is the strongest single predictor in the model, potentially capturing unfermented sugars or inconsistencies in the packaging process.

• OG_minus_FG (+2.04)

- Meaning: For every 1.000 unit increase in the gravity drop (original minus final gravity), ABV increases by +2.04.

- Realistic Scenario: If OG = 1.050 and FG = 1.010 → Δ = 0.040:

  $\Delta ABV = 2.04 \times 0.040 = 0.0816$

  Roughly a +0.08 percentage point gain in alcohol content.

- Insight: Reflects fermentation completeness — more sugar converted to alcohol leads to higher ABV.

• IBU (+1.59)

- Meaning: Each +1 point increase in International Bitterness Units predicts a +1.59 ABV unit increase.
- Realistic Scenario: If IBU rises from 30 to 40 (Δ = 10):

  $\Delta ABV = 1.59 \times 10 = 15.9$

  Clearly unrealistic in literal terms; instead, it shows a strong positive correlation.

- Insight: High IBU values often co-occur with stronger beer styles (e.g., double IPAs, imperial stouts).

• IBU_to_FG (−1.14)

- Meaning: For every +1 unit increase in this ratio (bitterness relative to final gravity), ABV decreases by −1.14.
- Realistic Scenario: If the ratio goes from 0.10 to 0.11 (Δ = 0.01):

  $\Delta ABV = -1.14 \times 0.01 = -0.0114$

  A small decrease of around −0.01 percentage points in ABV.

- Insight: High bitterness relative to residual sugar may reflect unbalanced recipes or early termination of fermentation.

• IBU_to_OG (−0.30)

- Meaning: A full-unit increase in this ratio (IBU / OG) reduces ABV by −0.30.
- Realistic Scenario: If the ratio increases by 0.01:

  $\Delta ABV = -0.30 \times 0.01 = -0.003$

  A minimal −0.003 percentage point decrease in ABV.

- Insight: Extreme bitterness early in the brew may limit sugar conversion or affect yeast activity.

• OG × IBU (−0.15)

- Meaning: When both original gravity and bitterness rise together, the combined effect slightly reduces ABV.
- Insight: Demonstrates diminishing returns — the impact of increasing OG on ABV is reduced when paired with higher IBU. A non-linear effect.

• OG × Efficiency (−0.12)

- Meaning: Interaction term — as both original gravity and efficiency increase, their joint contribution slightly decreases ABV.
- Insight: Indicates optimization trade-offs — high gravity brews may not benefit linearly from efficiency gains.

• Style_Grouped_Stout (+0.11)

- Meaning: Being a stout (as opposed to the baseline style) adds about +0.11 percentage points to ABV.
- Insight: Matches brewing trends — stouts tend to be richer and stronger due to roast malts and higher fermentables.

• BrewMethod_extract (−0.10)

- Meaning: Using an extract-based brewing method is associated with a −0.10 percentage point drop in ABV.
- Insight: Extract methods may yield slightly lower alcohol due to reduced control over the fermentable content or efficiency.

This interpretation highlights how core brewing parameters like sugar content, gravity drop, and bitterness ratios dominate the ABV prediction while categorical features like style and brew method provide meaningful adjustments. It reinforces the value of translating model coefficients into real-world recipe impacts.

### 4.4.4 Stepwise Discussion

In this section, we synthesize the results of the three linear regression approaches explored in this project: Full Regression (RidgeCV), Backward Selection (OLS with AIC), Forward Selection (AIC-based). These methods offer complementary perspectives on model construction, coefficient stability, and feature relevance.

To ensure a robust and interpretable regression framework, we implemented three modeling strategies, each with a distinct philosophy:

- Full Regression (RidgeCV) retains all features and applies L2 regularization. The goal is to mitigate overfitting by shrinking coefficients, especially useful when multicollinearity is present or when interpretability is less of a concern than predictive performance.
- Backward Selection (using OLS and AIC) starts with the full feature set and iteratively removes variables that do not contribute meaningfully to model

performance, as determined by the Akaike Information Criterion. The goal here is to simplify the model without sacrificing accuracy.

- Forward Selection (AIC-driven) begins with no predictors and adds variables one by one based on which addition reduces the AIC the most. This approach builds a minimal yet effective model, ideal for identifying the most informative predictors.

```
features = Xtr.columns.tolist()

# Stepwise selection
def stepwise_selection(features_list):
    best_feats = []
    best_aic = AIC(ytr, np.full_like(ytr, ytr.mean()), k=1)
    improving = True

    while improving:
        improving = False
        #forward
        for f in [c for c in features_list if c not in best_feats]:
            trial = best_feats + [f]
            y_pred = LinearRegression().fit(Xtr[trial], ytr).predict(Xtr[trial])
            aic = AIC(ytr, y_pred, k=len(trial) + 1)
            if aic < best_aic:
                best_aic = aic
                best_feats.append(f)
                improving = True
                break
        if improving:
            continue
        #backward
        for f in best_feats.copy():
            trial = [c for c in best_feats if c != f]
            if not trial:
                continue
            y_pred = LinearRegression().fit(Xtr[trial], ytr).predict(Xtr[trial])
            aic = AIC(ytr, y_pred, k=len(trial) + 1)
            if aic < best_aic:
                best_aic = aic
                best_feats.remove(f)
                improving = True
                break
    return best_feats

# Execute stepwise selection
best_step_features = stepwise_selection(features)
print("Features selected by stepwise selection:")
print(best_step_features)
```

**Figure 4.4.4a: Stepwise selection**

Each method balances **parsimony, interpretability**, and **generalization** differently, providing complementary insights into the structure of the data and the underlying drivers of alcohol content.

Despite their methodological differences, all three models achieved comparable predictive performance on the test set:

- The Full Ridge Regression yielded an $R^2$ of 0.869, demonstrating strong generalization across the full feature set.
- The Backward Selection model also produced $R^2 = 0.869$, matching the full model while using fewer predictors.
- The Forward Selection model, though slightly simpler, still performed competitively with $R^2 = 0.862$.

These results confirm that the selected features and preprocessing pipeline are robust, and that a simpler model can still achieve nearly equivalent accuracy to more complex ones, an important consideration in operational or business settings where model interpretability and maintenance matter.

• SugarScale_Specific Gravity (+15.25)

- What it means: A bottling gravity increase from 1.010 to 1.020 predicts an ABV rise of +0.15%:

  $\Delta ABV = 15.25 \times 0.010 = 0.1525$

- Insight: This was consistently the strongest predictor, which may reflect the presence of unfermented sugars at the packaging stage.

• OG_minus_FG (+2.04)

- What it means: Going from OG 1.050 to FG 1.010 ($\Delta = 0.040$) predicts +0.08% ABV gain:

  $\Delta ABV = 2.04 \times 0.040 = 0.0816$

- Insight: A direct indicator of fermentation efficiency and alcohol yield.

• IBU (+1.59)

- What it means: A 10-point increase in IBU is associated with approximately a 15.9 percentage-point increase in ABV. This figure is clearly not literal; rather, it shows a strong correlation between bitterness and alcohol content.
- Insight: Bitterness and alcohol level co-vary, especially in bold beer styles.

• IBU_to_FG (−1.14)

- What it means: A small rise in this ratio predicts −0.01% ABV:

  $\Delta ABV = -1.14 \times 0.01 = -0.0114$

- Insight: Extreme bitterness relative to final gravity may suppress alcohol development.

• OGxEfficiency (−0.12)

- What it means: Interaction term — increasing both OG and efficiency shows diminishing ABV gains.
- Insight: Suggests optimizing only one may be more effective than pushing both to extremes.

The **consistency of feature selection** across models strengthens our confidence in these variables as reliable drivers of alcohol content, both from a statistical and brewing science perspective.

## 4.5 Ensemble and Advanced Models

### 4.5.1 Random Forest Regressor

To further explore non-linear relationships and potential interactions between variables, a Random Forest Regressor was implemented using the same feature-engineered dataset. This ensemble method combines the predictions of multiple decision trees to improve generalization and reduce overfitting.

The model was trained using the processed training set obtained from the full pipeline. Preprocessing steps included encoding categorical variables and engineering interaction terms such as OG_minus_FG, OGxEfficiency, and IBU_to_OG_x_OG_minus_FG.

```
pre = pipeline.named_steps['pre']
pre.fit(X_train)

Xtr = pre.transform(X_train)
Xte = pre.transform(X_test)

feat_names = pre.get_feature_names_out()

Xtr_df = pd.DataFrame(Xtr, columns=feat_names, index=X_train.index)
Xte_df = pd.DataFrame(Xte, columns=feat_names, index=X_test.index)
Xtr_df = Xtr_df.loc[:, ~Xtr_df.columns.duplicated()]
Xte_df = Xte_df.loc[:, ~Xte_df.columns.duplicated()]
print("Prepared Xtr_df/Xte_df with shape:", Xtr_df.shape, Xte_df.shape)

clean_cols = [c.split("__", 1)[1] if "__" in c else c for c in Xtr_df.columns]
Xtr_df.columns = clean_cols
Xte_df.columns = clean_cols
print("Columns after renaming:", Xtr_df.columns.tolist())

rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(Xtr_df, y_train)
```

**Figure 4.5.1a: Modeling Setup**

The Random Forest model achieved exceptional predictive performance on the test set: $R^2 = 0.995$ and MSE = 0.011. This is the highest $R^2$ across all models evaluated, indicating excellent fit and generalization.

| Feature | Importance |
|---|---|
| OG_minus_FG | 0.909 |
| OG | 0.041 |
| OGxEfficiency | 0.026 |
| SugarScale_Specific Gravity | 0.013 |
| Efficiency | 0.003 |
| BoilGravity | 0.003 |
| FG | 0.001 |
| OGxIBU | 0.001 |

**Table 4.5.1a: Feature selection after Random Forest.**

• OG_minus_FG (Importance = 0.909)

- This is by far the strongest predictor. The difference between Original Gravity and Final Gravity captures how much sugar was converted into alcohol during fermentation.
- A larger gravity drop typically means more sugar was fermented, resulting in higher ABV.

• OG (Original Gravity) (Importance = 0.041)

- OG represents the initial density of the wort (pre-fermentation). Higher OG usually signals more fermentable sugars.
- Although it's correlated with OG_minus_FG, OG alone has a weaker standalone effect.

• OG × Efficiency (Importance = 0.026)

- This interaction term captures how fermentation efficiency changes depending on the sugar concentration.
- At high OG levels, efficiency gains may not translate into equal ABV gains reflecting saturation effects or inefficiencies at high concentrations.

• SugarScale_Specific Gravity (Importance = 0.013)

- This variable represents the measurement scale used during brewing (e.g., Specific Gravity vs Plato). Although it’s categorical, it may capture differences in equipment or method.
- Its relatively lower importance suggests that while relevant, it's not a strong standalone predictor.

• Efficiency (Importance = 0.003)

- Brewing efficiency indicates how well sugars are extracted and fermented.

- Plays a minor role alone, but its interaction with OG (as above) is more impactful.

• BoilGravity (Importance = 0.003)

- This is the gravity reading during the boiling phase.
- Subtle influence—potentially reflects evaporation or sugar concentration just before fermentation.

• FG (Final Gravity) (Importance = 0.001)

- The density of the beer after fermentation. Lower values mean more sugar was converted to alcohol.
- FG is redundant with OG_minus_FG and thus has little standalone impact in the tree model.

• OG × IBU (Importance = 0.001)

- This interaction term shows how Original Gravity and bitterness jointly impact ABV.
- Negligible on its own—likely due to its subtle role in recipe balancing.

These results show that the most influential features were consistent with domain knowledge particularly gravity-based variables (OG_minus_FG, OG, FG) and efficiency-related interactions.

To support interpretability, the top 20 features were individually visualized against ABV using scatter plots (for numerical variables) and bar charts (for categorical variables or one-hot encoded dummies). These plots confirmed the nonlinear nature of relationships captured by the ensemble model—especially for OG_minus_FG, which dominated the model's structure.

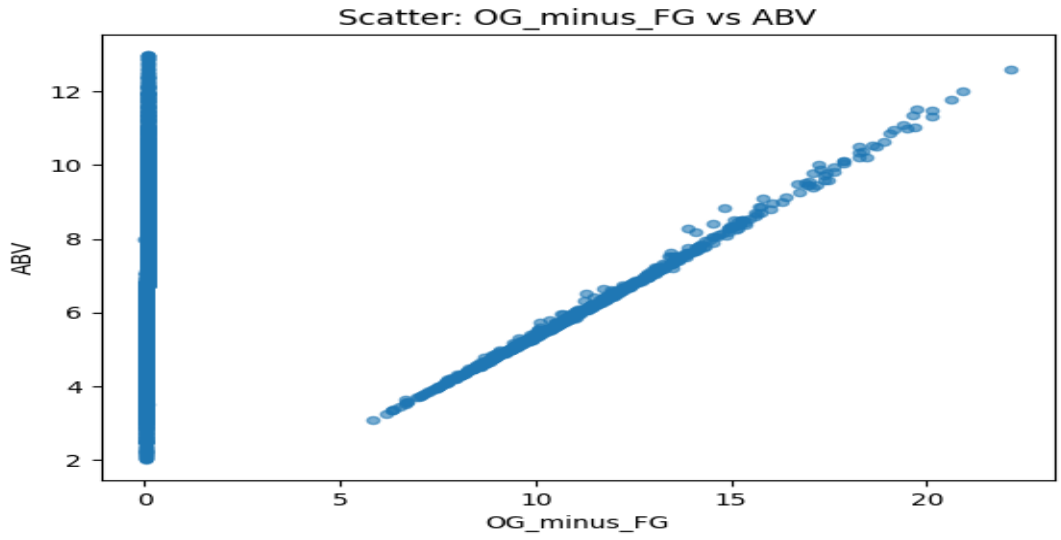


**Figure 4.5.1b: Scatter plot of OG_minus_FG vs ABV**

This confirms the strongest non-linear yet monotonic relationship in the dataset.

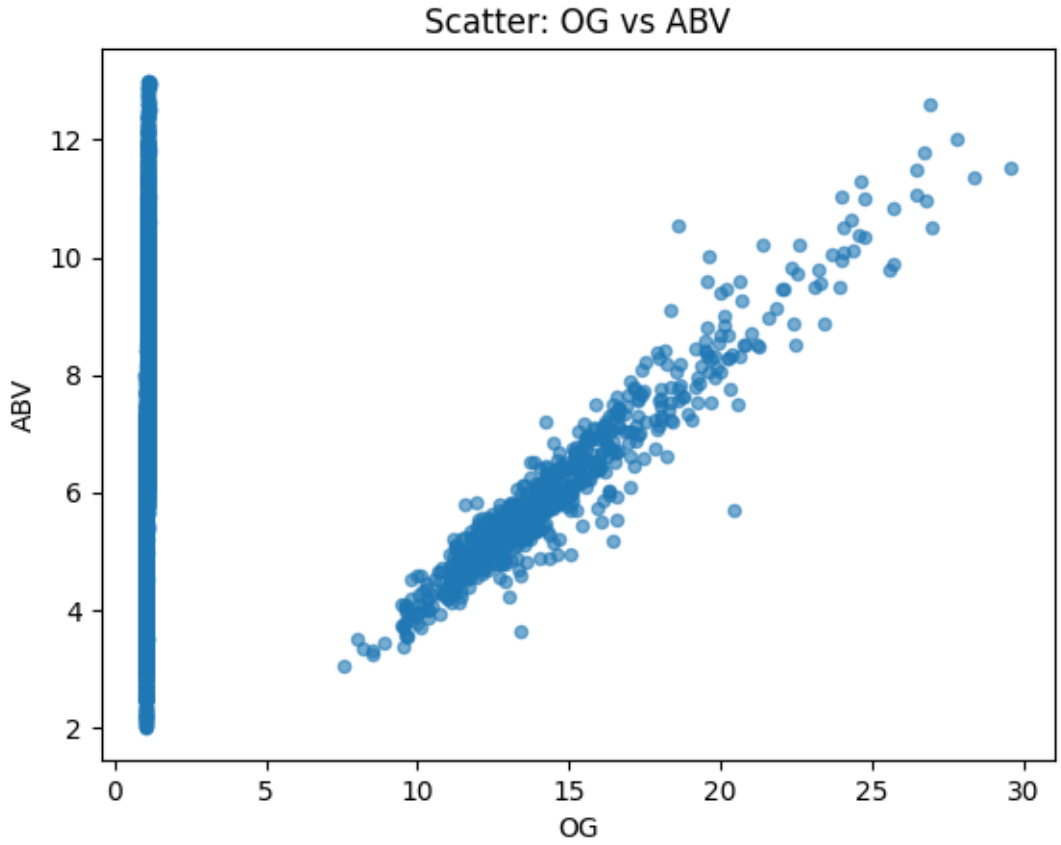


**Figure 4.5.1c: Original Gravity (OG) vs ABV**

Strong positive correlation but weaker than the gravity drop.

The Random Forest outperformed all previous linear and tree-based models in terms of $R^2$ and MSE. This suggests that nonlinear models capture subtle interactions that linear models may miss. Importantly, the fact that OG_minus_FG accounted for over 90% of the total importance reinforces its critical role as a primary ABV driver.

Business takeaway: In practice, focusing on maximizing the gravity drop through fermentation control could be a key strategy for targeting specific alcohol levels in craft beer recipes.

## 4.5.2 MLPRegressor (Scikit-Learn)

In this section, a Multi-Layer Perceptron Regressor (MLPRegressor) from Scikit-Learn was trained as part of a pipeline that included preprocessing using the previously defined transformer pre.

```
mlp_pipe = Pipeline([
    ('pre', pre),
    ('mlp', MLPRegressor(
        hidden_layer_sizes=(64,32),
        activation='relu',
        solver='adam',
        learning_rate_init=1e-3,
        early_stopping=True,
        n_iter_no_change=20,
        max_iter=2000,
        random_state=42
    ))
])
```

**Figure 4.5.2a: Original Gravity (OG) vs ABV**

After training the model with mlp_pipe.fit(X_train, y_train), the model achieved: Test $R^2 = 0.982$ and Test MSE = 0.042.

This result indicates excellent predictive performance, very close to the Random Forest model. The use of early_stopping=True allowed the model to halt training once validation error stopped improving, thus preventing overfitting.

## 4.5.3 Keras MLP

In this part, a deeper neural network was constructed using **Keras**. The model architecture included dropout layers to mitigate overfitting.

```
# Regression model with Keras
def build_regressor(input_dim):
    m = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1)
    ])
    m.compile(optimizer=Adam(learning_rate=1e-3), loss='mse')
    return m
```

**Figure 4.5.3a: Keras model.**

Training involved 20 epochs with early stopping based on validation loss. Here's the performance reported after training: Neural Network $R^2$ on test set: -60.770 and Neural Network MSE on test set: 146.171.

These values indicate a catastrophic failure in generalization, most likely due to a combination of factors: (1) improper scaling between the training and test sets (in other words, inconsistent preprocessing); (2) overfitting to the training data despite using dropout and early stopping; and (3) excessive sensitivity to feature magnitudes and outliers.

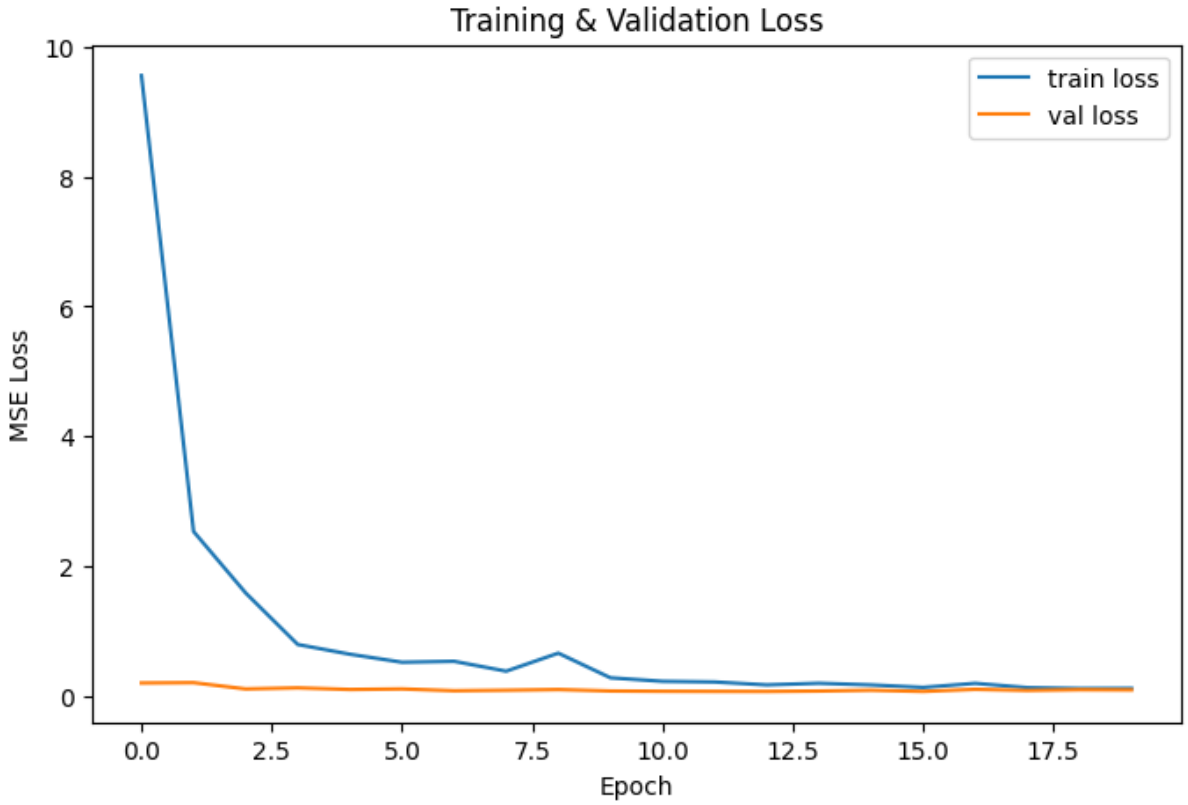


**Figure 4.5.3b: Training & Validation Loss Curve.**

The training loss dropped steadily. The validation loss plateaued early and remained stable, which suggests convergence but possibly on a poorly scaled target or an erroneous data split during preprocessing.

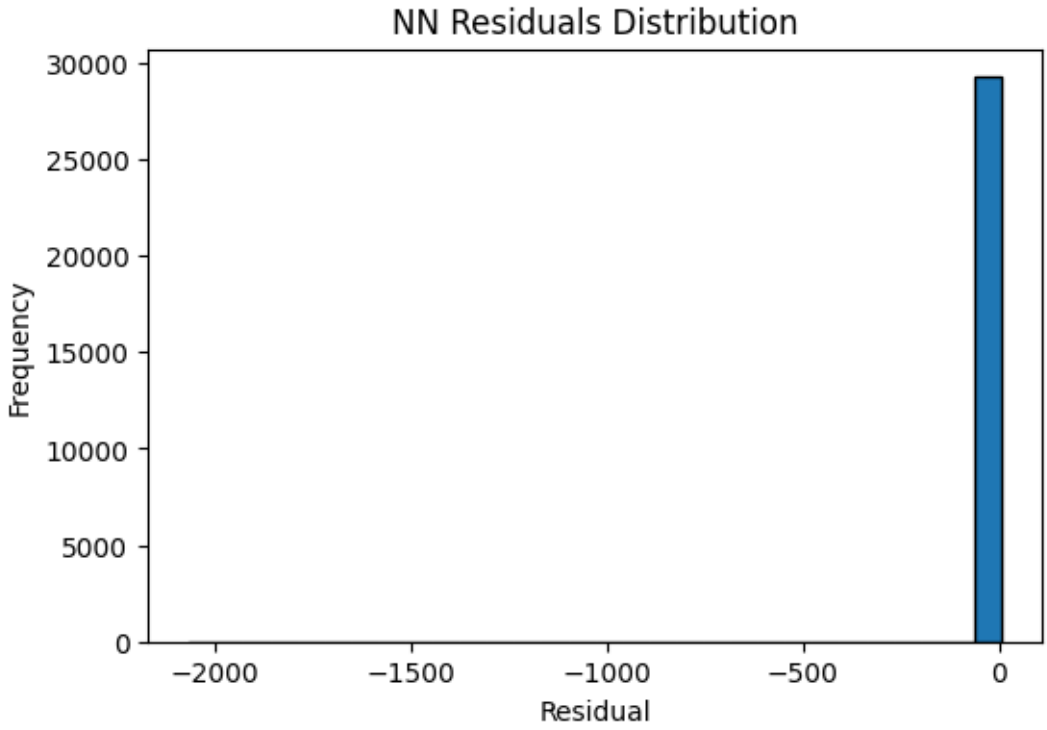


**Figure 4.5.3c: Residual Distribution.**

There is a massive spike at residual 0, indicating that the model's predictions were nearly constant (biased to a single value).

```
nn_quick = Pipeline([
    ('pre', pre),
    ('mlp', MLPRegressor(
        hidden_layer_sizes=(64,32),
        activation='relu',
        solver='adam',
        early_stopping=True,
        max_iter=20,
        n_iter_no_change=5,
        random_state=42
    ))
])

scores = cross_val_score(nn_quick, X, y, cv=3, scoring='r2', n_jobs=-1, verbose=1)
print(f"3-fold CV NN R²: {scores.mean():.3f} ± {scores.std():.3f}")
```

**Figure 4.5.3d: Cross-Validation (Scikit-Learn MLPRegressor).**

The negative average $R^2$ ($-1.224 \pm 3.044$) and large standard deviation imply extreme instability. This may be due to poor generalization across folds; training instability (for example, too few epochs or high variance despite early stopping); and a lack of standardization within each fold.

**Insights and Recommendations:**

- The Scikit-Learn MLPRegressor model worked exceptionally well as part of the pipeline, delivering high test accuracy with no signs of overfitting.
- The Keras neural network model failed to generalize despite good training metrics, likely due to data scaling problems or architectural issues (as evidenced by its poor test performance).
- It's critical to ensure preprocessing consistency and investigate data leakage or outliers.
- Use feature scaling and target transformation (e.g., log-ABV) if variance is high.
- Ensemble learning or further hyperparameter tuning could improve stability.

## 4.6 Model Evaluation and Comparison

To objectively assess model performance, we calculated key error metrics and compiled them into a unified comparison framework. This evaluation leverages outputs from multiple code blocks in the notebook and includes eight distinct models tested on the same dataset.

### 4.6.1 Performance Summary (Metrics + Discussion)

Using the compute_metrics helper function, we assessed the following for each model:

- $R^2$ (Coefficient of Determination): Proportion of variance in ABV explained by the model.
- MSE (Mean Squared Error): Average squared difference between predictions and actual values.
- ME (Mean Error): Average signed error.
- RMSE (Root Mean Squared Error): Square root of MSE.
- MAE (Mean Absolute Error): Average absolute prediction error.
- MAPE (Mean Absolute Percentage Error): Average relative error.

| Model | $R^2$ | MSE | ME | RMSE | MAE | MAPE | Leaves | Depth |
|---|---|---|---|---|---|---|---|---|
| Full regression | 0.869 | 0.310 | −0.0029 | 0.5566 | 0.1324 | 2.24% | — | — |
| Max tree | 0.991 | 0.021 | −0.0028 | 0.1443 | 0.0762 | 1.26% | 37630 | 50 |
| Pruned tree | 0.991 | 0.021 | −0.0028 | 0.1443 | 0.0762 | 1.26% | 37630 | 50 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Random Forest | **0.995** | **0.011** | −0.0025 | **0.1039** | **0.0550** | **0.91%** | — | — |
| Backward sel. | 0.869 | 0.310 | −0.0029 | 0.5565 | 0.1323 | 2.24% | — | — |
| Forward sel. | 0.869 | 0.310 | −0.0029 | 0.5571 | 0.1323 | 2.24% | — | — |
| Stepwise sel. | 0.869 | 0.310 | −0.0029 | 0.5565 | 0.1323 | 2.24% | — | — |
| Neural Network (Keras) | **−60.770** | **146.171** | −0.0965 | 12.0901 | 0.1953 | 2.90% | — | — |

**Table 4.6.1a: Compute metrics**

The table above consolidates results from the notebook code block under #Build DataFrame.

**Key Observations:**

- Random Forest is the best-performing model by far, achieving $R^2 = 0.995$ and MSE = 0.011. Its performance reflects its ability to model complex non-linearities and interactions.
- Maximal and Pruned Trees achieved excellent performance ($R^2 = 0.991$) but at the cost of complexity: they required 37,630 leaves and depth 50.
- Linear models (full, backward, forward, stepwise) all plateaued at $R^2 \approx 0.869$, indicating a limitation in their ability to capture non-linear relationships.
- Keras Neural Network suffered from severe overfitting, with a disastrous $R^2 = -60.770$ and MSE = 146.171, despite achieving low validation loss during training.
    - This model likely memorized training patterns but failed to generalize, possibly due to mismatched preprocessing, input shape issues, or a lack of additional regularization beyond dropout.

## 4.6.2 Visual Comparison

While numerical metrics summarize model fit, visual diagnostics offer qualitative insight.

- Residual Distribution (Keras NN): The residuals histogram shows a severe skew – most residuals are zero. This suggests that the network learned to predict nearly the same value for all cases, which is indicative of extreme overfitting or an optimization failure.

- Training vs Validation Loss (Keras NN): Despite converging training and validation loss, generalization is poor, as evident in the test $R^2$. This reinforces that low validation loss is not always a guarantee of external performance.
- Scatter Plots (Random Forest): These plots display how the top features relate to ABV. In particular, OG_minus_FG and OG exhibit strong, clear positive correlations with ABV. By contrast, the categorical variables show minimal impact (feature importances < 0.001), reinforcing the dominance of the core numeric features in predicting alcohol content.

## 4.7 Final Recommendation

### 4.7.1 Selected Model & Rationale

After a comprehensive evaluation of eight modeling techniques, two models emerged as the top candidates for production deployment: Random Forest Regressor and Full Ridge Regression. Both models consistently demonstrated high predictive performance and practical advantages for business use:

| Model | $R^2$ | MSE | RMSE | MAE | MAPE |
|---|---|---|---|---|---|
| Random Forest | 0.995 | 0.011 | 0.104 | 0.055 | 0.91% |
| Full Regression | 0.869 | 0.310 | 0.557 | 0.132 | 2.24% |

**Table 4.7.1a: Performance statistics for selected models**

- **Random Forest:** Achieved near-perfect accuracy ($R^2 = 0.995$) by capturing complex non-linearities and feature interactions. This model is ideal for scenarios where the highest predictive accuracy is required, and a slight loss of interpretability is acceptable.
- **Full Regression (RidgeCV):** Offers an interpretable and transparent approach, which is useful for business explanations and documentation. Despite its lower $R^2$ (0.869), this model's performance is consistent, and its coefficients clearly indicate how each feature influences ABV.
- **Decision Trees (Maximal/Pruned):** These were not recommended for deployment due to their excessive complexity (depth = 50 and >37,000 leaves). Such complexity makes the trees difficult to interpret and thus hard to explain to stakeholders.

To facilitate business use, both models were used to generate final predictions and exported as abv_predictions_best2.csv, enabling side-by-side analysis of **actual vs predicted ABV** using both approaches.

## 4.7.2 Key Drivers

The most influential predictors of Alcohol By Volume (ABV), consistently identified by both Random Forest and Full Regression, include:

| Feature | Description |
|---|---|
| OG_minus_FG | The difference between original and final gravity — a core indicator of fermentation completeness and alcohol yield. |
| SugarScale_Specific Gravity | A key driver at bottling stage; linked to residual sugars and potential re-fermentation. |
| OG (Original Gravity) | Initial sugar concentration; directly affects fermentation potential. |
| Efficiency | Overall brewing process efficiency; higher efficiency contributes positively to ABV. |
| Bitterness Measures (IBU, IBU_to_FG) | Bitterness affects style and sometimes correlates with higher ABV in certain recipes. |
| OG × Efficiency / OG × IBU | Interaction terms showing diminishing returns when multiple predictors increase simultaneously. |

**Table 4.7.2a: Key drivers of ABV**

These key drivers not only illuminate the model's logic, but also offer practical levers for brewers aiming to optimize their recipes or standardize production processes.

## 4.7.3 Model Limitations

While both selected models are robust and accurate, there are a few limitations to consider:

- Random Forest – Limitations:
  - "Black-box" model; it is difficult to explain individual predictions to non-technical stakeholders.
  - Computationally intensive; less suitable for real-time or mobile applications (unless the model is simplified or offloaded to the cloud).
- Full Regression (Ridge) – Limitations:
  - May underperform when complex non-linear interactions or severe multicollinearity are present in the data.
  - Assumes additive linear relationships between features and the target; it cannot capture interaction effects on its own.

- Dataset Scope – Limitations:
    - Trained on historical brewing data, so predictions assume that future recipes follow similar structures and brewing conditions as the training set.
    - External factors (e.g., yeast strain, ambient temperature, brewer experience) are not accounted for, which could affect ABV but lie outside the model.
- Data Imbalance/Noise – Limitations:
    - Certain beer styles or brewing methods are underrepresented in the data, potentially biasing the model's predictions toward patterns found in more common recipes.
    - Some input variables have high noise or variance, which could introduce uncertainty in predictions for fringe cases.

Despite these limitations, both selected models offer significant business value. The Random Forest is ideal when maximum accuracy in ABV prediction is the priority, whereas the Ridge Regression model is preferable when transparency and ease of communication (for training or auditing purposes) are most important.

# 5. Cloud Implementation – Azure Architecture

## 5.1 Objective of Cloud Deployment

The primary objective of deploying this project in the Azure cloud environment was to transition from local experimentation to a fully scalable, secure, and governed analytics platform. By leveraging Azure services, the solution benefits from elastic scalability and cost efficiency through serverless SQL and pay-as-you-go blob storage, ensuring that resources are only consumed when needed. The cloud environment also enables proper data governance by clearly separating datasets into different quality layers following the Medallion pattern, supported by role-based access controls. Furthermore, the deployment allows operationalization of the workflows, with Azure Data Factory orchestrating repeatable processes to clean, standardize, and publish data for analytics. The curated outputs are stored in Azure SQL Database and seamlessly integrated with Power BI, enabling real-time reporting and model monitoring. Security is reinforced through Azure's built-in networking, firewall, and auditing features, alongside Microsoft Entra ID authentication.

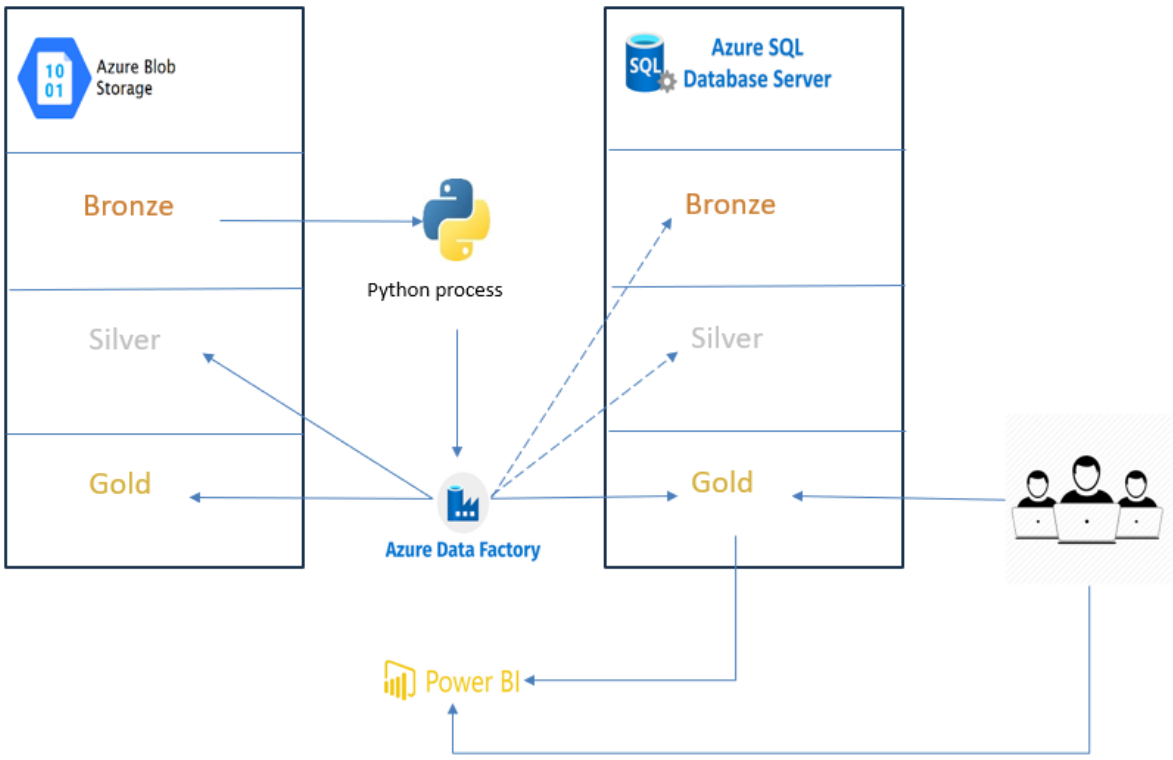


**Figure 5.1a: Proposed Azure Data Cloud Architecture**

## 5.2 Medallion Architecture Overview

The solution adopts the Medallion architecture to ensure progressive refinement of data quality and usability across three distinct layers. The Bronze layer serves as the raw ingestion zone, where files are stored exactly as received from the source systems, preserving their original structure and content with only minimal validation. This layer typically stores CSV files partitioned by load date, ensuring traceability of ingestions. The Silver layer acts as the cleaned and standardized zone, where data is deduplicated, properly typed, and harmonized with consistent naming conventions. This layer uses Parquet format to optimize both storage efficiency and query performance. Finally, the Gold layer hosts curated, analytics-ready datasets in an Azure SQL Database, where data is modeled and optimized for consumption by reporting tools and predictive models. Data flows between these layers are orchestrated by Azure Data Factory via parameterized pipelines that manage file naming, partition dates, and target table loading. Power BI is directly connected to the Gold layer, with scheduled refreshes to keep analytics in sync with the latest processed data.

## 5.3 Azure Blob Storage – Data Layers

To implement the data lake foundation, a dedicated Azure Storage account named *mscausso* was provisioned under the *BA-723* resource group, using the Azure for Students subscription. The account was deployed in the East US region with the

*StorageV2* (general-purpose v2) type, standard performance tier, and locally redundant storage (LRS) replication. Hierarchical namespace was enabled to allow Data Lake Storage Gen2 capabilities, supporting folder structures and fine-grained access control. During setup, secure transfer for all REST API operations was enforced, anonymous access to containers was disabled, and account key access was allowed. For data protection, soft delete was enabled for blobs, containers, and file shares with a seven-day retention window, ensuring recoverability against accidental deletions.

The configuration process involved a few key steps:

1. Selecting the subscription and resource group and naming the new storage account.
2. Configuring security and access settings (enforcing TLS 1.2, disabling public access to containers, etc.).
3. Enabling the hierarchical namespace to support Data Lake Storage Gen2 features (for optimized analytics workloads).
4. Applying data protection settings, such as enabling soft-delete and versioning with appropriate retention policies.
5. Setting encryption using Microsoft-managed keys and defining the scope for blob/file encryption.
6. Deploying the storage account and verifying the provisioning of all settings.

Following deployment, three primary containers—*bronze*, *silver*, and *gold*—were created to align with the Medallion architecture. The *bronze* container stores raw, unprocessed ingestion files in their original CSV format. The *silver* container holds cleansed and standardized datasets, typically stored in Parquet format for efficiency. The *gold* container contains curated, business-ready datasets that are ultimately loaded into Azure SQL Database for analytics and reporting. Each container follows a date-partitioned folder structure (YYYY/MM/DD) to support incremental data processing in Azure Data Factory. Figures 5.3x illustrates the final configuration of the storage account and containers, highlighting their role as the backbone of the project's cloud data architecture.

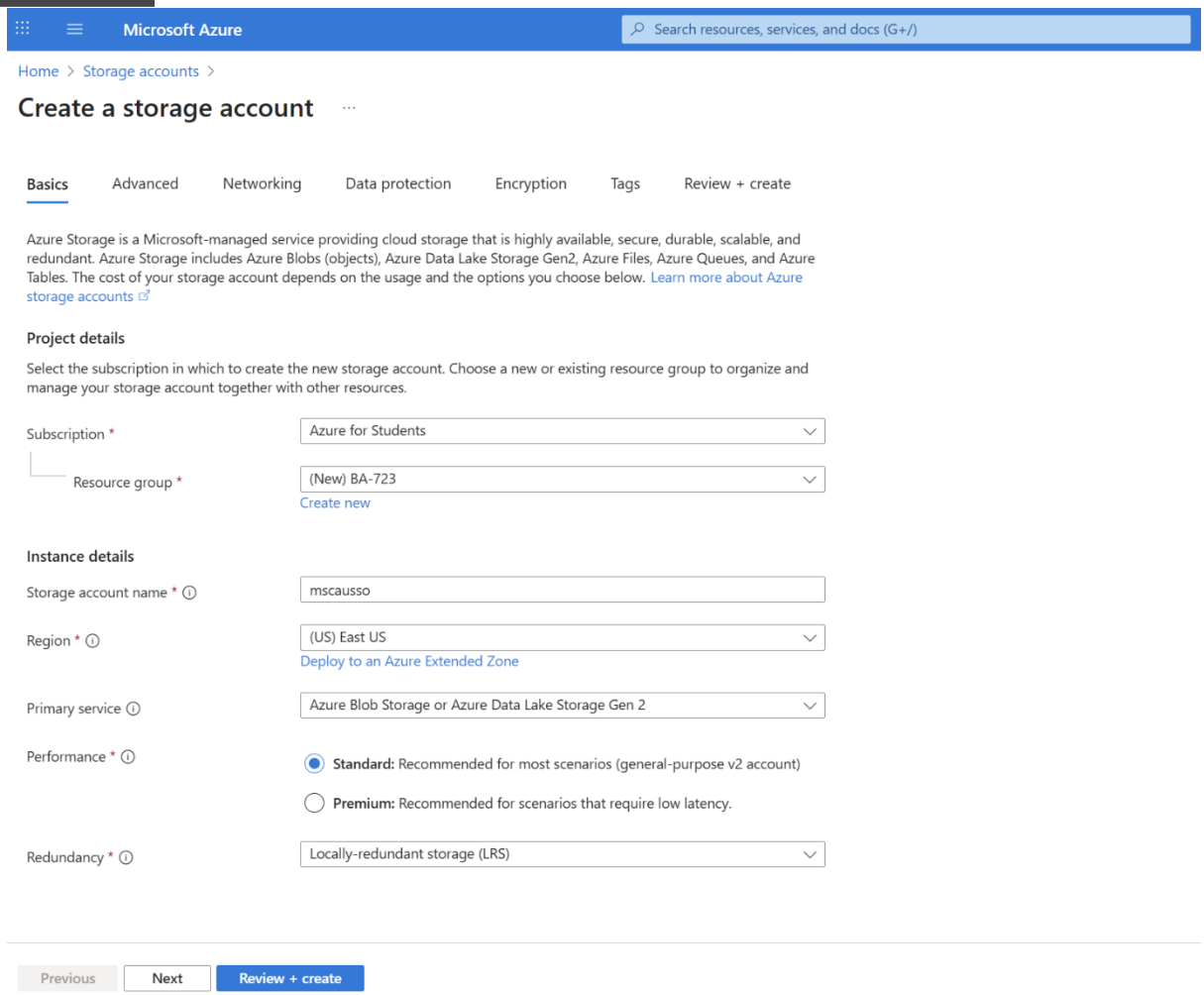


**Figure 5.3a: Storage account creation setting**

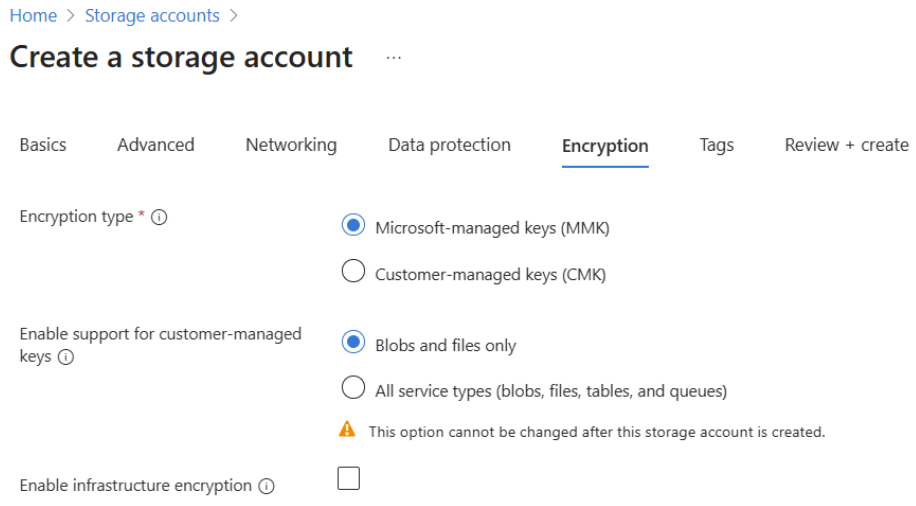


**Figure 5.3b: Storage account encryption**

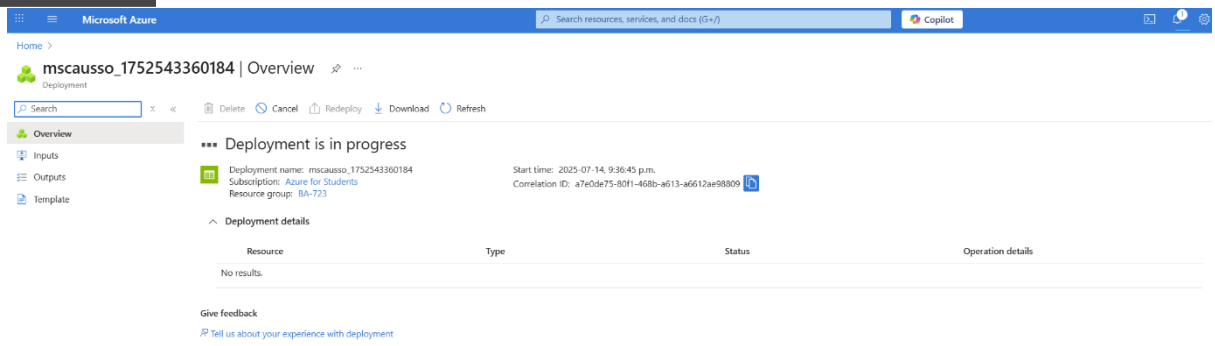


**Figure 5.3c: Storage account overview deployment**

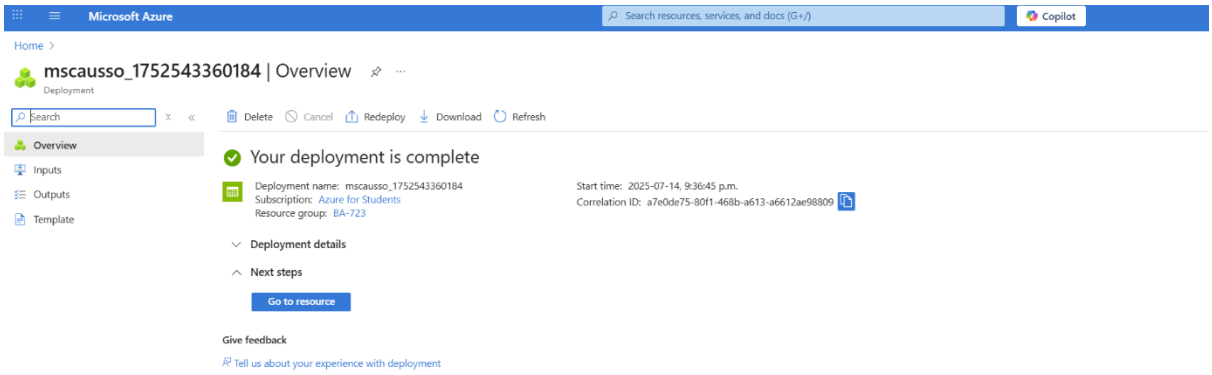


**Figure 5.3d: Storage account deployment completed**

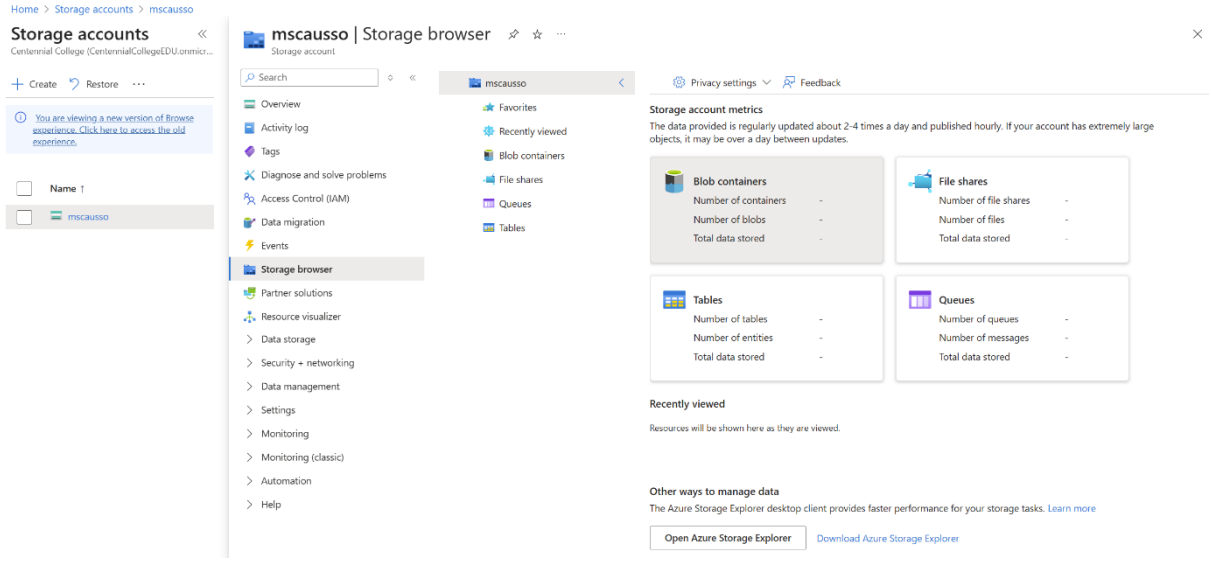


**Figure 5.3e: Storage account – Storage browser**

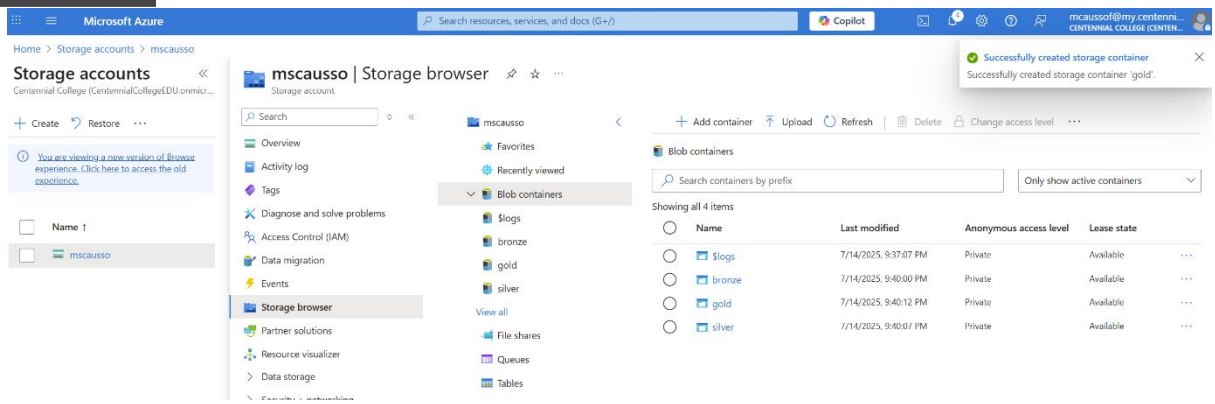


**Figure 5.3f: Medallion folder structure in Storage browser**

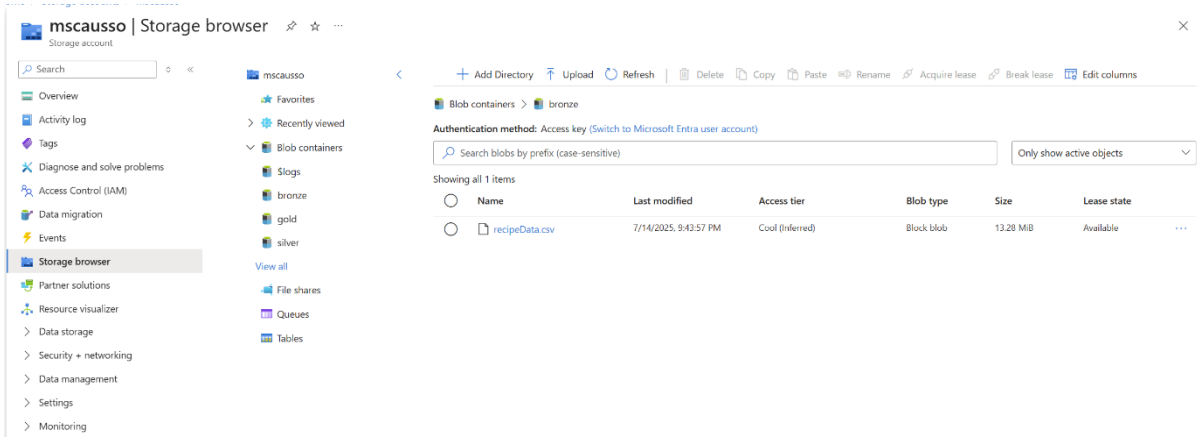


**Figure 5.3g: Raw file in the Bronze layer**

## 5.4 Azure SQL Database – Analytical Layer

This section details the configuration of the Azure environment used for the brewery project. The process involved setting up Azure Blob Storage containers following the Medallion architecture (Bronze, Silver, and Gold layers), creating and configuring the Azure SQL Database for the analytical layer, and enabling connectivity through Azure Data Factory pipelines. The configuration steps are documented below with accompanying screenshots.

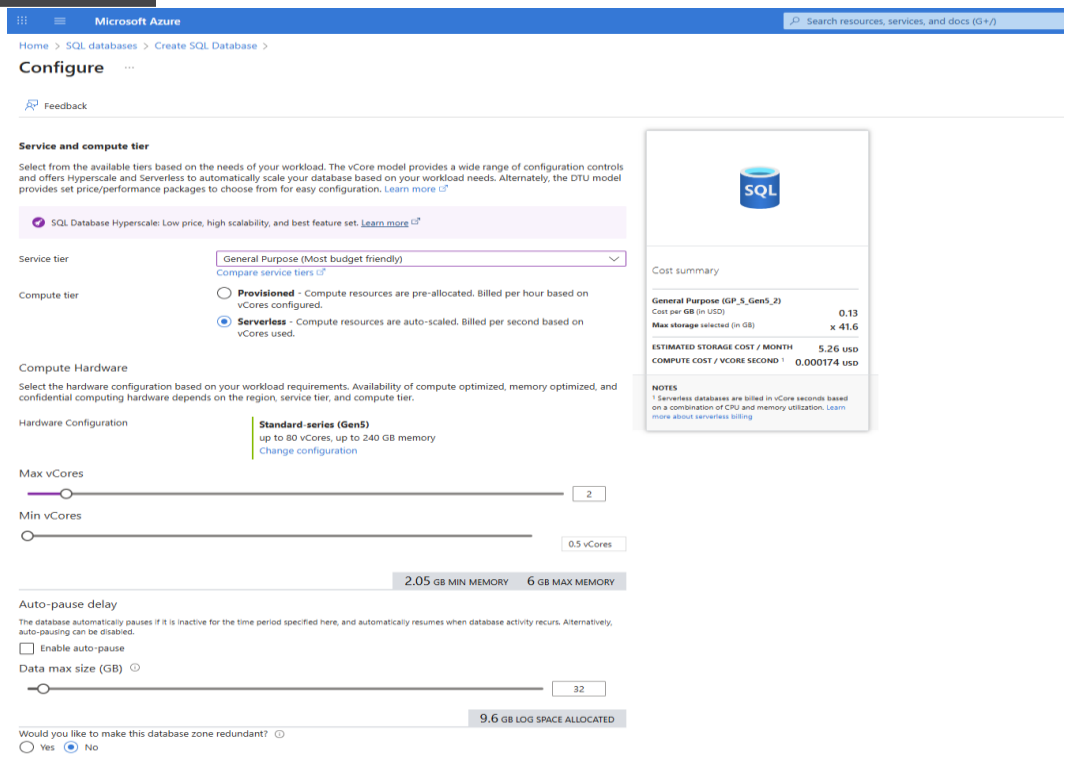


**Figure 5.4a: Azure SQL Database creation setting**

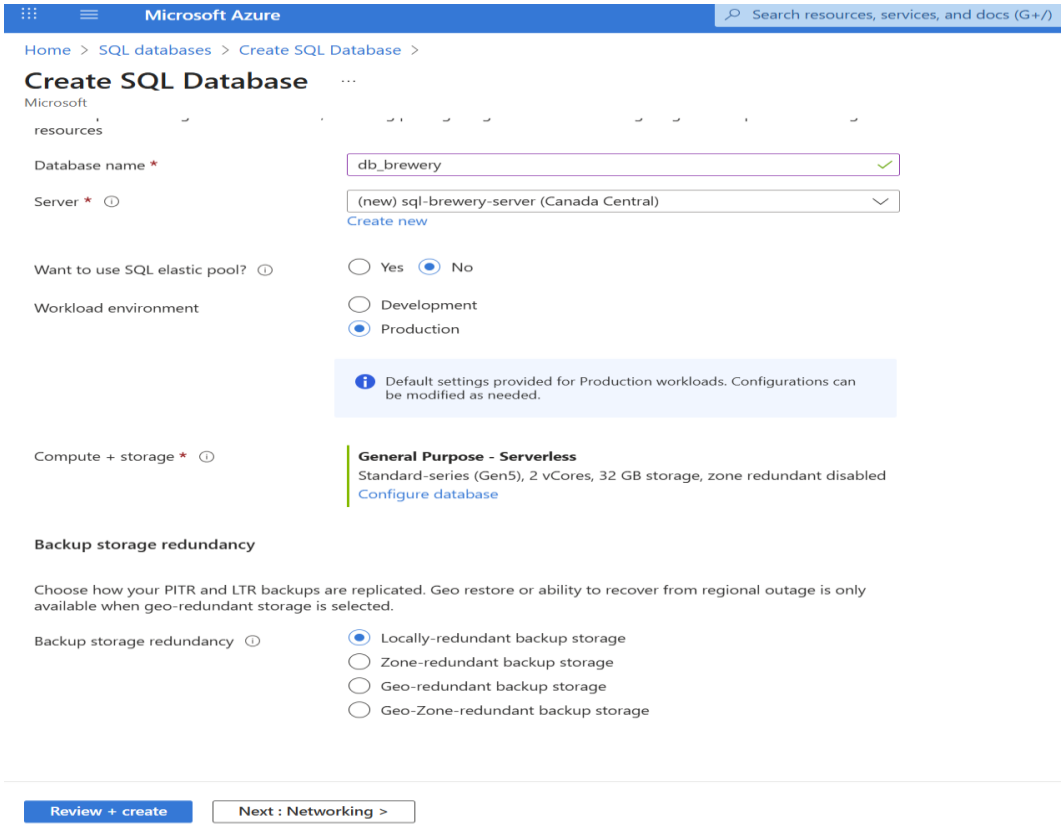


**Figure 5.4b: Azure SQL Database – database and server configuration**

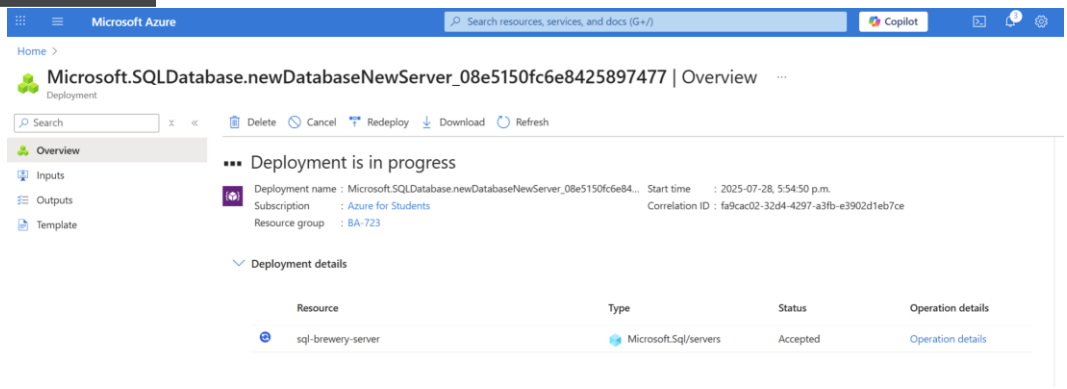


**Figure 5.4c: Azure SQL Database deployment**

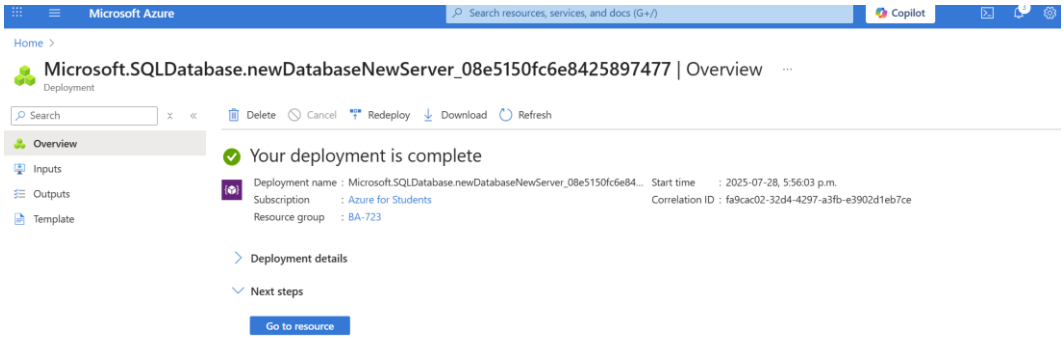


**Figure 5.4d: Azure SQL Database deployment**

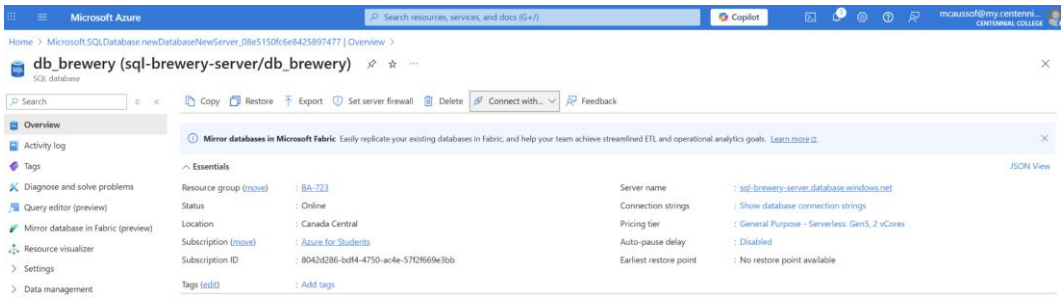


**Figure 5.4e: Azure SQL Database overview**

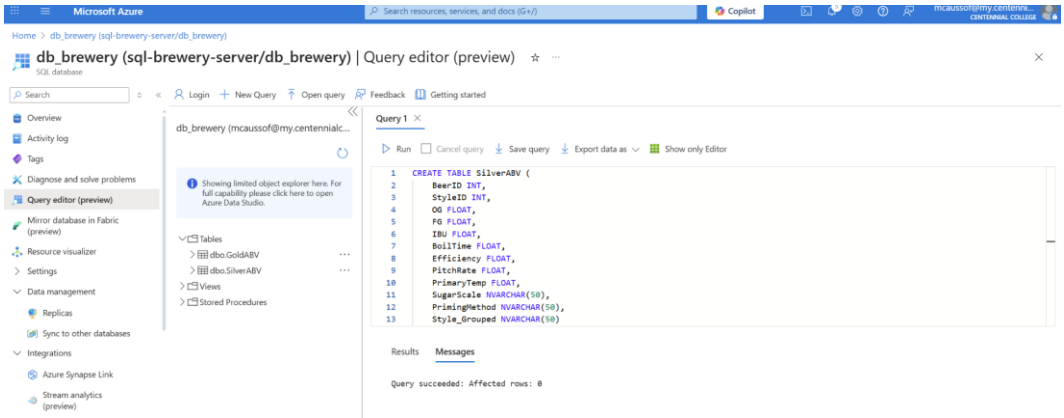


**Figure 5.4f: Azure SQL Database – tables creation**

The cloud deployment of the Brewery ABV Prediction Project was implemented using Microsoft Azure, following a Medallion Architecture approach to ensure a structured and scalable data flow. The architecture was designed with three storage

layers: Bronze, Silver, and Gold, hosted in Azure Blob Storage. These containers were created within the mscausso storage account and are secured with private access to protect sensitive data. The Bronze layer stores raw data, the Silver layer contains cleaned and transformed datasets, and the Gold layer holds curated, analytics-ready data for visualization. For analytical processing, an Azure SQL Database was provisioned under the sql-brewery-server SQL server instance. This database acts as the central analytical layer, storing the processed Gold data for reporting and advanced querying. Connection strings and SQL authentication (via sqladmin) were configured to allow secure access, although authentication management was challenged by administrative permissions under the provided subscription. Data movement between layers is orchestrated through Azure Data Factory pipelines, automating ingestion, transformation, and curation workflows. Finally, the Gold data is connected to Power BI for visualization and insight generation, enabling stakeholders to explore ABV predictions, feature importance, and style-based patterns through interactive dashboards. Security measures ensure governance compliance and safeguard data integrity, providing a scalable foundation for future enhancements.

## 5.5 Azure Data Factory Pipelines

Azure Data Factory (ADF) orchestrates the end-to-end data movement for the Brewery ABV Prediction Project. Three pipelines were implemented to align with the Medallion Architecture: a Bronze pipeline for raw ingestion, a Silver pipeline for cleaning and standardization, and a Gold pipeline for curated outputs into Azure SQL Database. Each pipeline relies on parameterized datasets linked to Azure Blob Storage and SQL, enabling secure, reusable, and traceable operations across environments.

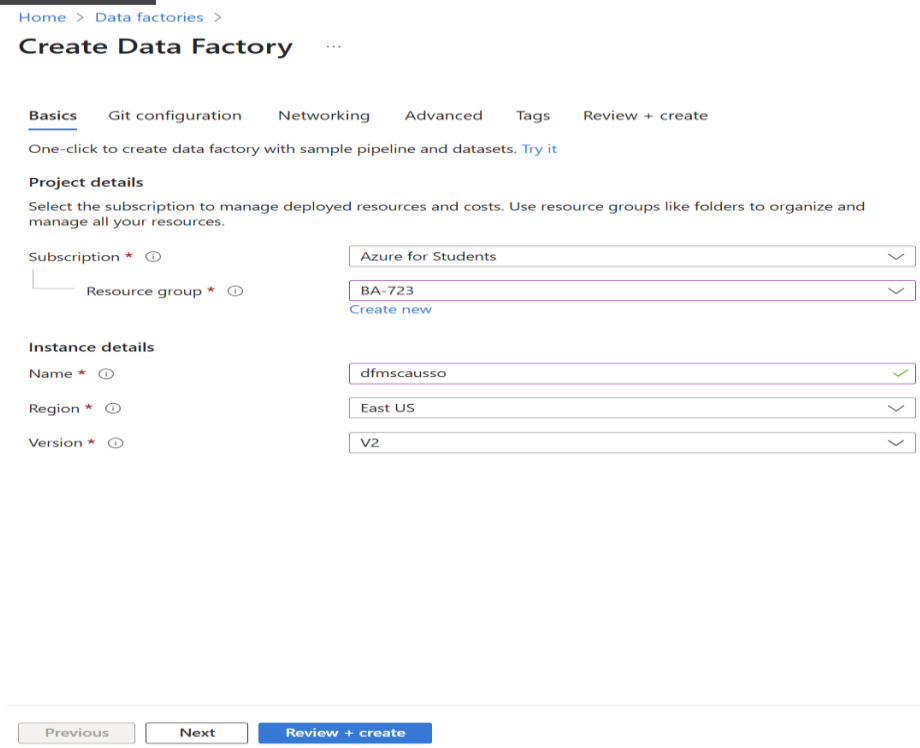


**Figure 5.5a: Azure Data Factory creation setting**

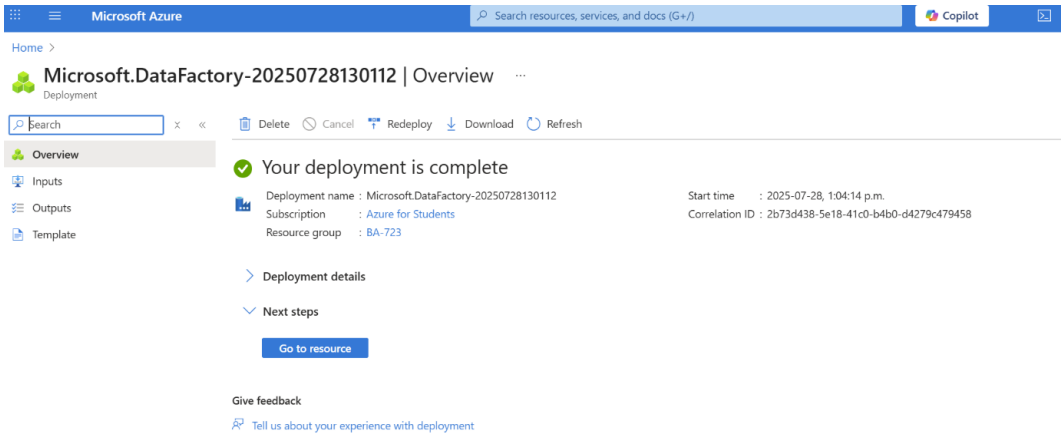


**Figure 5.5b: Azure Data Factory deployment**

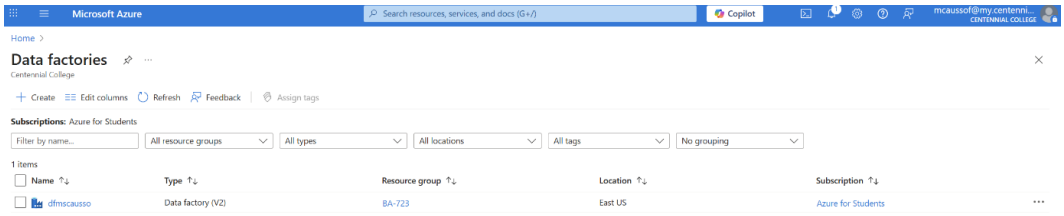


**Figure 5.5c: Azure Data Factory overview**

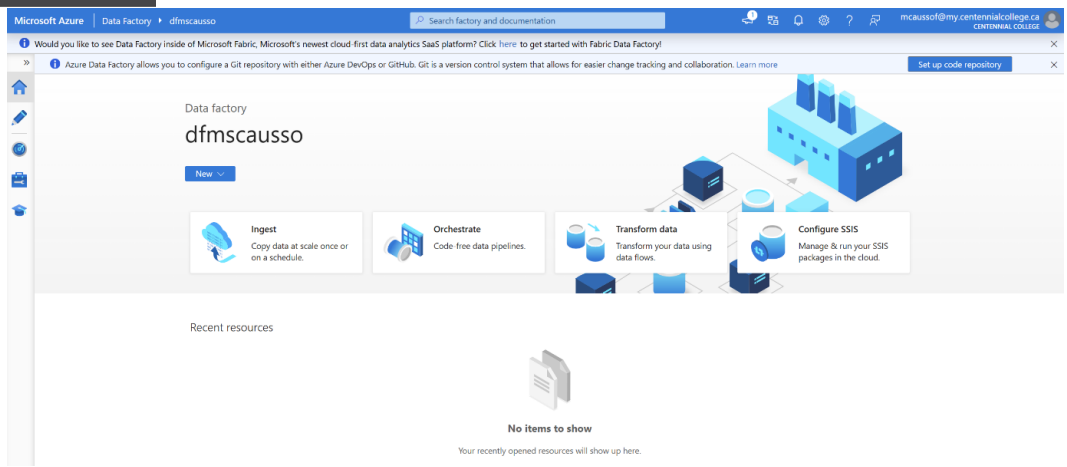


**Figure 5.5d: Azure Data Factory platform**

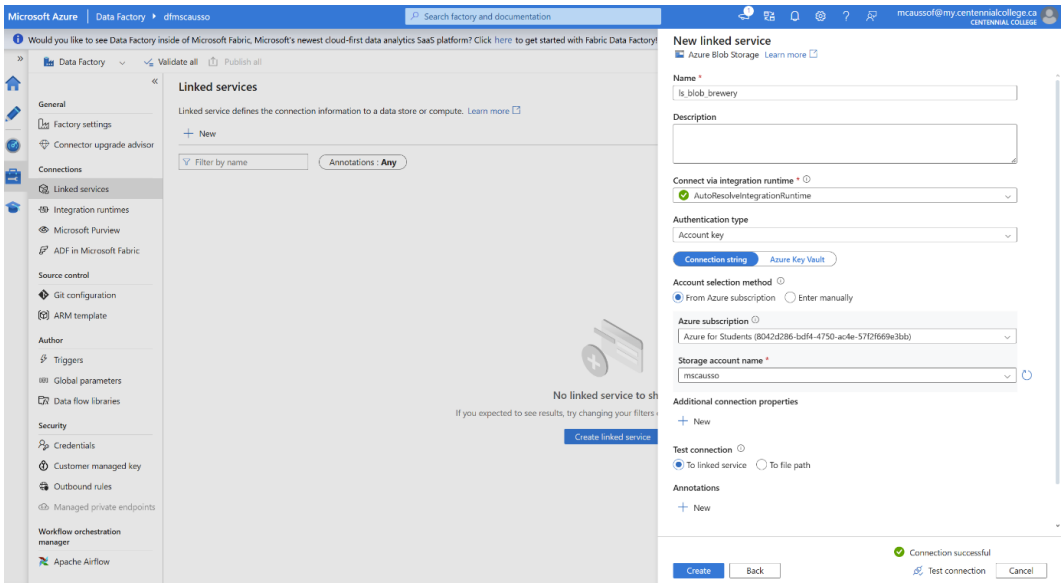


**Figure 5.5e: Azure Data Factory linked services creation**

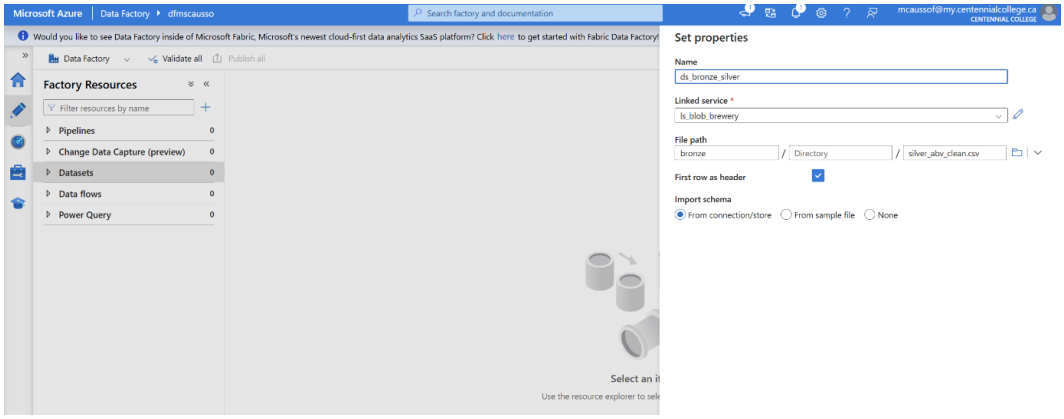


**Figure 5.5f: Azure Data Factory datasets creation**

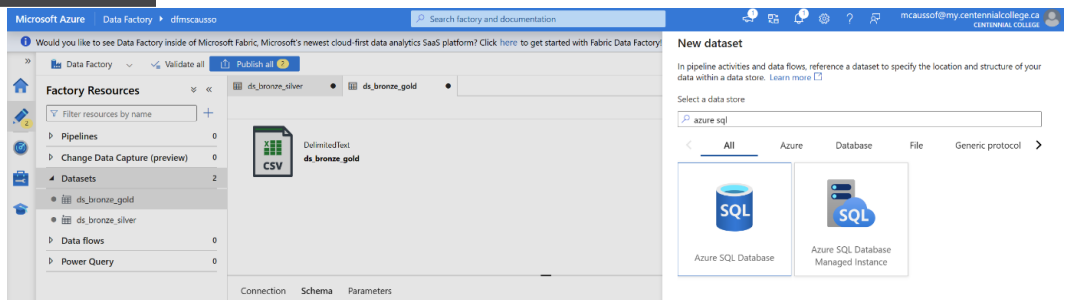


**Figure 5.5g: Azure Data Factory SQL datasets creation**

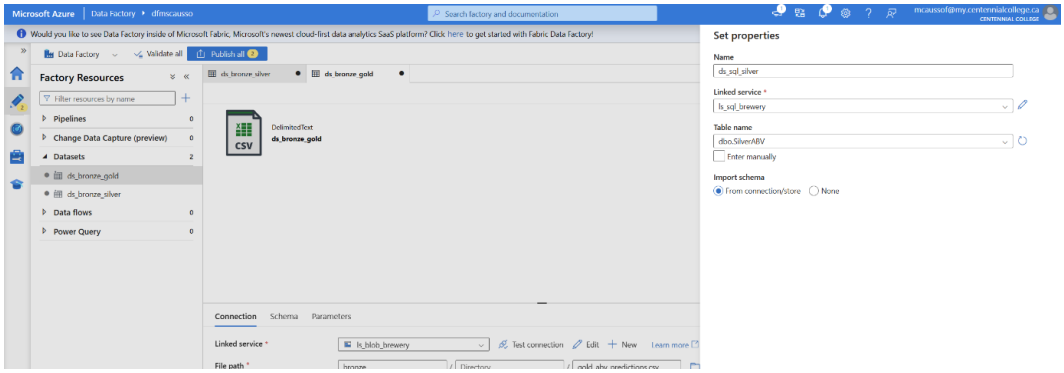


**Figure 5.5h: Azure Data Factory CSV creation**

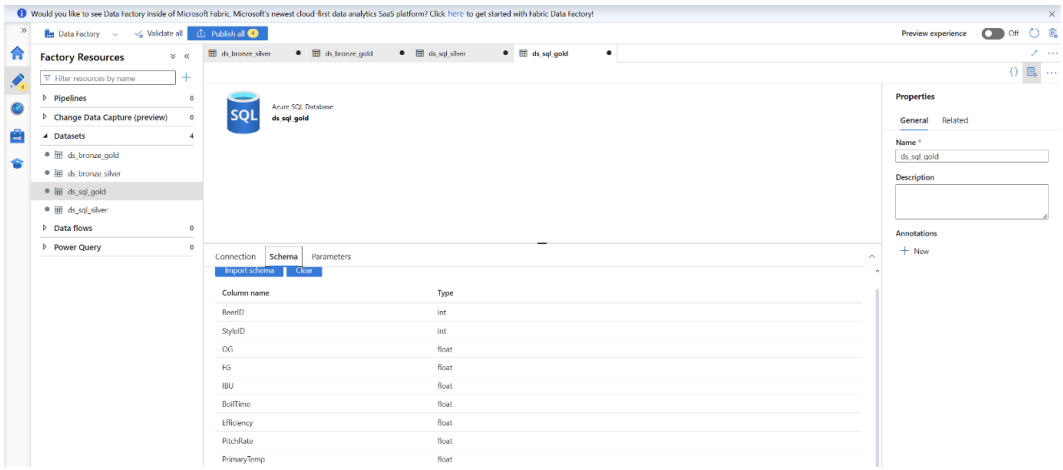


**Figure 5.5i: Azure Data Factory SQL dataset overview**

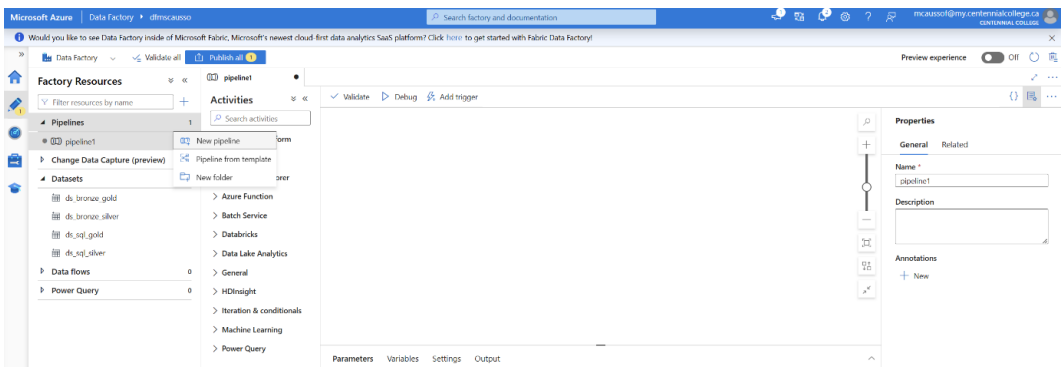


**Figure 5.5a: Azure Data Factory pipeline creation**

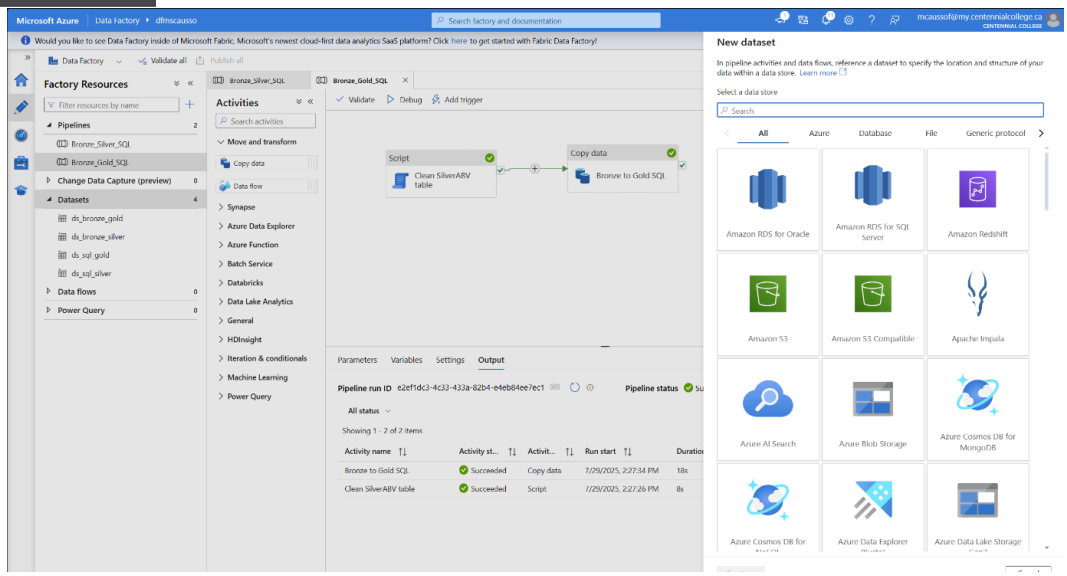


**Figure 5.5a: Azure Data Factory pipeline overview**

## 5.6 Integration with Power BI

The reporting and visualization layer of the Brewery ABV Prediction Project was implemented using Microsoft Power BI, connected directly to the **Gold layer** of the Azure SQL Database. This layer contains curated, analytics-ready datasets, including the predicted ABV values generated by the machine learning models. By sourcing data from the Gold layer, Power BI ensures that all visualizations reflect standardized, validated, and up-to-date information.

The connection was established through a secure SQL database connector, using the configured server and database credentials. Once connected, the curated tables were imported into Power BI. Measures, calculated columns, and visualizations were then developed to present the model predictions, feature importance insights, and style-based ABV patterns. Scheduled refreshes were set up to match the Gold layer's update frequency, ensuring that stakeholders always have access to the latest results without any manual intervention. This integration allows seamless transformation of analytical outputs into interactive dashboards that support decision-making and facilitate the communication of project outcomes.

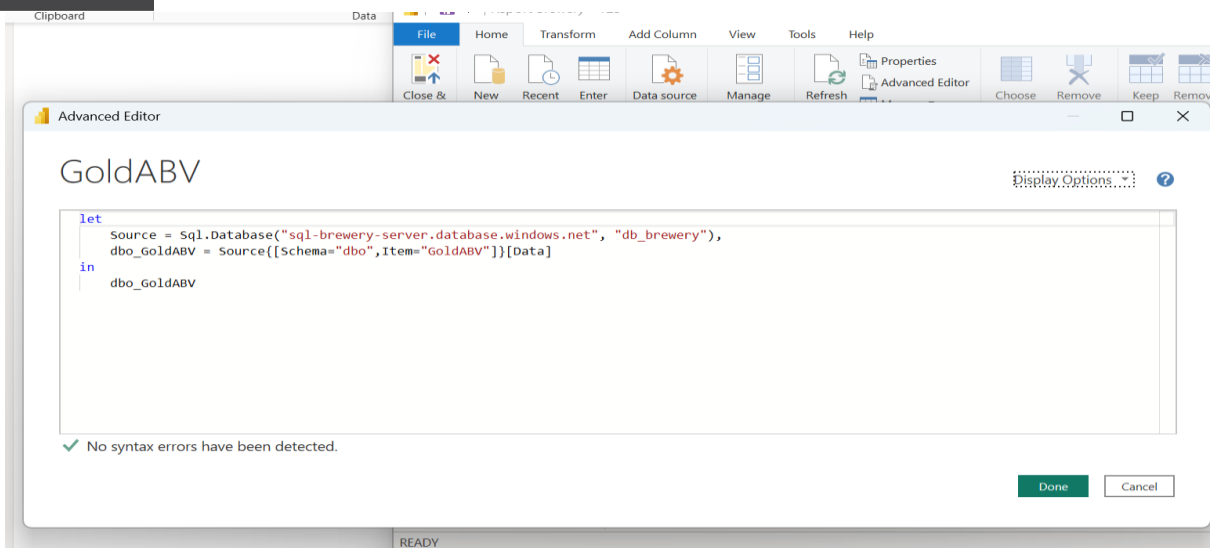


**Figure 5.6a: Integrate with Azure SQL Database in Power BI to create reports**

## 5.7 Security & Governance

The project's data governance is anchored in the Medallion architecture implemented on Azure: Bronze preserves raw inputs as received, Silver standardizes and cleanses entities with consistent types and naming, and Gold curates analytics-ready tables in Azure SQL Database for reporting. This layered design is our first line of governance because it separates raw evidence from transformed facts and business-ready outputs, making lineage, quality checks, and accountability explicit across the flow from Blob Storage into SQL. In practice, Bronze is immutable and timestamp-partitioned for traceability; Silver applies schema enforcement, deduplication, and text standardization; and Gold publishes narrow, consumable tables (including predicted ABV) that Power BI reads directly.

Access to data is controlled with least-privilege principles at each layer. Storage containers are private, and only the ADF managed connections and designated developers can access Bronze/Silver paths; Gold tables in Azure SQL are exposed to reporting through a dedicated credential (currently SQL authentication, with Microsoft Entra ID planned as the target control when tenant permissions allow). Firewall rules on the SQL server restrict ingress to approved clients, and passwords are rotated when administrative actions occur. These controls ensure only authorized identities can move data across layers, while keeping the reporting surface area limited to the curated Gold schema.

Governance also covers documentation, change control, and validation. For documentation, the solution records model purpose and design, key drivers and assumptions, data needs, operating procedures, security/change procedures, and validation plans—elements aligned with model-documentation guidance cited in your course materials. This documentation ensures that any new user can understand the model's purpose and design, the key assumptions and drivers, the operating procedures, and the validation plans. A new stakeholder could safely take operational responsibility for the solution over time in a manner appropriate to the model's use-case and complexity (as advised in BA723 documentation). In the

same spirit, the project explicitly defines "model risk" in simple terms—i.e., the potential adverse consequences from incorrect model selection, implementation, or use—and mitigates it by constraining complexity, clarifying assumptions, and limiting the model's use to well-defined ABV prediction tasks with curated outputs (BA723 documentation). These action thresholds and monitoring practices come directly from the governance guidelines provided in the course materials, consistent with the nominal/ordinal monitoring guidance (BA723 documentation).

Ongoing monitoring is baked into governance. Beyond the initial fit statistics you reported (e.g., $R^2$ and MSE for the Random Forest), the project adopts drift surveillance as data and usage evolve. For continuous targets and distributions that matter to ABV predictions, we use Population Stability Index (PSI) to quantify how much the outcome or key input distributions shift compared to the training baseline; PSI is computed as $\Sigma(\text{Actual\%} - \text{Expected\%}) \times \ln(\text{Actual\%}/\text{Expected\%})$ (BA723 documentation). A PSI below 0.10 indicates no material change, 0.10–0.20 calls for judgment and closer watch, and ≥0.20 triggers retraining or a rebuild; these action thresholds come directly from the governance references in your assignment. BA723 Documentation Pre… For categorical drivers (e.g., style group), we extend monitoring with chi-square tests on build vs. holdout distributions to catch non-stationarity in class mixtures, consistent with the course guidance on nominal/ordinal monitoring (BA723 Documentation).

At the variable level, governance defines inputs, acceptable ranges, and exception handling so the pipelines can fail fast on schema breaks while imputing or capping where appropriate. For interval/ratio variables we track characteristic stability (CSI—same calculation as PSI) alongside summary stats; for nominal variables we track reference distributions and use chi-square for shifts. This variable-level monitoring pairs with your layer checks (row counts, null-rates, and schema validation in Silver) so that issues are detected before they reach Gold and Power BI (BA723 Documentation).

Finally, data governance ties back to decision governance through risk tiering and documented operating procedures. The model and its dashboards are classified as low-to-moderate impact for educational and internal analytical use, but the same framework scales: as complexity, uncertainty, coverage, or potential impact grows, the required documentation depth, validation rigor, and approval workflow grow with it (BA723 Documentation). In sum, the combination of Medallion separation, restricted access to curated Gold tables, documented procedures, and scheduled drift/variable monitoring provides a pragmatic governance baseline for your Azure implementation—strong enough to keep the project safe and explainable today, and structured enough to extend as the scope or stakes increase.

## 5.8 Benefits & Scalability Considerations

Deploying the Brewery ABV Prediction Project on Azure delivers immediate operational benefits. The Medallion layout in Blob Storage separates raw, standardized, and curated assets, so each audience works at the right level of quality: ingestion and experimentation in Bronze/Silver, and trusted consumption

from Gold. Parameterized ADF pipelines make the flow repeatable and auditable; a run today or next month follows the same steps with only dates and file names changing. Publishing the curated outputs to Azure SQL Database under a stable schema lets Power BI connect to one source of truth for predicted ABV and supporting attributes, which simplifies refresh, governance, and troubleshooting.

Performance and cost are controlled by design. Storing intermediate data as Parquet in Silver reduces I/O and speeds transforms, while the SQL Gold layer provides fast, indexed queries for BI. Because ADF uses Copy and lightweight transforms, throughput scales with data volume without re-engineering the flow. As data volume grows and refresh windows lengthen, dataset imports in Power BI can adopt incremental refresh (partitioning data by date) to keep refresh times predictable and costs low. The architecture also supports pay-as-you-go scaling: storage grows with data, ADF charges per activity/runtime, and the SQL tier can be right-sized or paused/moved to an elastic pool when appropriate.

The solution is ready to scale up in several directions. As source data expands to tens of millions of rows, the Silver zone can increase partitioning by load date and entity, and the Gold layer can introduce indexed views or summary tables for the heaviest visuals. If analytical concurrency or query latency becomes a constraint, the curated layer can be lifted to a bigger Azure SQL tier or shifted to a warehouse engine (e.g., Synapse/Dedicated SQL) without changing upstream contracts. For model lifecycle, the same ADF orchestration can schedule periodic scoring jobs and retraining. The drift thresholds defined in governance (e.g., retrain when PSI $\geq$ 0.20) translate naturally into pipeline triggers that promote a new model only after validation, keeping predictions reliable as distributions evolve.

Security and reliability scale with the workload as well. Private containers and limited SQL surface area already minimize exposure; as the footprint grows, credentials can be centralized in Azure Key Vault, the SQL database can be moved behind private endpoints, and Microsoft Entra ID can replace SQL authentication when tenant permissions allow—all while preserving the same role-based separation of access across the Bronze, Silver, and Gold layers. Backups, soft-delete, and point-in-time restore in SQL provide operational resilience; if higher availability is needed, geo-redundant storage and secondary regions can be added without altering the data contracts that Power BI and downstream users rely on.

Finally, the design preserves flexibility for future features. If near-real-time predictions are required, the current batch scoring can be complemented with a small event path (e.g., Functions or a lightweight API) that writes results back to Gold tables, so reports remain unchanged. If collaboration and release management become more formal, the ADF assets can adopt CI/CD (dev/test/prod) and parameterized environments with the same pipelines. In short, the current implementation already delivers governed, fast, and affordable analytics for ABV prediction—and it provides clear, low-risk steps to scale volume, users, and model sophistication without rewriting the platform.

CENTENNIAL COLLEGE

# REFERENCES


ABVCalculator. (n.d.). *ABV calculator for beer, wine, mead, cider, and cocktails*. ABVCalculator.ca. https://www.abvcalculator.ca/

Government of Canada, Canadian Food Inspection Agency. (2025, January 15). *Labelling requirements for alcoholic beverages*. https://inspection.canada.ca/en/food-labels/labelling/industry/alcoholic-beverages

Liquor Control Board of Ontario. (2025, April 9). *Manufacturers – Actual vs Declared Alcohol Content – Guideline*. HelloLCBO. https://hellolcbo.com/app/answers/detail/a_id/1743/~/manufacturers---actual-vs-declared-alcohol-content---guideline

Perennia Food and Agriculture Inc., & Nova Scotia Liquor Corporation. (2025). *Factsheet: Alcohol by volume* [PDF]. https://www.mynslc.com/-/media/NSLC/MyNSLC/Trade-MyNSLC/Product-Testing/FY26/Alcohol-by-Volume-Factsheet.pdf

*BA723 documentation presentation governance M25 [Lecture slides]. (2025). Centennial College.*

Trofe, J. (2016). *Brewer's Friend beer recipes* [Data set]. Kaggle. https://www.kaggle.com/datasets/jtrofe/beer-recipes

Alcohol and Tobacco Tax and Trade Bureau. (2023). Malt beverage labeling: Alcohol content (27 CFR 7.65). https://www.ttb.gov/regulated-commodities/beverage-alcohol/beer/labeling/malt-beverage-alcohol-content

Liquor Control Board of Ontario. (n.d.). Product testing | Doing business with LCBO. https://www.doingbusinesswithlcbo.com/content/dbwl/en/basepage/home/quality-assurance/quality-assurance-policies---guidelines/product-testing.html

Brewer's Friend. (2023). Batch statistics for extract base recipes. https://www.brewersfriend.com/extract-ogfg/