Latent Variable Regression on Wine Quality Data

Jeewon Han
Department of Applied Mathematics
Johns Hopkins University
Baltimore, MD, 21218
jhan67@jhu.edu

Lucas Rozendaal
Department of Applied Mathematics
Johns Hopkins University
Baltimore, MD, 21218
lrozend1@jhu.edu

Abstract

This paper explores the application of a classifier based on physicochemical properties measured during certification to predict wine quality. The dataset used is a subset of the UCI Wine Quality dataset, focusing on red variants of Portuguese "Vinho Verde" wine. The research introduces Bayesian methods, specifically Bayesian linear regression and ordered probit regression, to analyze the dataset, providing a novel perspective in comparison to the frequentist and machine learning approaches in the literature. The analysis covers both the original 0-10 quality ratings and a binned 1-3 scale, offering insights into the performance of the models across different categorizations. The results demonstrate the effectiveness of Bayesian ordered probit regression over Bayesian linear regression and frequentist methods, with higher accuracy in predicting wine quality, especially for the binned target variable. While Bayesian methods slightly outperform some machine learning models, overall prediction accuracy remains modest, suggesting that the physicochemical properties alone might not fully capture the complexity of wine quality determination.

1 Purpose

Wine certification and assessment are two critical components of the wine production process. During certification, physicochemical tests are performed to ensure the wine adheres to various health standards. During assessment, wine tasters rate the wine, classifying the wine by quality. Wine quality assessment is a costly and unreliable manual process since it depends so heavily on human taste (Cortez et al., 2009). As such, a classifier based on the physicochemical properties of wine, already measured at the certification step, can prove to be cost-effective and beneficial. For wine producers, such a classifier can help with labor costs and manufacturing speed; wine consumers stand to benefit from consistent pricing based on measured quality.

1.1 Data Source

The dataset of interest is a subset of the Wine Quality dataset from the UCI Machine Learning Repository. The dataset contains the physicochemical properties of red variants of Portuguese “Vinho Verde” wine as well as an ordinal “quality” variable rated on a scale of 0-10 based on sensory data. Paulo Cortez et al. (2009), who originated this data, conduct data mining on a larger dataset of both red and white wine, but in this paper we limit ourselves to the 1,599 entries on red wine for both

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Table 1: Variables in Dataset

Variable Name	Role	Type	Description
fixed_acidity	Feature	Continuous	
volatile_acidity	Feature	Continuous	
citric_acid	Feature	Continuous	
residual_sugar	Feature	Continuous	
chlorides	Feature	Continuous	
free_sulfur_dioxide	Feature	Continuous	
total_sulfur_dioxide	Feature	Continuous	
density	Feature	Continuous	
pH	Feature	Continuous	
sulphates	Feature	Continuous	
alcohol	Feature	Continuous	
quality	Target	Integer	score between 0 and 10
color	Other	Categorical	red or white

simplicity and better alignment with previous research. The variables in the dataset are described in Table 1.

2 Literature Review

2.1 Wine Quality

The wine quality data available through UCI is one of the only, if not *the* only, publicly-available dataset on the physicochemical properties of wine and wine quality. As such, almost all research on the relationship between these properties of wine and its quality are based upon this dataset.

Cortez et al. (2009) perform analyses using multiple regression (MR), neural networks (NN), and support vector machines (SVM) with variable selection through sensitivity analysis for both the red and white wines. They find that a SVM model performs the best, boasting an accuracy score of 0.624 for red wine. This figure increases to 0.890 when widening the range of acceptance to one quality level above or below the correct level. They note this model much outperforms a random classifier. The NN and MR models perform similarly to each other but worse than the SVM model (with the NN model being slightly more accurate); however, the benefit of the MR model is its interpretability.

A modified version of the red wine dataset exists as part of a Kaggle competition, in which users of the website are given pre-determined training and test sets. While the results are not directly comparable to Cortez et al.'s (2009) findings due to the differences in the data, the most successful entries use some combination of XGBoost and NN with a myriad of other models in ensemble. The accuracy scores of these models range from 0.594 to 0.597. While the documentation for the competition entries is not very robust, one of the top entries notes that engineered features involving fixed acidity and density provided lift during modeling.

Independent analyses outside of Kaggle use similar approaches, including MR, regularization, and random forest. Of the three, Dexter Nguyen (2020) finds random forest performs the best with an R^2 of 0.485, with the features of volatile acidity, citric acid, sulphates, and alcohol being the most predictive of wine quality. An analysis at Penn State University also finds a random forest model to be the most effective from a selection of models with an accuracy score of 0.677 but ultimately concludes that wine quality is not well supported by its physicochemical properties. They find alcohol, volatile acidity, and density to be most important predictors in their analysis. These results are summarized in Table 2.

2.2 Methodology

While conducting the literature review, we did not notice any Bayesian methods of significance being used in the analysis of the data, so we wanted to explore various types of Bayesian regression in our analysis.

One of the largest challenges of the project is designing a reasonable approach to model selection for those Bayesian regressions, given that we have a large number of potential predictors and interaction

Table 2: Comparison of Approaches

Source	Best Primary Approach	Most Important Features	Accuracy	R^2
Cortez et al. (2009)	SVM	-	0.624	-
Nguyen (2020)	Random Forest	volatile acidity citric acid sulphates alcohol	-	0.485
Penn State University (n.d.)	Random Forest	volatile acidity density alcohol	0.677	-
Medeiros (2023) - Kaggle #1	XGBoost	-	0.597	-
NHopeT (2023) - Kaggle #2	XGBoost	-	0.596	-
Vandewiele (2023) - Kaggle #3	Weighted Mode Ensemble Model	-	0.596	-
Thomas (2023) - Kaggle #4	Stacked Ensemble Model	fixed acidity density	0.594	-

terms. George and McCulloch (1993) propose an approach to finding relatively high probability Bayesian linear models in instances when it may not be computationally feasible to calculate the true highest probability model. Their approach is to introduce a latent variable $\gamma_i \in \{0, 1\}$, such that each coefficient β_i is parameterized as $b_i \times \gamma_i$ (i.e. the latent variable γ_i is an indicator variable that can be used to include or exclude each b_i). They then use Gibbs sampling to indirectly sample from the posterior distribution. The authors observe that higher probability models "can be identified by their more frequent appearance in the Gibbs sample" (p. 881). We intend to employ their strategy in our Bayesian linear model selection and use the selected models for both our Bayesian linear regression and our Bayesian ordered probit regression.

Quinn (2004) designs a factor analysis MCMC algorithm that allowed him to accommodate both ordinal variables and continuous data. Quinn's approach allows him to predict on ordinal variables without having to treat those variables as continuous, which could result in "falsely precise and possibly biased estimates" (p. 339). We intend to use a somewhat similar model during our Bayesian ordered probit regression, although we intend to also perform a Bayesian linear regression to compare the two approaches.

3 Method

3.1 Research Question

The primary goal of our project is to model how chemical variables, such as acidity levels and alcohol content, correlate with the observed rankings of wine quality across different varieties of *Vinho Verde* red wine.

We expand upon the prior research on this dataset by attempting to predict wine quality using both Bayesian linear regression models and Bayesian ordered probit regression models. We compare these results with the results from various frequentist linear and ordered probit models. Because the target variable is ordinal, it makes sense to try ordered probit regressions. However, because the target variable is a rating from 0-10, it is also, in a sense, numerical, and therefore we felt it would be worthwhile to try linear models for comparison.

For each type of model, we intend to perform two sets of analyses: one for the original target variable, a quality rating from 0-10, and another for a binned target variable from 1-3. We created this binned target variable using the following rule: quality ratings less than 5 were assigned into bin 1, quality ratings equal to 5 were assigned into bin 2, and quality ratings greater than 5 were assigned to bin 3. We believe that the frequentist and Bayesian ordered probit regression models will perform significantly better on this binned target variable with fewer categories.

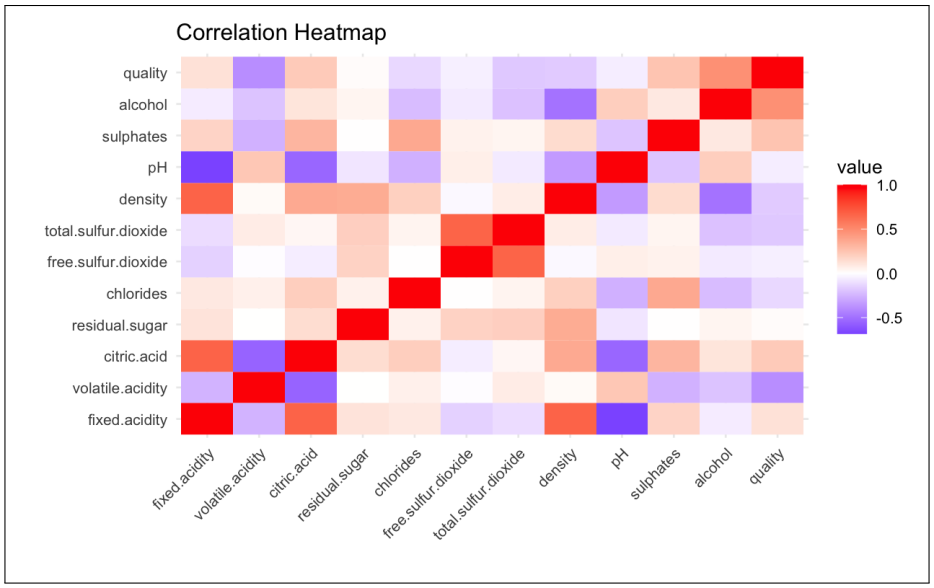


Figure 1: Correlation heatmap for the Wine Quality dataset.

Before we can perform our regression, we needed to carry out model selection for each of the modeling approaches that we will be using in order to create the most parsimonious model. We devised model selection approaches for all of the frequentist models and for the Bayesian linear model selection. Initially, we intended to carry out Bayesian model selection on our ordered probit regression model. However, it was not computationally feasible to do model selection for Bayesian ordered probit regression. Instead, we simply settled for using the model from our Bayesian linear regression for our Bayesian ordered probit regression.

3.2 Priors

For our Bayesian linear regression model, we simply used uninformative default priors. We set $g = n, \nu_0 = 1$. We calculated $\hat{\sigma}^2_{ols}$ and set $\sigma^2_0 = \hat{\sigma}^2_{ols}$.

For our ordered probit regression model, we used the prior

$$\beta \sim MVN(0, n(X^T X)^{-1}).$$

We also needed to define a $p(g)$ constrained so that $g_1 < ... < g_{K-1}$. Hoff (2009) notes that that coming up with a prior distribution for g that represents actual prior information is non-trivially difficult. As such, we used the arbitrary prior for $p(g)$ established in Hoff (2009).

4 Analysis and Results

4.1 Exploratory Data Analysis

The correlation heatmap for the Wine Quality dataset is depicted in Figure 1.

A few trends stand out from the heatmap. First, though we will perform more rigorous model selection later, there are a few variables that seem to be correlated with overall quality, namely volatile.acidity and alcohol, which also aligns with our literature review. Second, there seems to be a problem of multicollinearity with many of the variables. For example, pH and fixed.acidity have a strong negative correlation. That correlation makes sense because those two chemical properties are directly related. To mitigate the impact of multicollinearity, we chose to exclude a few of the heavily correlated variables (discussed further in the Data Preprocessing subsection).

Our target variable for wine quality ratings, which is named "quality" in the dataset, has data values ranging from 3 to 8, as depicted in the Figure 2. The data appears to have a roughly normal distribution, although it is slightly left-skewed because there are not many values lower than 5. Notably, because

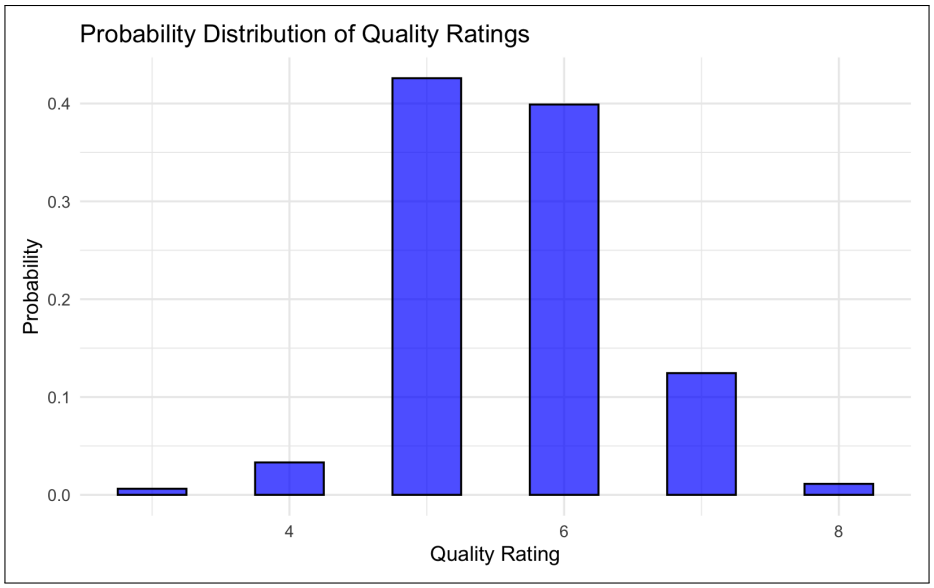
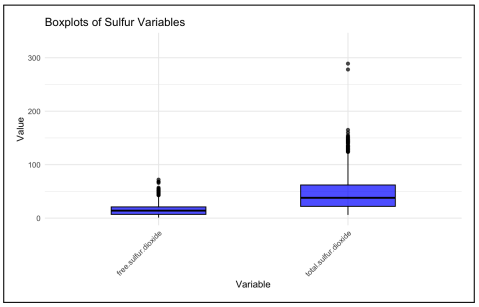
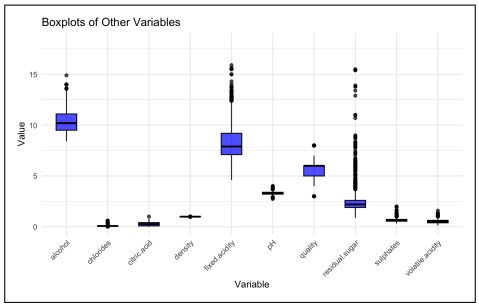


Figure 2: Probability distribution for the "quality" variable.



(a) Boxplots of sulfur variables



(b) Boxplots of other variables

Figure 3: Boxplots of variables in the wine quality dataset

there are no examples of wine in the dataset that are rated worse than a 3 or better than an 8, our ordinal regression model will only be able to predict within the 3-8 bins. The lack of especially high or low values was part of the justification for our decision to try to model with a binned target variable of three ratings; we're interested to see if our model can, at least, predict whether a wine will be generally good, generally bad, or average.

Additional, from the box plots of our predictor variables (Figure 3), it is apparent that the scale of the predictor variables varies enough that the data should be normalized prior to modeling.

4.2 Data Preprocessing

All the variables other than our response variable for quality were normalized. The data was split into train and test sets, with 80% of the data being used in the train set.

To reduce the impact of multicollinearity on our results, we dropped a few highly correlated predictor variables, instead choosing to begin our model selection process with the set of all non-highly correlated variables {residual.sugar, chlorides, total.sulfur.dioxide, density, pH, sulphates, alcohol} and their interaction terms.

4.3 Preliminary Frequentist Analysis

Before conducting the Bayesian ordinal probit regression outlined in the Methodology section, we conducted preliminary analyses using the frequentist methods of ordinal regression and multiple linear regression, which we compared to baseline models. We conducted two sets of analyses: one for the original target variable, a quality rating from 0-10, and another for the binned target variable from 1-3. The baseline model for each set assigned the predicted target variable to be the mode from the train set. The most parsimonious models were found through backwards elimination. We used the Akaike information criterion (AIC) in backwards elimination for MR. For ordinal regression, we devised our own algorithm for eliminating the variable yielding the highest increase in accuracy when dropped as eliminating using AIC did not result in any improvement in accuracy. Results are summarized in Table 3. See Appendix A for code.

Table 3: Comparison of Frequentist Approaches

Primary Approach	Target Variable	Most Important Features (Excluding Interaction Terms)	Best Accuracy Achieved
Baseline Model	Quality 0-10	-	0.421
Ordinal Regression	Quality 0-10	chlorides total.sulfur.dioxide alcohol	0.028
Multiple Linear Regression	Quality 0-10	residual.sugar chlorides total.sulfur.dioxide density pH sulphates alcohol	0.494
Baseline Model	Binned Quality 1-3	-	0.536
Ordinal Regression	Binned Quality 1-3	chlorides sulphates total.sulfur.dioxide pH density alcohol	0.730
Multiple Linear Regression	Binned Quality 1-3	residual.sugar chlorides total.sulfur.dioxide density pH sulphates alcohol residual.sugar	0.470

4.4 Bayesian Analysis

Before we could perform our Bayesian linear regression and Bayesian ordered probit regression, we needed to perform model selection to decide which variables to include. For the Bayesian linear regression, we wanted to test models with all the non-highly correlated variables and their interaction terms, a total of 29 regression coefficients. As such, it would have been impractical for us to compute the marginal probability of each of the 2^{29} models. Instead, we used a Gibbs sampler with 10,000 iterations to select a relatively high-probability model. To do so, we averaged the probability of the latent variables across the iterations of the Gibbs samplers. Then, if the probability was above 0.5, we included the corresponding predictor variable in the model; otherwise, we excluded it. As mentioned in the Methodology section, however, we did not have a computationally viable way of performing model selection for Bayesian ordered probit regression. Instead, we simply used the average model from our Bayesian linear model selection for both the Bayesian linear regression and the Bayesian ordered probit regression.

Once we had the model that we wanted to test, we simply carried out the ordered probit regression and linear regression for both our original target variable and the binned target variable. Results are summarized in Table 4. See Appendix B for code.

Table 4: Comparison of Bayesian Approaches

Primary Approach	Target Variable	Features Used	Best Accuracy Achieved
Baseline Model	Quality 0-10	-	0.421
Ordered Probit Regression	Quality 0-10	chlorides total.sulfur.dioxide pH sulphates alcohol residual.sugar*alcohol total.sulfur.dioxide*sulphates pH*sulphates	0.601
Multiple Linear Regression	Quality 0-10	chlorides total.sulfur.dioxide pH sulphates alcohol residual.sugar*alcohol total.sulfur.dioxide*sulphates pH*sulphates	0.491
Baseline Model	Binned Quality 1-3	-	0.536
Ordered Probit Regression	Binned Quality 1-3	total.sulfur.dioxide pH sulphates alcohol residual.sugar*alcohol chlorides*sulphates total.sulfur.dioxide*sulphates	0.718
Multiple Linear Regression	Binned Quality 1-3	total.sulfur.dioxide pH sulphates alcohol residual.sugar*alcohol chlorides*sulphates total.sulfur.dioxide*sulphates	0.464

On both the full quality target variable and the binned target variable, our Bayesian ordered probit models outperformed the Bayesian linear models. The prediction accuracy of our Bayesian ordered

probit model on the binned target variable was marginally worse than that of our frequentist ordered probit regression. However, the accuracy of our Bayesian ordered probit regression was significantly better on the full quality target variable than our Frequentist ordered probit regression had been.

Our prediction accuracy of 0.601 is somewhat worse than the best results from the literature review (Penn State University's random forest model, which had a prediction accuracy of 0.677), but it is ever so slightly better than any of ensemble model, weighted mode, or XGBoost approaches from Kaggle. In any case, the Bayesian methods used in our analyses perform on par with Frequentist and machine learning approaches. However, none of these approaches thus far have done exceptionally well at predicting overall wine quality. Therefore, either physiochemical properties do not provide sufficient information to make highly accurate predictions about wine quality, or else an optimal model has yet to be found.

5 References

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4), 547-553. doi:10.1016/j.dss.2009.05.016

George, E. I., & McCulloch, R. E. (1997). Approaches for bayesian variable selection. *Statistica Sinica, 7*(2), 339-373. Retrieved from JSTOR database. Retrieved from http://www.jstor.org/stable/24306083

Hoff P. D. (2009). *A first course in bayesian statistical methods.* Springer. https://doi.org/10.1007/978-0-387-92407-6.

Nguyen, D. (2020). (rep.). *Red Wine Quality Prediction Using Regression Modeling and Machine Learning.* Retrieved from https://github.com/dexterngn/Red-Wine-Quality-Prediction-Using-Regression-Modeling-and-Machine-Learning/blob/main/Individual

Ordinal Regression with a Tabular Wine Quality Dataset. Kaggle. (2023, January 30). https://www.kaggle.com/competitions/playground-series-s3e5/leaderboard

The Pennsylvania State University. (n.d.). *Analysis of Wine Quality Data: Stat 508*. Penn State Eberly College of Science. https://online.stat.psu.edu/stat508/lesson/analysis-wine-quality-data

Quinn, K. M. (2017). Bayesian Factor Analysis for Mixed Ordinal and Continuous Responses. *Political Analysis, 12*(4), 338-353. doi:10.1093/pan/mph022

6 Appendices

6.1 Appendix A: R Code for Frequentist Analysis

```
```{r, include=FALSE}
library("latex2exp")
library("ggplot2")
library("MASS")
#library("MCMCpack")
library("coda")
library("dplyr")
library("tidyr")
library("caret")
library("reshape2")
library("DescTools")
```

```{r, warning=FALSE, message=FALSE}
### Exploratory Data Analysis
# Import Data
data <- read.csv("winequality-red.csv", sep=';')
quality_column <- data$quality
summary_stats <- summary(data)
features <- c("fixed.acidity", "volatile.acidity", "citric.acid",
              "residual.sugar", "chlorides", "free.sulfur.dioxide"
                  ,
              "total.sulfur.dioxide", "density", "pH", "sulphates"
                  ,
              "alcohol")
all_features <- c("fixed.acidity", "volatile.acidity", "citric.
    acid",
              "residual.sugar", "chlorides", "free.sulfur.dioxide"
                  ,
              "total.sulfur.dioxide", "density", "pH", "sulphates"
                  ,
              "alcohol", "quality")

# Scatterplots
pairwise_scatterplots <- pairs(data[, all_features])

# Heatmap
correlation_matrix <- cor(data[, all_features])
correlation_long <- melt(correlation_matrix)
heatmap <- ggplot(data = correlation_long, aes(x = Var1, y = Var2,
     fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red",
      midpoint = 0) +
  theme_minimal() +
  labs(title = "Correlation␣Heatmap", x = "", y = "") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Histograms for Continuous Variables
histograms <- lapply(features, function(var) {
  ggplot(data, aes(x = get(var))) +
    geom_histogram(binwidth = 1, fill = "blue", color = "black",
        alpha = 0.7) +
    labs(title = paste("Histogram␣of", var), x = var, y = "
        Frequency") +
```
```

```
    theme_minimal()
})

histograms_quality <- ggplot(data, aes(x = data$quality)) +
    geom_histogram(bins=10, fill = "blue", color = "black", alpha
        = 0.7) +
    labs(title = "Histogram of Quality", x = "Quality", y = "
        Frequency") +
    theme_minimal()

# Boxplots - Separated Out Sulfur Variables Since Scales Much
    Larger
sulfur_variables <- c("total.sulfur.dioxide", "free.sulfur.dioxide
    ")
other_variables <- setdiff(all_features, sulfur_variables)

# Boxplot for Sulfur Variables
boxplot_sulfur <- ggplot(data %>% pivot_longer(cols = sulfur_
    variables),
                         aes(x = name, y = value)) +
  geom_boxplot(fill = "blue", color = "black", alpha = 0.7, width
      = 0.5) +
  labs(title = "Boxplots of Sulfur Variables", x = "Variable", y =
       "Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.2)))

# Boxplot for Other Variables
boxplot_other <- ggplot(data %>% pivot_longer(cols = other_
    variables),
                        aes(x = name, y = value)) +
  geom_boxplot(fill = "blue", color = "black", alpha = 0.7, width
      = 0.5) +
  labs(title = "Boxplots of Other Variables", x = "Variable", y =
      "Value") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.2)))

#Plot Probability Distributions
rank_probs <- prop.table(table(data$quality))
plot_data <- data.frame(Rank = as.numeric(names(rank_probs)),
    Probability = as.vector(rank_probs))
data_prob <- ggplot(plot_data, aes(x = Rank, y = Probability)) +
  geom_bar(stat = "identity", fill = "blue", color = "black",
      alpha = 0.7, width = 0.5) +
  labs(title = "Probability Distribution of Quality Ratings",
       x = "Quality Rating",
       y = "Probability") +
  theme_minimal()

# Display Summary Statistics
print(summary_stats)

# Display Scatterplots
print(pairwise_scatterplots)

# Display Heatmap
```

```
print(heatmap)

# Display Histograms
for (hist_plot in histograms) {
  print(hist_plot)
}
print(histograms_quality)

# Display Boxplot
print(boxplot_sulfur)
print(boxplot_other)

# Display Probability Distribution
print(data_prob)

```

```{r, warning=FALSE, message=FALSE}
### 0-10 Target Variable

# Standardize Data
standardized_data <- as.data.frame(scale(data[, -which(names(data)
    == "quality")]))
standardized_data <- cbind(quality = quality_column, standardized_
    data)

# Train/Test Split
set.seed(123)  # Set seed for reproducibility
split_index <- createDataPartition(standardized_data$quality, p =
    0.8, list = FALSE)
train_data <- standardized_data[split_index, ]
test_data <- standardized_data[-split_index, ]

# Baseline Model
most_common_quality <- Mode(train_data$quality)
baseline_prediction <- rep(most_common_quality, nrow(test_data))
predicted_labels_baseline <- as.integer(baseline_prediction)
true_labels <- as.integer(test_data$quality)

# Accuracy
accuracy_baseline <- sum(predicted_labels_baseline == true_labels)
    / length(true_labels)
cat("Accuracy for the baseline model:", accuracy_baseline, "\n")

# Ordinal Regression

# Backward elimination for the most parsimonious model
# Non-correlated features
features_reg <- c("residual.sugar", "chlorides",
              "total.sulfur.dioxide", "density", "pH", "sulphates"
                  ,
              "alcohol")
all_features <- setdiff(features_reg, "quality")
# Interaction Terms
selected_features <- c(all_features, "residual.sugar*chlorides", "
    residual.sugar*total.sulfur.dioxide",
                       "residual.sugar*density", "residual.sugar*
                           pH", "residual.sugar*sulphates",
```

```
                        "residual.sugar*alcohol", "chlorides*total.
                            sulfur.dioxide",
                        "chlorides*density", "chlorides*pH", "
                            chlorides*sulphates", "chlorides*
                            alcohol",
                        "total.sulfur.dioxide*density", "total.
                            sulfur.dioxide*pH",
                        "total.sulfur.dioxide*sulphates", "total.
                            sulfur.dioxide*alcohol",
                        "density*pH", "density*sulphates", "density
                            *alcohol", "pH*sulphates",
                        "pH*alcohol", "sulphates*alcohol")
best_accuracy <- 0
best_feature_set <- NULL
best_num_variables <- Inf  # Initialize with a large value

while (length(selected_features) >= 1) {
  current_accuracy <- 0
  current_num_variables <- length(selected_features)
  worst_feature <- NULL
  exit_loop <- FALSE  # Flag to control loop exit

  for (feature in selected_features) {
    current_features <- setdiff(selected_features, feature)

    if (length(current_features) == 0) {
      exit_loop <- TRUE
      break  # Exit the inner loop when only one feature is left
    }

    # Train the model with the current set of features
    formula_str <- paste("factor(quality) ~ ", paste(current_
        features, collapse = "+"), sep = "")
    model <- polr(as.formula(formula_str), data = train_data, Hess
        = TRUE)

    # Predict using the trained model on the test set
    # (Ensure test_data is also a data frame)
    predicted_labels <- predict(model, newdata = as.data.frame(
        test_data))

    # Convert predicted and true labels to integers (if not
        already)
    predicted_labels <- as.integer(predicted_labels)
    true_labels <- as.integer(test_data$quality)

    # Compute accuracy
    accuracy <- sum(predicted_labels == true_labels) / length(true
        _labels)

    # Update the current accuracy, worst feature, and number of
        variables if needed
    if (accuracy > current_accuracy) {
      current_accuracy <- accuracy
      worst_feature <- feature
      current_num_variables <- length(current_features)
    }
  }
```

```
  if (exit_loop) {
    break  # Exit the outer loop when only one feature is left
  }


  # Remove the worst feature from the selected features
  selected_features <- setdiff(selected_features, worst_feature)

  # Update the best feature set, accuracy, and number of variables
      if needed
  if (current_accuracy > best_accuracy ||
      (current_accuracy == best_accuracy && current_num_variables
         < best_num_variables)) {
    best_accuracy <- current_accuracy
    best_feature_set <- selected_features
    best_num_variables <- current_num_variables
  }

  #cat("Selected features:", selected_features, "\n")
  #cat("Current accuracy:", current_accuracy, "\n")
  #cat("Current number of variables:", current_num_variables, "\n\
     n")
}
cat("Best feature set:", best_feature_set, "\n")
cat("Best accuracy:", best_accuracy, "\n")
cat("Best number of variables:", best_num_variables, "\n")

# Multiple Regression
initial_model <- lm(quality ~ residual.sugar + chlorides + total.
    sulfur.dioxide + density+ pH +
                      sulphates + alcohol + residual.sugar*
                         chlorides +
                      residual.sugar*total.sulfur.dioxide +
                      residual.sugar*density + residual.sugar*pH +
                          residual.sugar*sulphates +
                      residual.sugar*alcohol + chlorides*total.
                         sulfur.dioxide +
                      chlorides*density + chlorides*pH + chlorides
                         *sulphates + chlorides*alcohol +
                      total.sulfur.dioxide*density + total.sulfur.
                         dioxide*pH +
                      total.sulfur.dioxide*sulphates + total.
                         sulfur.dioxide*alcohol +
                      density*pH + density*sulphates + density*
                         alcohol + pH*sulphates +
                      pH*alcohol + sulphates*alcohol, data = train
                         _data)

# Perform backward elimination using stepwise regression
final_model <- step(initial_model, direction = "backward")

# Display the final model
summary(final_model)

X_test <- as.data.frame(test_data[, !grepl("quality", names(test_
    data))])
predicted_labels <- predict(final_model, newdata = X_test)
predicted_labels <- as.integer(predicted_labels)
```

```
# Convert predicted and true labels to integers (if not already)
predicted_labels <- as.integer(predicted_labels)
true_labels <- as.integer(test_data$quality)

# Compute accuracy
accuracy <- sum(predicted_labels == true_labels) / length(true_
    labels)
print(accuracy)
```

```{r, warning=FALSE, message=FALSE}
### 1-3 Target Variable

# Standardize Data
standardized_data <- as.data.frame(scale(data[, -which(names(data)
    == "quality")]))
standardized_data <- cbind(quality = quality_column, standardized_
    data)
standardized_data$ordinal_quality <- cut(standardized_data$quality
    ,
                    breaks = c(-Inf, 4.5, 5.5, Inf),
                    labels = c(1, 2, 3),
                    include.lowest = TRUE)
standardized_data$ordinal_quality <- as.numeric(standardized_data$
    ordinal_quality)
standardized_data <- select(standardized_data, -quality)

# Train/Test Split
set.seed(123)  # Set seed for reproducibility
split_index <- createDataPartition(standardized_data$ordinal_
    quality, p = 0.8, list = FALSE)
train_data <- standardized_data[split_index, ]
test_data <- standardized_data[-split_index, ]

# Baseline Model
most_common_quality <- Mode(train_data$ordinal_quality)
baseline_prediction <- rep(most_common_quality, nrow(test_data))
predicted_labels_baseline <- as.integer(baseline_prediction)
true_labels <- as.integer(test_data$ordinal_quality)

# Accuracy
accuracy_baseline <- sum(predicted_labels_baseline == true_labels)
    / length(true_labels)
cat("Accuracy for the baseline model:", accuracy_baseline, "\n")

# Ordinal Regression
# Backward elimination for the most parsimonious model
# Non-correlated features
features_reg <- c("residual.sugar", "chlorides",
              "total.sulfur.dioxide", "density", "pH", "sulphates"
                  ,
              "alcohol")
all_features <- setdiff(features_reg, "ordinal_quality")
# Interaction Terms
selected_features <- c(all_features, "residual.sugar*chlorides", "
    residual.sugar*total.sulfur.dioxide",
                       "residual.sugar*density", "residual.sugar*
                           pH", "residual.sugar*sulphates",
```

```
                        "residual.sugar*alcohol", "chlorides*total.
                            sulfur.dioxide",
                        "chlorides*density", "chlorides*pH", "
                            chlorides*sulphates", "chlorides*
                            alcohol",
                        "total.sulfur.dioxide*density", "total.
                            sulfur.dioxide*pH",
                        "total.sulfur.dioxide*sulphates", "total.
                            sulfur.dioxide*alcohol",
                        "density*pH", "density*sulphates", "density
                            *alcohol", "pH*sulphates",
                        "pH*alcohol", "sulphates*alcohol")
best_accuracy <- 0
best_feature_set <- NULL
best_num_variables <- Inf  # Initialize with a large value

while (length(selected_features) >= 1) {
  current_accuracy <- 0
  current_num_variables <- length(selected_features)
  worst_feature <- NULL
  exit_loop <- FALSE  # Flag to control loop exit

  for (feature in selected_features) {
    current_features <- setdiff(selected_features, feature)

    if (length(current_features) == 0) {
      exit_loop <- TRUE
      break  # Exit the inner loop when only one feature is left
    }

    # Train the model with the current set of features
    formula_str <- paste("factor(ordinal_quality) ~ ", paste(
        current_features, collapse = "+"), sep = "")
    model <- polr(as.formula(formula_str), data = train_data, Hess
        = TRUE)

    # Predict using the trained model on the test set
    # (Ensure test_data is also a data frame)
    predicted_labels <- predict(model, newdata = as.data.frame(
        test_data))

    # Convert predicted and true labels to integers (if not
        already)
    predicted_labels <- as.integer(predicted_labels)
    true_labels <- as.integer(test_data$ordinal_quality)

    # Compute accuracy
    accuracy <- sum(predicted_labels == true_labels) / length(true
        _labels)

    # Update the current accuracy, worst feature, and number of
        variables if needed
    if (accuracy > current_accuracy) {
      current_accuracy <- accuracy
      worst_feature <- feature
      current_num_variables <- length(current_features)
    }
  }
```

```
  if (exit_loop) {
    break  # Exit the outer loop when only one feature is left
  }


  # Remove the worst feature from the selected features
  selected_features <- setdiff(selected_features, worst_feature)

  # Update the best feature set, accuracy, and number of variables
      if needed
  if (current_accuracy > best_accuracy ||
      (current_accuracy == best_accuracy && current_num_variables
         < best_num_variables)) {
    best_accuracy <- current_accuracy
    best_feature_set <- selected_features
    best_num_variables <- current_num_variables
  }

  #cat("Selected features:", selected_features, "\n")
  #cat("Current accuracy:", current_accuracy, "\n")
  #cat("Current number of variables:", current_num_variables, "\n\
     n")
}
cat("Best feature set:", best_feature_set, "\n")
cat("Best accuracy:", best_accuracy, "\n")
cat("Best number of variables:", best_num_variables, "\n")

# Multiple Regression
initial_model <- lm(ordinal_quality ~ residual.sugar + chlorides +
     total.sulfur.dioxide +
                      density+ pH + sulphates + alcohol + residual
                         .sugar*chlorides +
                      residual.sugar*total.sulfur.dioxide +
                      residual.sugar*density + residual.sugar*pH +
                          residual.sugar*sulphates +
                      residual.sugar*alcohol + chlorides*total.
                         sulfur.dioxide +
                      chlorides*density + chlorides*pH + chlorides
                         *sulphates + chlorides*alcohol +
                      total.sulfur.dioxide*density + total.sulfur.
                         dioxide*pH +
                      total.sulfur.dioxide*sulphates + total.
                         sulfur.dioxide*alcohol +
                      density*pH + density*sulphates + density*
                         alcohol + pH*sulphates +
                      pH*alcohol + sulphates*alcohol, data = train
                         _data)

# Perform backward elimination using stepwise regression
final_model <- step(initial_model, direction = "backward")

# Display the final model
summary(final_model)

X_test <- as.data.frame(test_data[, !grepl("ordinal_quality",
    names(test_data))])
predicted_labels <- predict(final_model, newdata = X_test)

# Convert predicted and true labels to integers (if not already)
```

```
predicted_labels <- as.integer(predicted_labels)
true_labels <- as.integer(test_data$ordinal_quality)

# Compute accuracy
accuracy <- sum(predicted_labels == true_labels) / length(true_
    labels)
print(accuracy)
```

6.2 Appendix B: R Code for Bayesian Analysis

```r
```{r, include=FALSE}
library("latex2exp")
library("ggplot2")
library("MASS")
library("coda")
library("dplyr")
library("tidyr")
library("caret")
library("reshape2")
library("DescTools")
```

```{r, warning=FALSE, message=FALSE}
rm(list=ls())
#Preprocessing to prepare the data for Bayesian regression on
    Quality
#Read data
data <- read.csv("winequality-red.csv", sep=';')
quality_column <- data$quality

# Standardize all columns except the "quality" column
standardized_data <-
  as.data.frame(scale(data[, -which(names(data) == "quality")]))
standardized_data <- cbind(quality = quality_column, standardized_
    data)

set.seed(123)  # Set seed for reproducibility

# Interaction Terms
interaction_terms <- c("residual.sugar*chlorides",
                       "residual.sugar*total.sulfur.dioxide",
                       "residual.sugar*density", "residual.sugar*
                          pH",
                       "residual.sugar*sulphates",
                       "residual.sugar*alcohol",
                       "chlorides*total.sulfur.dioxide",
                       "chlorides*density", "chlorides*pH",
                       "chlorides*sulphates", "chlorides*alcohol",
                       "total.sulfur.dioxide*density",
                       "total.sulfur.dioxide*pH",
                       "total.sulfur.dioxide*sulphates",
                       "total.sulfur.dioxide*alcohol",
                       "density*pH", "density*sulphates",
                       "density*alcohol", "pH*sulphates",
                       "pH*alcohol", "sulphates*alcohol")

# Iterate through the interaction terms and create new
# columns for each interaction term
for (term in interaction_terms) {
  terms <- strsplit(term, "*")[[1]]
  col_name <- paste(terms, collapse = "_times_")
  # Create the new column
  standardized_data <- mutate(standardized_data, !!col_name
                              := !!sym(terms[1]) * !!sym(terms[2])
                                 )
}
```
```

```
# Non-correlated features
features_reg <- c("residual.sugar", "chlorides",
                  "total.sulfur.dioxide", "density", "pH", "
                     sulphates",
                  "alcohol",
                  "residual.sugar_times_chlorides",
                  "residual.sugar_times_total.sulfur.dioxide",
                  "residual.sugar_times_density",
                  "residual.sugar_times_pH",
                  "residual.sugar_times_sulphates",
                  "residual.sugar_times_alcohol",
                  "chlorides_times_total.sulfur.dioxide",
                  "chlorides_times_density",
                  "chlorides_times_pH",
                  "chlorides_times_sulphates",
                  "chlorides_times_alcohol",
                  "total.sulfur.dioxide_times_density",
                  "total.sulfur.dioxide_times_pH",
                  "total.sulfur.dioxide_times_sulphates",
                  "total.sulfur.dioxide_times_alcohol",
                  "density_times_pH",
                  "density_times_sulphates",
                  "density_times_alcohol",
                  "pH_times_sulphates",
                  "pH_times_alcohol",
                  "sulphates_times_alcohol",
                  "quality")

standardized_data <- standardized_data[, (names(standardized_data)
                                          %in% features_reg)]
standardized_data_q <- standardized_data



```

```{r, warning=FALSE, message=FALSE}
#Preprocessing to prepare the data for Bayesian regression on
    Ordinal_Quality

standardized_data_oq <- standardized_data

# Add ordinal_quality - don't do when doing quality
standardized_data_oq$ordinal_quality <- cut(standardized_data_oq$
    quality,
                    breaks = c(-Inf, 4.5, 5.5, Inf),
                    labels = c(1, 2, 3),
                    include.lowest = TRUE)

# Convert the new_column to numeric type
standardized_data_oq$ordinal_quality <-
  as.numeric(standardized_data_oq$ordinal_quality)

# DROP QUALITY COLUMN - when doing ordinal quality
standardized_data_oq <-
  standardized_data_oq[, !(names(standardized_data_oq) %in% c("
      quality"))]
```

```
```

```{r, warning=FALSE, message=FALSE}
#MCMC Model Selection for Quality, using Bayesian Linear
    Regression

#Function to carry out Bayesian linear regression (used in our
    Gibbs sampler)
lm.gprior<-function(y,X,g=dim(X)[1],nu0=1,s20=
                        try(summary(lm(y~-1+X))$sigma^2,silent=TRUE)
                            ,S=1000)
{

  n<-dim(X)[1] ; p<-dim(X)[2]
  Hg<- (g/(g+1)) * X%*%solve(t(X)%*%X)%*%t(X)
  SSRg<- t(y)%*%( diag(1,nrow=n) - Hg ) %*%y

  s2<-1/rgamma(S, (nu0+n)/2, (nu0*s20+SSRg)/2 )

  Vb<- g*solve(t(X)%*%X)/(g+1)
  Eb<- Vb%*%t(X)%*%y

  E<-matrix(rnorm(S*p,0,sqrt(s2)),S,p)
  beta<-t(  t(E%*%chol(Vb)) +c(Eb))

  list(beta=beta,s2=s2)
}

#Function to compute the marginal probability
lpy.X<-function(y,X,
   g=length(y),nu0=1,s20=try(summary(lm(y~-1+X))$sigma^2,silent=
       TRUE))
{
  n<-dim(X)[1] ; p<-dim(X)[2]
  if(p==0) { s20<-mean(y^2) }
  H0<-0 ; if(p>0) { H0<- (g/(g+1)) * X%*%solve(t(X)%*%X)%*%t(X) }
  SS0<- t(y)%*%( diag(1,nrow=n) - H0 ) %*%y

  -.5*n*log(2*pi) +lgamma(.5*(nu0+n)) - lgamma(.5*nu0)  - .5*p*log
      (1+g) +
   .5*nu0*log(.5*nu0*s20) -.5*(nu0+n)*log(.5*(nu0*s20+SS0))
}

#Setup for Gibbs Sampler
predictors <- names(standardized_data_q)[names(standardized_data_q
    ) != "quality"]
X <- model.matrix(quality ~ ., data =
                      standardized_data_q[, c("quality", predictors)
                          ])
y <- standardized_data_q$quality
n<-dim(X)[1]
p<-dim(X)[2]
S<-10000
BETA<-Z<-matrix(NA,S,p)
z<-rep(1,dim(X)[2] )
lpy.c<-lpy.X(y,X[,z==1,drop=FALSE])
```

```
#Gibbs Sampler
for(s in 1:S)
{
  for(j in sample(1:p))
  {
    zp<-z ; zp[j]<-1-zp[j]
    lpy.p<-lpy.X(y,X[,zp==1,drop=FALSE])
    r<- (lpy.p - lpy.c)*(-1)^(zp[j]==0)
    z[j]<-rbinom(1,1,1/(1+exp(-r)))
    if(z[j]==zp[j]) {lpy.c<-lpy.p}
  }

  beta<-z
  if(sum(z)>0){beta[z==1]<-lm.gprior(y,X[,z==1,drop=FALSE],S=1)$
      beta }
  Z[s,]<-z
  BETA[s,]<-beta
}

colnames(BETA) <- colnames(X)

```

```{r, warning=FALSE, message=FALSE}

#Save our Beta and Z Gibbs Sample results
BETA_q <- BETA
Z_q <- Z

write.csv(BETA_q, file = "BETA_Quality.csv", row.names = FALSE)
write.csv(Z_q, file = "Z_Quality.csv", row.names = FALSE)



```

```{r, warning=FALSE, message=FALSE}

z <- read.csv("Z_Quality.csv", sep=',')
#Calculate posterior probabilities for each of the 29 regressors
Zcp<- apply(z, 2, cumsum) / seq_len(nrow(z))
last_row <- Zcp[nrow(Zcp),, drop=FALSE]
#set each z = 1 if p(z|y,x) > 0.5 and z = 0 if p(z|y,x) <= 0.5
last_row[last_row > 0.5] <- 1
last_row[last_row <= 0.5] <- 0
colnames(last_row) <- colnames(X)
#get the list of variables with z = 1
variables_to_use_q_model <- colnames(last_row)[which(last_row ==
    1)]
variables_to_use_q_model <- variables_to_use_q_model[variables_to_
    use_q_model
                                                     != "(
                                                         Intercept
                                                         )"]
#select the columns to be used for the quality model
q_model <- standardized_data_q[, variables_to_use_q_model, drop =
    FALSE]
#add quality back into the model
```

```
q_model <- cbind(quality = quality_column, q_model)
```

```
```{r, warning=FALSE, message=FALSE}
#MCMC Model Selection For Ordinal Quality, using Bayesian Linear
    Regression

#Get all predictors, X and Y
predictors <- names(standardized_data_oq)[names(standardized_data_
    oq)
                                          != "ordinal_quality"]
X <- model.matrix(ordinal_quality ~ ., data =
                    standardized_data_oq[, c("ordinal_quality",
                       predictors)])
y <- standardized_data_oq$ordinal_quality

#Set up for Gibbs Sampler
n<-dim(X)[1]
p<-dim(X)[2]
S<-10000
BETA<-Z<-matrix(NA,S,p)
z<-rep(1,dim(X)[2] )
lpy.c<-lpy.X(y,X[,z==1,drop=FALSE])

for(s in 1:S)
{
  for(j in sample(1:p))
  {
    zp<-z ; zp[j]<-1-zp[j]
    lpy.p<-lpy.X(y,X[,zp==1,drop=FALSE])
    r<- (lpy.p - lpy.c)*(-1)^(zp[j]==0)
    z[j]<-rbinom(1,1,1/(1+exp(-r)))
    if(z[j]==zp[j]) {lpy.c<-lpy.p}
  }

  beta<-z
  if(sum(z)>0){beta[z==1]<-lm.gprior(y,X[,z==1,drop=FALSE],S=1)$
      beta }
  Z[s,]<-z
  BETA[s,]<-beta
}
colnames(BETA) <- colnames(X)
```

```
```{r, warning=FALSE, message=FALSE}

#Save our Beta and Z Gibbs Sample results
BETA_oq <- BETA
Z_oq <- Z

write.csv(BETA_oq, file = "BETA_Ordinal_Quality.csv", row.names =
    FALSE)
write.csv(Z_oq, file = "Z_Ordinal_Quality.csv", row.names = FALSE)
```

```
```

```{r, warning=FALSE, message=FALSE}

z <- read.csv("Z_Ordinal_Quality.csv", sep=',')
#Calculate posterior probabilities for each of the 29 regressors
Zcp<- apply(z, 2, cumsum) / seq_len(nrow(z))
last_row <- Zcp[nrow(Zcp),, drop=FALSE]
#set each z = 1 if p(z|y,x) > 0.5 and z = 0 if p(z|y,x) <= 0.5
last_row[last_row > 0.5] <- 1
last_row[last_row <= 0.5] <- 0
colnames(last_row) <- colnames(X)
#get the list of variables with z = 1
variables_to_use_oq_model <- colnames(last_row)[which(last_row ==
    1)]
variables_to_use_oq_model <-
  variables_to_use_oq_model[variables_to_use_oq_model != "(
      Intercept)"]
#select the columns to be used for the quality model
oq_model <- standardized_data_oq[, variables_to_use_oq_model, drop
    = FALSE]
#add ordinal quality back into the model
ordinal_quality_column <- standardized_data_oq$ordinal_quality
oq_model <- cbind(ordinal_quality = ordinal_quality_column, oq_
    model)

```

```{r, warning=FALSE, message=FALSE}

### Linear regression with quality as the target variable
predictors <- names(q_model)[names(q_model) != "quality"]

# Train/Test Split
set.seed(123)  # Set seed for reproducibility
split_index <- createDataPartition(q_model$quality, p = 0.8, list
    = FALSE)
train_data <- standardized_data_q[split_index, ]
test_data <- standardized_data_q[-split_index, ]

yquality<-match(train_data$quality,sort(unique(train_data$quality)
    ))

y_train<-train_data$quality
X_train<- model.matrix(quality ~ .,
                       data = train_data[, c("quality", predictors
                           )])
keep<- (1:length(y_train))[ !is.na( apply( cbind(X_train,y_train)
    ,1,mean) ) ]
X_train<-X_train[keep,] ; y_train<-y_train[keep]

n<-dim(X_train)[1]
p<-dim(X_train)[2]

X_test<- model.matrix(quality ~ ., data = test_data[, c("quality",
     predictors)])
y_test <-test_data$quality
keep<- (1:length(y_test))[ !is.na( apply( cbind(X_test,y_test),1,
    mean) ) ]
X_test<-X_test[keep,] ; y_test<-y_test[keep]
```

```
#set default priors for g and nu0
g = n
nu0 = 1

#Compute sigma2_hat_ols to set as our sigma20
# Fit a linear regression model
lm_model <- lm(quality ~ ., data = q_model)
residuals <- residuals(lm_model)
n_ols <- length(residuals)
p_ols <- length(coef(lm_model))  # Number of coefficients,
    including intercept
s2_hat_ols <- sum(residuals^2) / (n_ols - p_ols)
s20 <- s2_hat_ols

#Priors: g = n, nu0 = 1, sigma20 = sigma2_hat_ols
S <- 10000

Hg <- (g / (g + 1)) * X_train %*% solve(t(X_train) %*% X_train) %*
    % t(X_train)
SSRg <- t(y_train) %*% (diag(1, nrow = n) - Hg) %*% y_train

s2 <- 1 / rgamma(S, (nu0 + n) / 2, (nu0 * s20 + SSRg) / 2)

Vb <- g * solve(t(X_train) %*% X_train) / (g + 1)
Eb <- Vb %*% t(X_train) %*% y_train

E <- matrix(rnorm(S * p, 0, sqrt(s2)), S, p)
beta <- t(t(E %*% chol(Vb)) + c(Eb))

beta_bayes <- as.matrix(colMeans(beta))

y_bayes <- X_test %*% beta_bayes
y_bayes <- y_bayes

y_test <- y_test

bayes_df = data.frame(
  observed = y_test,
  predicted = y_bayes
)
ggplot(bayes_df, aes(x = observed, y = predicted)) +
  geom_point()

predicted_labels <- as.integer(y_bayes)

# Convert predicted and true labels to integers (if not already)
true_labels <- as.integer(y_test)


# Compute accuracy
accuracy <- sum(predicted_labels == true_labels) / length(true_
    labels)
print(accuracy)


```

```{r, warning=FALSE, message=FALSE}


```

```
### Ordered probit regression with quality as the target variable

#Shift quality down by two for the ordered probit gression
q_model$quality <- q_model$quality - 2

# Train/Test Split
set.seed(123)  # Set seed for reproducibility
split_index <- createDataPartition(q_model$quality, p = 0.8, list
    = FALSE)
train_data <- q_model[split_index, ]
test_data <- q_model[-split_index, ]

yquality<-match(train_data$quality,sort(unique(train_data$quality)
    ))

# Regress on all
tmp<-lm(yquality ~ ., data=train_data)

#####
X<-as.matrix(train_data[, !grepl("quality", names(train_data))]) #
     drop Y column
y<-train_data$quality
keep<- (1:length(y))[ !is.na( apply( cbind(X,y),1,mean) ) ]
X<-X[keep,] ; y<-y[keep]
ranks<-match(y,sort(unique(y))) ;
uranks<-sort(unique(ranks))
#ranks<-10 ;
#uranks<-10 ;
n<-dim(X)[1] ; p<-dim(X)[2]
iXX<-solve(t(X)%*%X)  ; V<-iXX*(n/(n+1)) ; cholV<-chol(V)

#### Ordinal probit regression

## setup
set.seed(1)
beta<-rep(0,p)
z<-qnorm(rank(y,ties.method="random")/(n+1))
g<-rep(NA,length(uranks)-1)
K<-length(uranks)
BETA<-matrix(NA,1000,p) ; Z<-matrix(NA,1000,n) ; ac<-0
mu<-rep(0,K-1) ; sigma<-rep(1000,K-1)

## MCMC
S<-25000
for(s in 1:S)
{

  #update g
  for(k in 1:(K-1))
  {
  a<-max(z[y==k])
  b<-min(z[y==k+1])
  u<-runif(1, pnorm( (a-mu[k])/sigma[k] ),
              pnorm( (b-mu[k])/sigma[k] ) )
  g[k]<- mu[k] + sigma[k]*qnorm(u)
  }

  #update beta
  E<- V%*%( t(X)%*%z )
```

```
  beta<- cholV%*%rnorm(p) + E

  #update z
  ez<-X%*%beta
  a<-c(-Inf,g)[ match( y-1, 0:K) ]
  b<-c(g,Inf)[y]
  u<-runif(n, pnorm(a-ez),pnorm(b-ez) )
  z<- ez + qnorm(u)


  #help mixing
  c<-rnorm(1,0,n^(-1/3))
  zp<-z+c ; gp<-g+c
  lhr<-  sum(dnorm(zp,ez,1,log=T) - dnorm(z,ez,1,log=T) ) +
         sum(dnorm(gp,mu,sigma,log=T) - dnorm(g,mu,sigma,log=T) )
  if(log(runif(1))<lhr) { z<-zp ; g<-gp ; ac<-ac+1 }

  if(s%%(S/1000)==0)
  {
    #cat(s/S,ac/s,"\n")
    BETA[s/(S/1000),]<-  beta
    Z[s/(S/1000),]<- z
  }
}

# Assuming BETA and Z matrices are available from the MCMC
    sampling

# Extract mean values from MCMC samples
mean_beta <- colMeans(BETA, na.rm = TRUE)

# Extract relevant columns from the test set
X_test <- as.matrix(test_data[, !grepl("quality", names(test_data)
    )])

# Make predictions using the mean of MCMC samples
final_g <- X_test %*% mean_beta

print(g)
# Convert predictions to ordinal labels
predicted_labels <- cut(final_g,
                        breaks = c(-Inf, g, Inf),
                        labels = c(3, 4, 5, 6, 7, 8),
                        include.lowest = TRUE)

# Evaluate the predictions
#(assuming you have true ordinal_quality values for the test set)
confusion_matrix <- table(predicted_labels, test_data$quality+2)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

# Print the confusion matrix and accuracy
print("Confusion␣Matrix:")
print(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 4)))
```

```r
```{r, warning=FALSE, message=FALSE}
### Linear regression with ordinal_quality as the target variable


predictors <- names(oq_model)[names(oq_model) != "ordinal_quality"
    ]

# Train/Test Split
set.seed(123)  # Set seed for reproducibility
split_index <-
  createDataPartition(oq_model$ordinal_quality, p = 0.8, list =
      FALSE)
train_data <- oq_model[split_index, ]
test_data <- oq_model[-split_index, ]

yquality<-match(train_data$quality,sort(unique(train_data$ordinal_
    quality)))

y_train<-train_data$ordinal_quality
X_train<- model.matrix(ordinal_quality ~ .,
                       data = train_data[, c("ordinal_quality",
                          predictors)])
keep<- (1:length(y_train))[ !is.na( apply( cbind(X_train,y_train)
    ,1,mean) ) ]
X_train<-X_train[keep,] ; y_train<-y_train[keep]

n<-dim(X_train)[1]
p<-dim(X_train)[2]

X_test<- model.matrix(ordinal_quality ~ .,
                      data = test_data[, c("ordinal_quality",
                         predictors)])
y_test <-test_data$ordinal_quality
keep<- (1:length(y_test))[ !is.na( apply( cbind(X_test,y_test),1,
    mean) ) ]
X_test<-X_test[keep,] ; y_test<-y_test[keep]

#set default priors for g and nu0
g = n
nu0 = 1

#Compute sigma2_hat_ols to set as our sigma20
# Fit a linear regression model
lm_model <- lm(ordinal_quality ~ ., data = oq_model)
residuals <- residuals(lm_model)
n_ols <- length(residuals)
p_ols <- length(coef(lm_model))  # Number of coefficients,
    including intercept
s2_hat_ols <- sum(residuals^2) / (n_ols - p_ols)
s20 <- s2_hat_ols

#Priors: g = n, nu0 = 1, sigma20 = sigma2_hat_ols


S <- 25000

Hg <- (g / (g + 1)) * X_train %*% solve(t(X_train) %*% X_train) %*
    % t(X_train)
SSRg <- t(y_train) %*% (diag(1, nrow = n) - Hg) %*% y_train
```
```

```
s2 <- 1 / rgamma(S, (nu0 + n) / 2, (nu0 * s20 + SSRg) / 2)

Vb <- g * solve(t(X_train) %*% X_train) / (g + 1)
Eb <- Vb %*% t(X_train) %*% y_train

E <- matrix(rnorm(S * p, 0, sqrt(s2)), S, p)
beta <- t(t(E %*% chol(Vb)) + c(Eb))

beta_bayes <- as.matrix(colMeans(beta))

y_bayes <- X_test %*% beta_bayes
y_bayes <- y_bayes

y_test <- y_test

bayes_df = data.frame(
  observed = y_test,
  predicted = y_bayes
)
ggplot(bayes_df, aes(x = observed, y = predicted)) +
  geom_point()

predicted_labels <- as.integer(y_bayes)

# Convert predicted and true labels to integers (if not already)
true_labels <- as.integer(y_test)

# Compute accuracy
accuracy <- sum(predicted_labels == true_labels) / length(true_
    labels)
print(accuracy)

```



```{r, warning=FALSE, message=FALSE}
### ordered probit regression with ordinal_quality as the target
    variable


# Train/Test Split
set.seed(123)  # Set seed for reproducibility
split_index <-
  createDataPartition(oq_model$ordinal_quality, p = 0.8, list =
      FALSE)
train_data <- oq_model[split_index, ]
test_data <- oq_model[-split_index, ]

yordquality<-
  match(train_data$ordinal_quality,sort(unique(train_data$ordinal_
      quality)))

# Regress on all
tmp<-lm(yordquality ~ ., data=train_data)

#####
X<-as.matrix(train_data[, !grepl("ordinal_quality",
```

```
                                        names(train_data))]) # drop Y
                                            column
y<-train_data$ordinal_quality
keep<- (1:length(y))[ !is.na( apply( cbind(X,y),1,mean) ) ]
X<-X[keep,] ; y<-y[keep]
ranks<-match(y,sort(unique(y))) ; uranks<-sort(unique(ranks))
n<-dim(X)[1] ; p<-dim(X)[2]
iXX<-solve(t(X)%*%X) ; V<-iXX*(n/(n+1)) ; cholV<-chol(V)

#### Ordinal probit regression

## setup
set.seed(1)
beta<-rep(0,p)
z<-qnorm(rank(y,ties.method="random")/(n+1))
g<-rep(NA,length(uranks)-1)
K<-length(uranks)
BETA<-matrix(NA,1000,p) ; Z<-matrix(NA,1000,n) ; ac<-0
mu<-rep(0,K-1) ; sigma<-rep(1000,K-1)

## MCMC
S<-25000
for(s in 1:S)
{

  #update g
  for(k in 1:(K-1))
  {
  a<-max(z[y==k])
  b<-min(z[y==k+1])
  u<-runif(1, pnorm( (a-mu[k])/sigma[k] ),
              pnorm( (b-mu[k])/sigma[k] ) )
  g[k]<- mu[k] + sigma[k]*qnorm(u)
  }

  #update beta
  E<- V%*%( t(X)%*%z )
  beta<- cholV%*%rnorm(p) + E

  #update z
  ez<-X%*%beta
  a<-c(-Inf,g)[ match( y-1, 0:K) ]
  b<-c(g,Inf)[y]
  u<-runif(n, pnorm(a-ez),pnorm(b-ez) )
  z<- ez + qnorm(u)


  #help mixing
  c<-rnorm(1,0,n^(-1/3))
  zp<-z+c ; gp<-g+c
  lhr<- sum(dnorm(zp,ez,1,log=T) - dnorm(z,ez,1,log=T) ) +
        sum(dnorm(gp,mu,sigma,log=T) - dnorm(g,mu,sigma,log=T) )
  if(log(runif(1))<lhr) { z<-zp ; g<-gp ; ac<-ac+1 }

  if(s%%(S/1000)==0)
  {
    #cat(s/S,ac/s,"\n")
    BETA[s/(S/1000),]<- beta
    Z[s/(S/1000),]<- z
```

```
  }
}

# Assuming BETA and Z matrices are available from the MCMC
    sampling

# Extract mean values from MCMC samples
mean_beta <- colMeans(BETA, na.rm = TRUE)

# Extract relevant columns from the test set
X_test <- as.matrix(test_data[, !grepl("ordinal_quality", names(
    test_data))])

# Make predictions using the mean of MCMC samples
final_g <- X_test %*% mean_beta
predicted_labels <- cut(final_g,
                        breaks = c(-Inf, g, Inf),
                        labels = c(1, 2, 3),
                        include.lowest = TRUE)

# Evaluate the predictions
#(assuming you have true ordinal_quality values for the test set)
confusion_matrix <- table(predicted_labels, test_data$ordinal_
    quality)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

# Print the confusion matrix and accuracy
print("Confusion Matrix:")
print(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 4)))
```