# Dataset Documentation

Link to Dataset Documentation:

<https://archive.ics.uci.edu/dataset/94/spambase>

<https://archive.ics.uci.edu/dataset/186/wine+quality>

# Linear Regression Analysis

## Business Understanding / Objective

## This project attempts to develop a predictive model to classify Portuguese "Vinho Verde" red wine quality based on some physicochemical properties including acidity, alcohol content, and sugar levels. By doing this we can predict a wine's sensory quality score based on these chemical properties, allowing wine producers and distributors to estimate quality further upstream in the production chain.

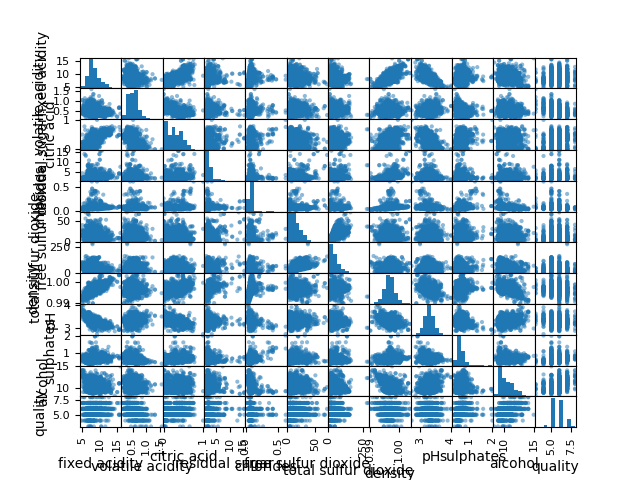
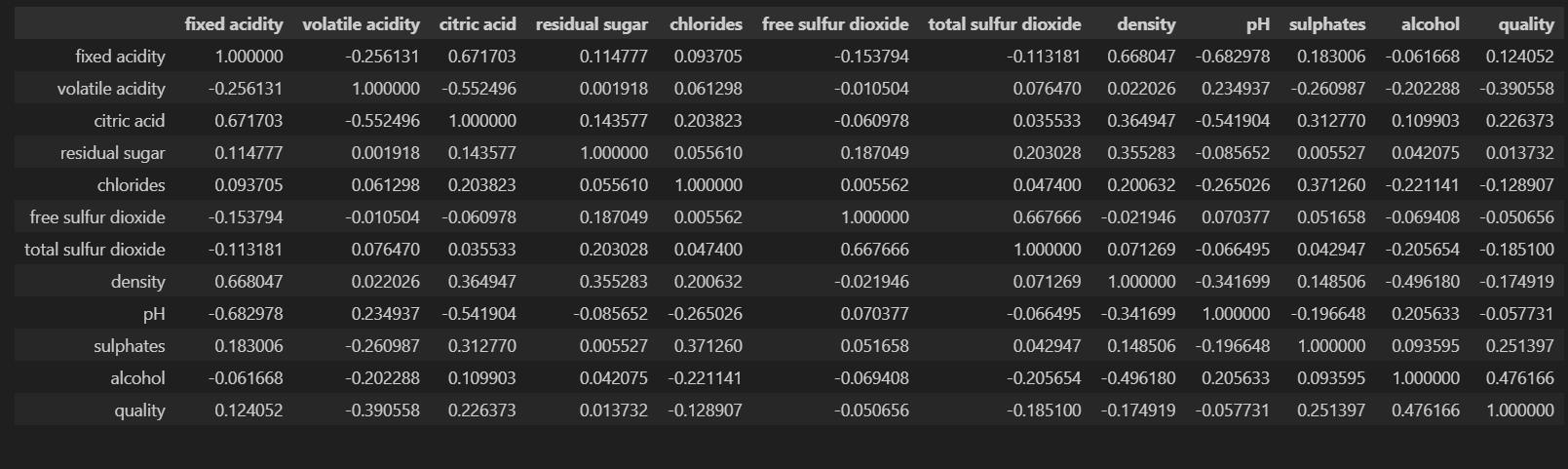
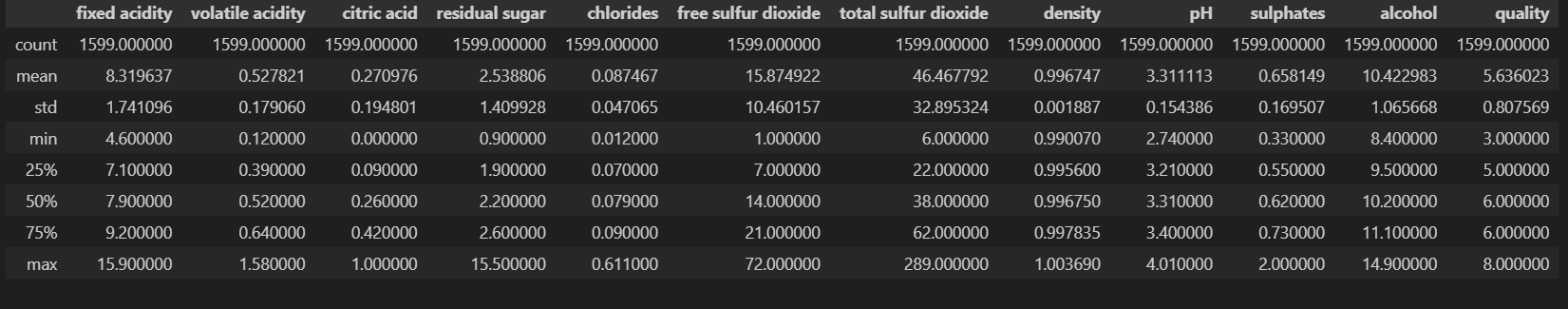
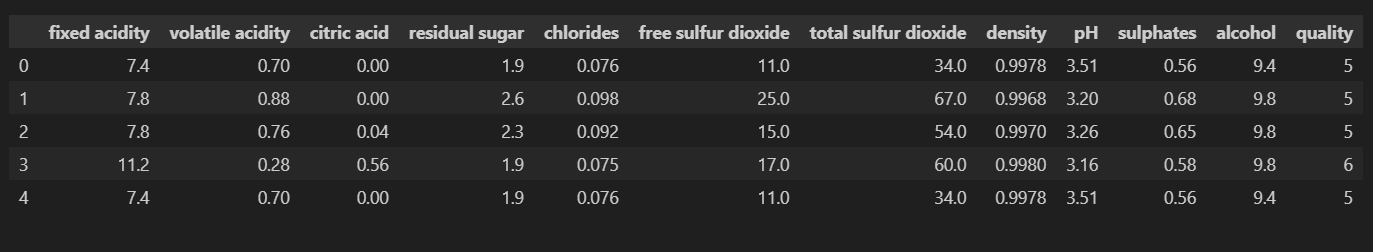
## This dataset permits us to abstract the problem as a regression one, since in our case the output (to be predicted) is a continuous quality score. Such a model may provide producers with better decision making on what wines may perform better in a score (or possibly identifying areas in the process where quality may be improved).

## Data Understanding / Exploration

Dataset Characteristics:  
- Number of features: 11  
- Target variable: Quality (0 to 10)  
- Feature descriptions: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol

Code snippet for loading and exploring data:

df = pd.read\_csv('data/winequality-red.csv', sep=’;’)  
df.head()  
df.describe()  
df.corr()  
scatter\_matrix(df)  
plt.show()



Based on the scatter matrix plotted, it is unlikely to get a good model because very little correlation can be seen between features and the target variable.

## Modeling

The Linear Regression model was configured with a random\_state for training/testing data split.

Code snippet for model fitting:

X = df.drop(columns = 'quality')

y = df.quality

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state=1)  
  
model = LinearRegression()  
model.fit(X\_train, y\_train)

## Evaluation

Evaluation metrics include R² score and root mean squared error (RMSE):  
  
print('coefficient of determination R^2:', model.score(X\_train, y\_train))  
yhat = model.predict(X\_test)  
print("RMSE:", mean\_squared\_error(y\_test, yhat, squared=False))

coefficient of determination R^2: 0.36558499214790485

RMSE: 0.6189280908000773

## Conclusion

The Linear Regression model provided an RMSE of 0.62 and an R² of 0.37. This analysis indicates that the model is bad for predicting the target variable in this business context since it performs poorly with low- and high-quality wines.

# Decision Tree Analysis

## Business Understanding / Objective

## This project aims to build a predictive model to classify incoming emails as either "spam" or "not spam." This model could also be useful for people or businesses who want to filter out spam and save space in their inboxes and unwanted links.

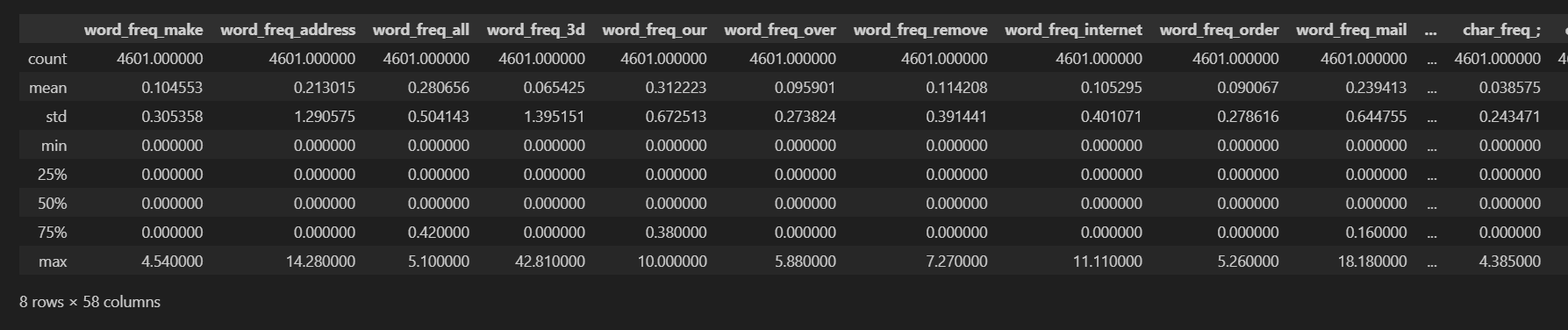
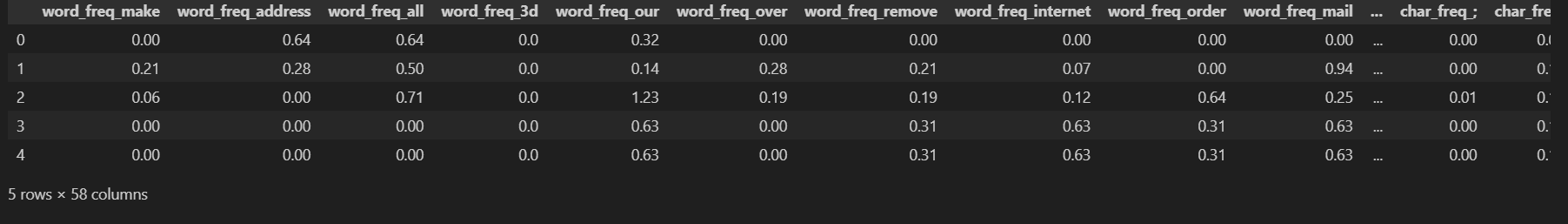
## This dataset utilizes the patterns that can be found within a spam or non-spam email, to determine the category of an email. For instance, non-spam emails may contain words (like "george" and area code "650") that are absent in spam emails, and spam emails contain regularities of non-personal computer usage (like particular phrase patterns typical of advertising or scams). Then we will use this knowledge of how language works to build a custom spam filter which will filter the mails very well and enhances user efficiency and email system.

## Data Understanding / Exploration

Dataset Characteristics:  
- Number of features: 57  
- Target variable: Class (Spam or not spam)  
- Feature descriptions: Frequency of characters appearing in an email

Code snippet for loading and exploring data:

col\_names = [all features]  
df = pd.read\_csv('data/spambase.csv', header=None, names=col\_names)  
df.head()  
df.describe()



Based on frequency charts that were plotted, words like 'you' and 'will' seems to appear a lot more.

## Modeling

The model configuration includes setting a random\_state for reproducibility and using train\_test\_split for training and test data separation.

Code snippet for model fitting:  
X = df.drop(columns = 'Class')

y = df["Class"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
model = DecisionTreeClassifier()

model.fit(X\_train, y\_train)

## Evaluation

Evaluation metrics include accuracy and confusion matrix:  
  
yhat = model.predict(X\_test)  
accuracy = accuracy\_score(y\_test, yhat)

conf\_matrix = confusion\_matrix(y\_test, yhat)

print('Accuracy:', accuracy)

print('Confusion Matrix:', conf\_matrix)

Accuracy: 0.9077090119435396

Confusion Matrix: [[494 37]

[ 48 342]]

Cross validation was needed for this decision tree since we need to find the best depth for the model.

Finding the best depth for the decision tree using cross validation:

for d in range(2, 20):

model = DecisionTreeClassifier(max\_depth=d)

scores = cross\_val\_score(model, X\_train, y\_train, cv=5)

print(f"Depth: {d}, Mean Cross-Validation Score: {scores.mean():.4f}")

Depth: 2, Mean Cross-Validation Score: 0.8677

Depth: 3, Mean Cross-Validation Score: 0.8785

Depth: 4, Mean Cross-Validation Score: 0.9016

Depth: 5, Mean Cross-Validation Score: 0.9060

Depth: 6, Mean Cross-Validation Score: 0.9109

Depth: 7, Mean Cross-Validation Score: 0.9144

Depth: 8, Mean Cross-Validation Score: 0.9163

Depth: 9, Mean Cross-Validation Score: 0.9204

Depth: 10, Mean Cross-Validation Score: 0.9136

Depth: 11, Mean Cross-Validation Score: 0.9174

Depth: 12, Mean Cross-Validation Score: 0.9171

Depth: 13, Mean Cross-Validation Score: 0.9185

Depth: 14, Mean Cross-Validation Score: 0.9160

Depth: 15, Mean Cross-Validation Score: 0.9117

Depth: 16, Mean Cross-Validation Score: 0.9144

Depth: 17, Mean Cross-Validation Score: 0.9166

Depth: 18, Mean Cross-Validation Score: 0.9163

Depth: 19, Mean Cross-Validation Score: 0.9136

By using cross validation, the decision tree with a max depth of 9 appears to have the highest accuracy.

Retrain the model with the best decision tree (depth = 9) and creating a plot for it:

model = DecisionTreeClassifier(max\_depth = 9)

model.fit(X\_train, y\_train)

# Predicting the target on the test set

yhat = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, yhat)

conf\_matrix = confusion\_matrix(y\_test, yhat)

print('Accuracy:', accuracy)

print('Confusion Matrix:', conf\_matrix)

Accuracy: 0.9218241042345277

Confusion Matrix: [[508 23]

[ 49 341]]

target\_names = ["spam", "not spam"]

dot\_data = StringIO()

export\_graphviz(model, out\_file=dot\_data,

filled=True, rounded=True,

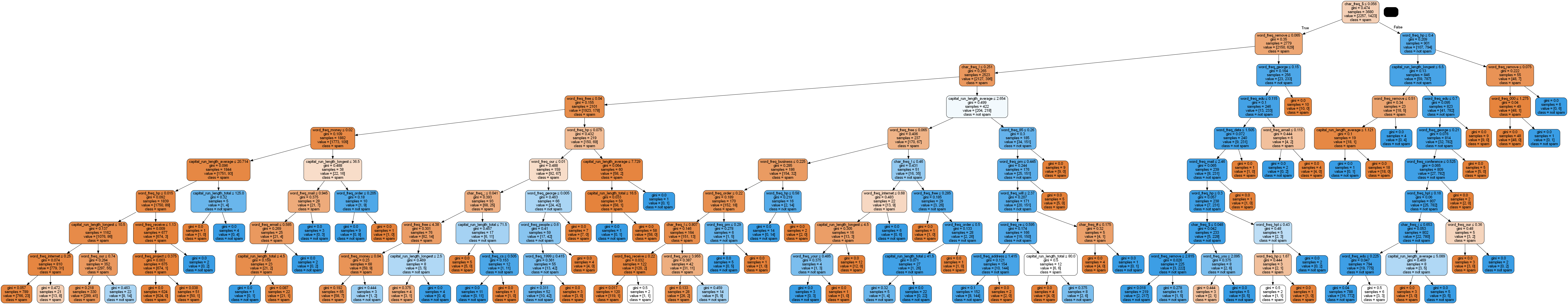
special\_characters=True, feature\_names = features,

class\_names = target\_names)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('plots/spambase\_bestDepth.png')

Image(graph.create\_png())



## Conclusion

The Decision Tree model performed with an accuracy of 92%, indicating it is effectively distinguishing between spam and non-spam emails, making it a strong model for this classification task.