PROJECT 4 WINE ANALYSIS

11.25.2024

Predicting Wine Quality

Collaborators:

Chuck Bui

Jack Jeffries

Beau Massie

Christopher Turner

Problem Statement

VinoVista, a renowned winery, is committed to producing consistently exceptional wines. However, they have observed variability in the quality of their vinho verde batches. To address this challenge, VinoVista seeks a predictive model that can accurately assess the quality of wine batches based on their chemical properties before bottling.

Project Goal:

The primary objective of this project is to develop a robust machine learning model capable of predicting wine quality on a scale of 0-10. This model will utilize a dataset containing various chemical properties of wine, such as acidity, pH, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, sulfates, and alcohol content.

Resources:

Wine Quality Dataset from UC Irvine Machine Learning Repository with 4899 samples

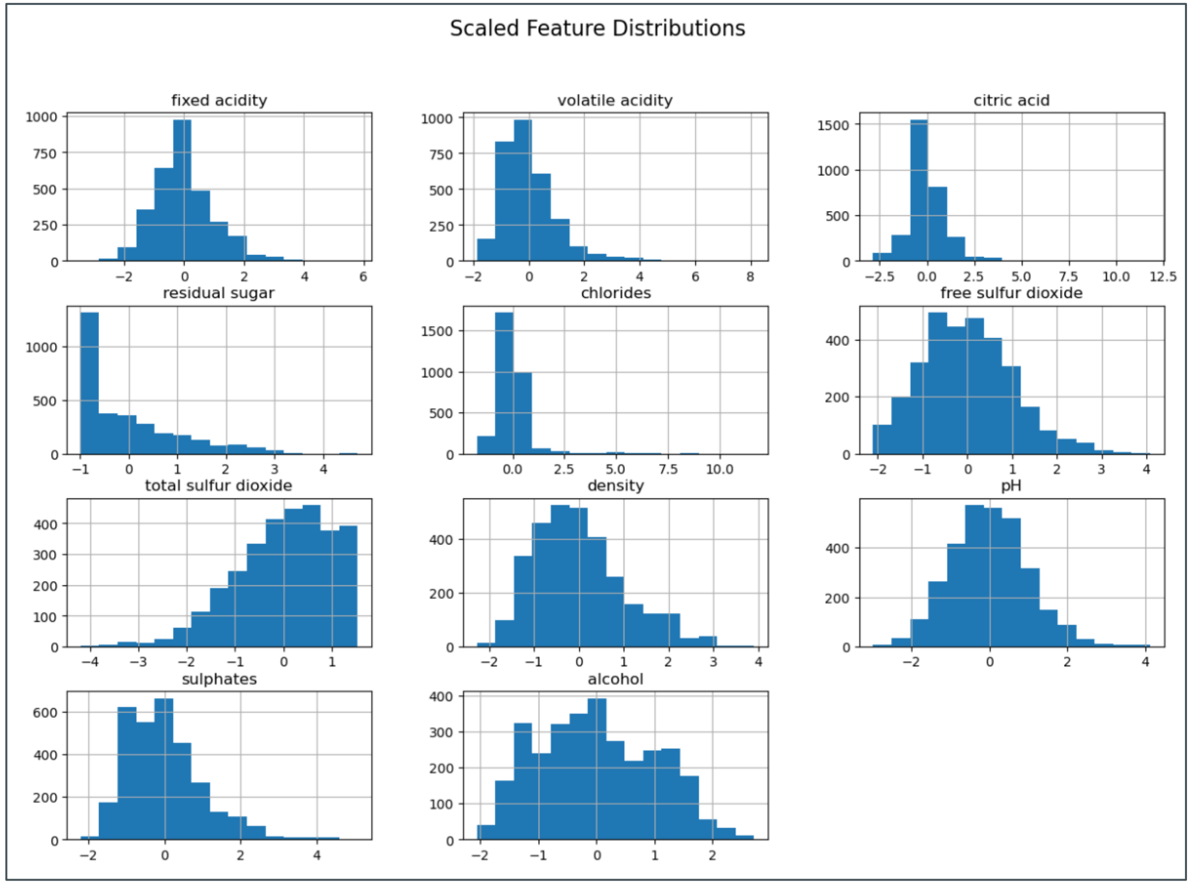
QUICK TAKE

- This data set was chosen because it contains information about the chemical properties of vinho verde, such as acidity, sugar levels, and alcohol content and the data is modeled after the physicochemical wine tests (laboratory-based tests that assess wine quality).
- The number of samples in our dataset after filtering is 3,090. We used Quality as our target and there are 11 total features.
- We cleaned our database by checking for null values, dropping any outliers in ‘total sulfur dioxide’ above 150 as that is likely a data entry error. Our quality values range from 1-10, so we created binary classification to use in our machine learning models. We chose 7 and above as good; 6 and below as not good.
- We checked for multi collinearity among our features for possible features that could be dropped to increase our precision and recall values and filtered our data to only keep wines that pH values between 3 and 4, as this is the nominal level for white wines.

QUICK TAKE

- The 3 acid categories are what combine to make the total acidity in wine. Acid, sugar, and alcohol content are the three most dominant qualities for wine evaluation for the consumer. When adding some information about chemical compounds, such as sulphur dioxide, ph and chloride, the machine can make pretty accurate predictions of a good or not good wine.

Feature distribution across the dataset



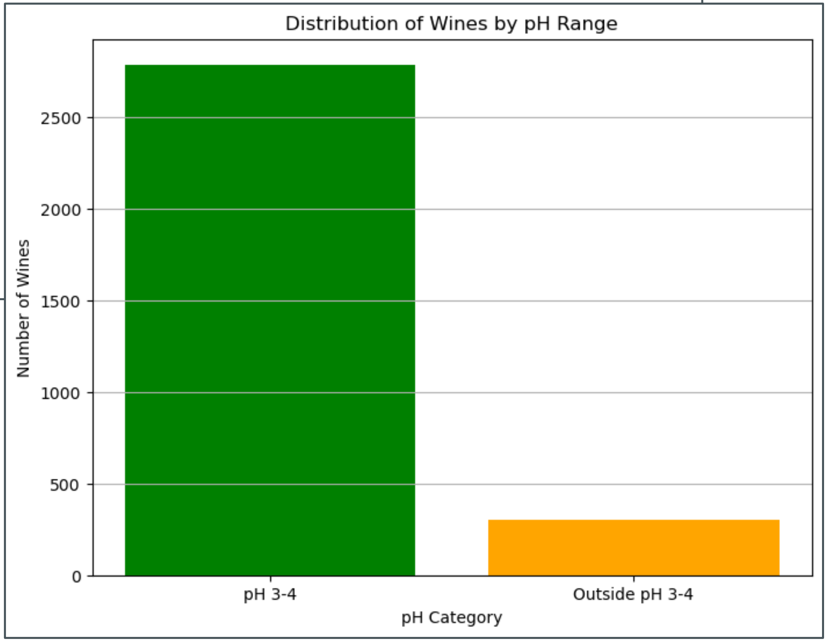
CHECK THE DATA AND PREPPING

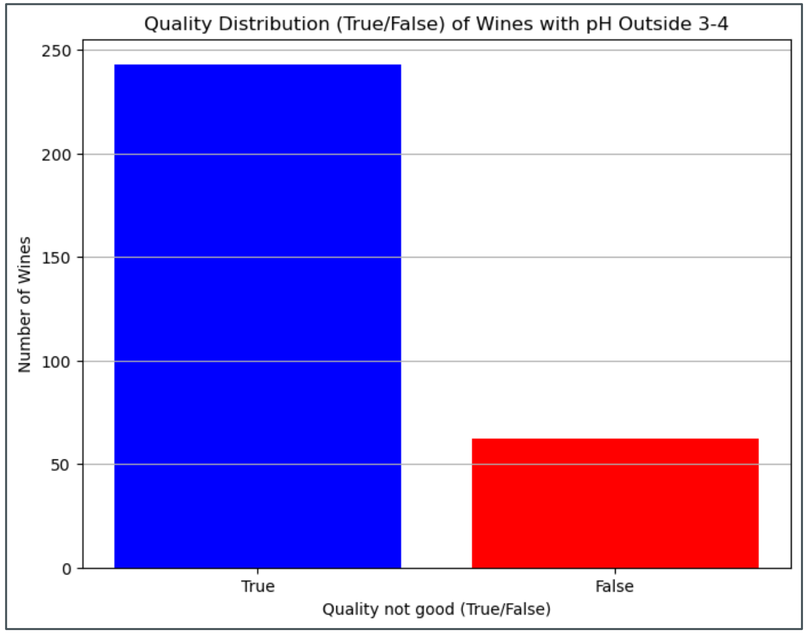
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
# check for nulls ( no nulls), and data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

```
# anything over 150 is likely data entry error, so we are dropping those values
df = df[df['total sulfur dioxide'] <= 150]
```



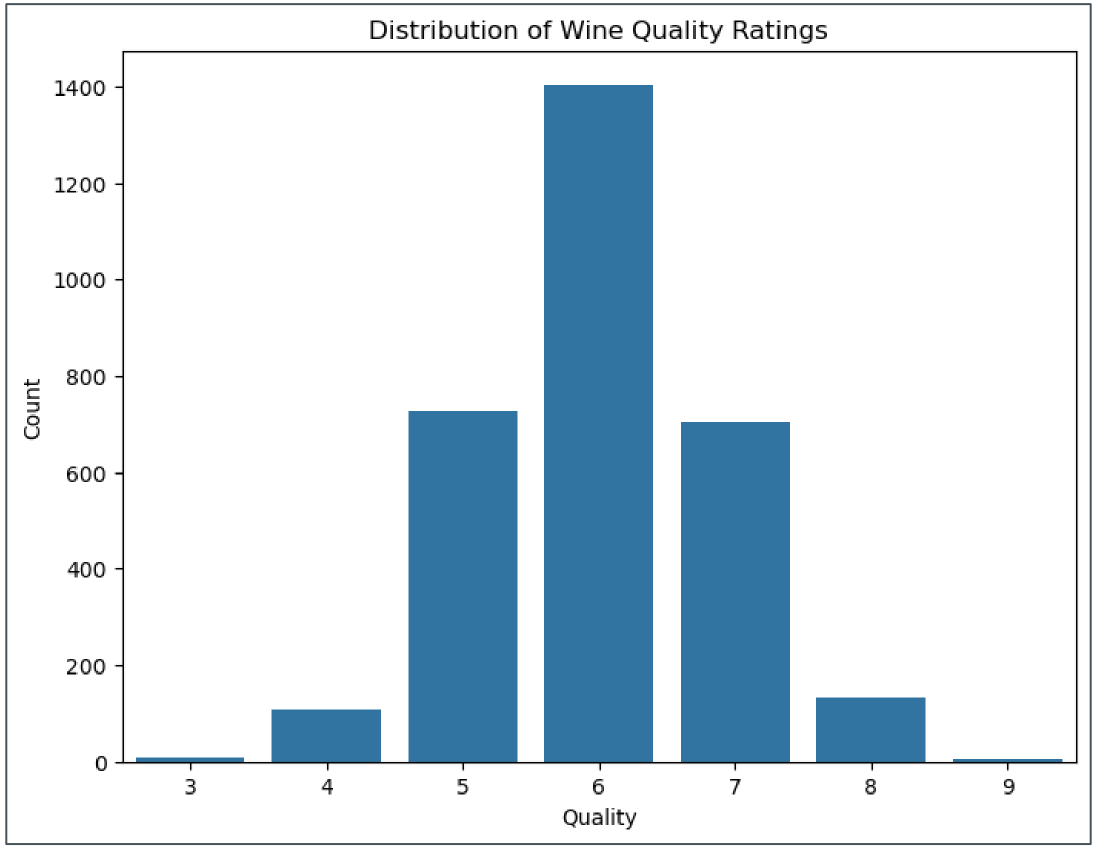


CREATING OUR CLASSIFICATIONS

```python
# The scale for wine quality is 1-10, so we create a binary classification to use for our ML models
df['quality_label'] = np.where(df['quality'] >= 7, 'good', 'not_good')
df.drop('quality', axis=1, inplace=True)
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3090 entries, 1 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         3090 non-null   float64
 1   volatile acidity      3090 non-null   float64
 2   citric acid           3090 non-null   float64
 3   residual sugar        3090 non-null   float64
 4   chlorides             3090 non-null   float64
 5   free sulfur dioxide   3090 non-null   float64
 6   total sulfur dioxide  3090 non-null   float64
 7   density               3090 non-null   float64
 8   pH                    3090 non-null   float64
 9   sulphates             3090 non-null   float64
 10  alcohol               3090 non-null   float64
 11  quality_label         3090 non-null   object
dtypes: float64(11), object(1)
memory usage: 313.8+ KB
```



ENCODING AND SCALING

```python
# separate features from target
y=df['quality_label_not_good']
X=df.drop(columns='quality_label_not_good')
```

```python
# clean it up, scale it etc
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

```python
# convert categorical variables into numerical ones using one-hot encoding to use in our model
df = pd.get_dummies(df, columns=['quality_label'], drop_first=True)
```

```python
df.head()
```

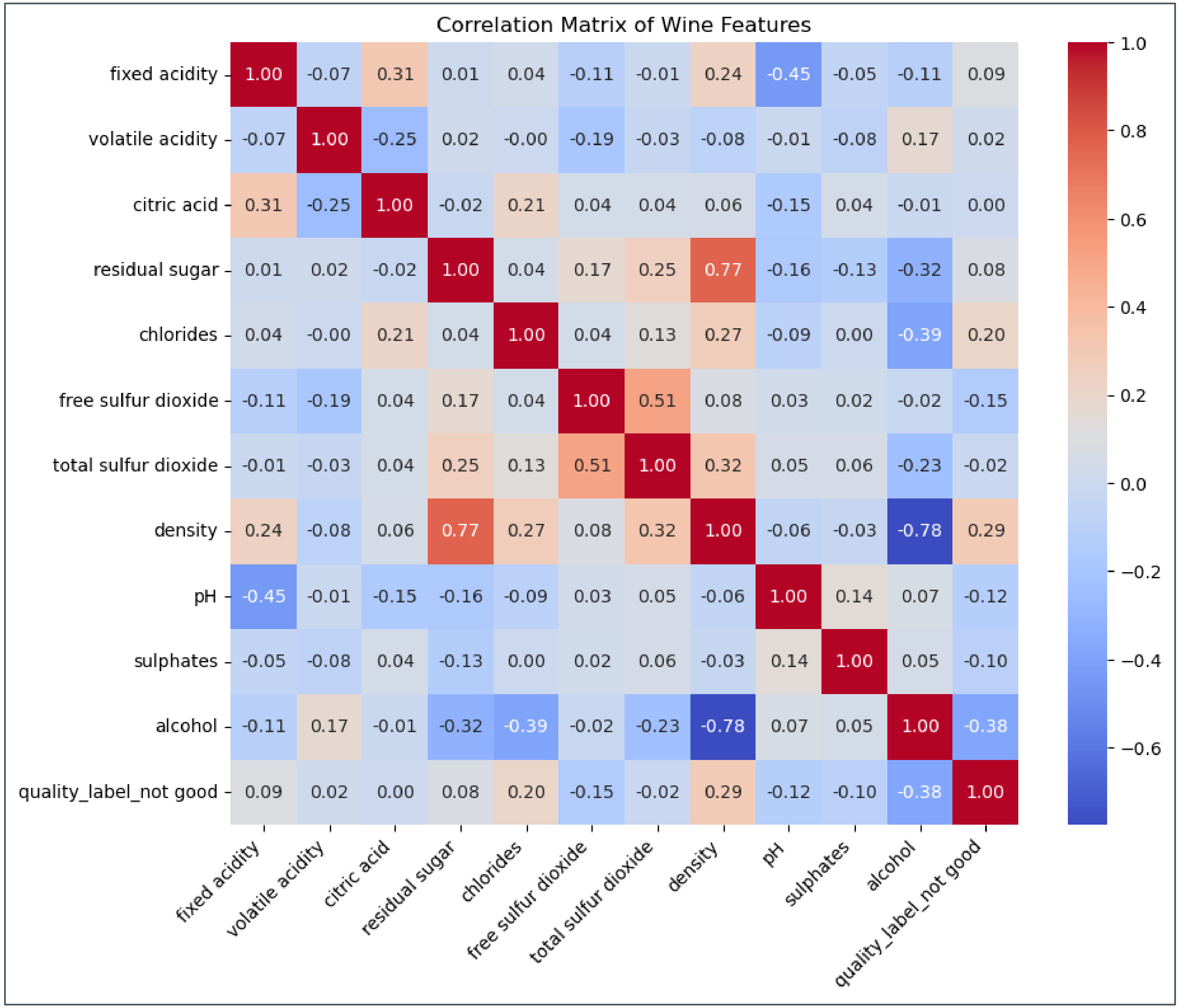
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality_label_not good
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	True
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	True
5	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	True
6	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	True
8	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	True

```python
# check that shape
df.shape
```

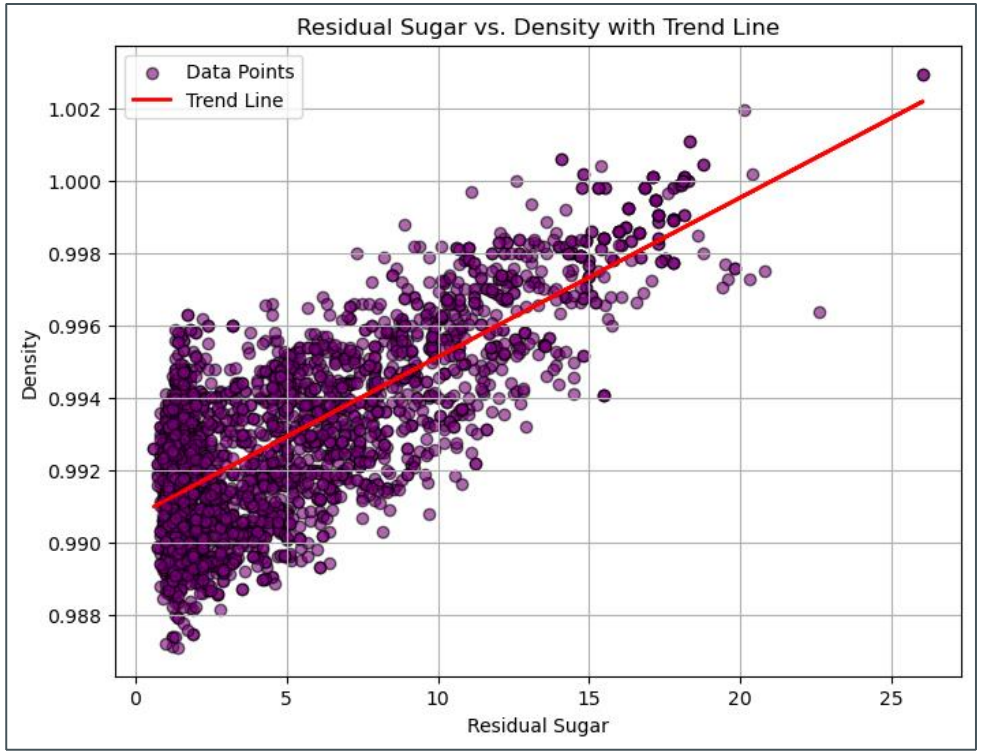
```
(3090, 12)
```

We found 2 interesting relations

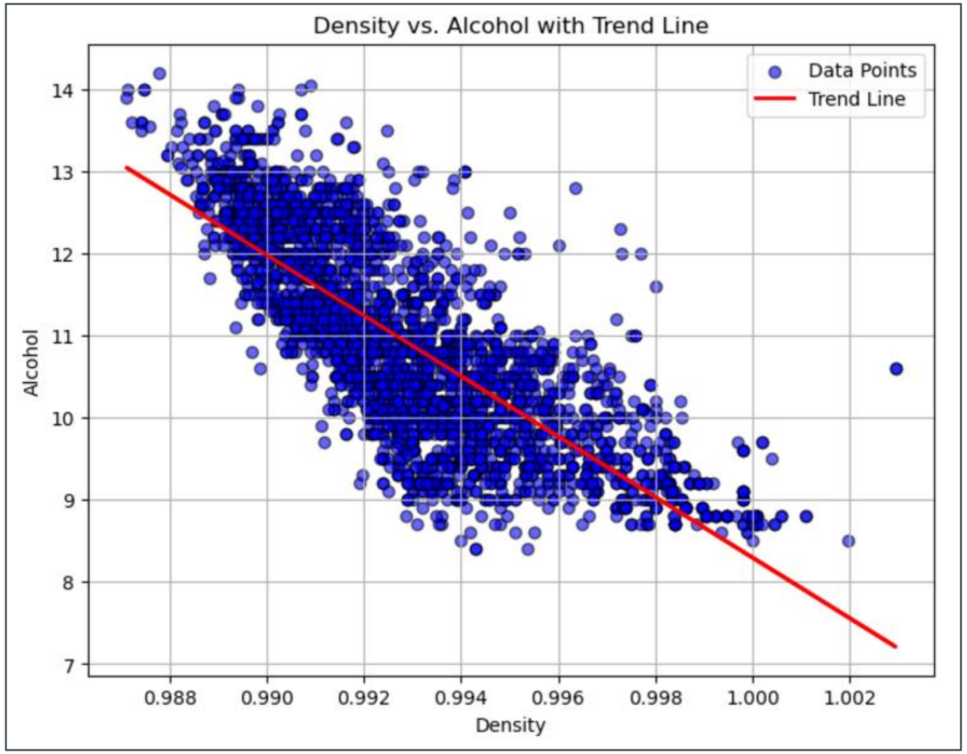
- The sugar amounts and density have a high positive correlation
- Density and alcohol have a high negative correlation



CHECKING CORRELATIONS



Correlation coefficient: 0.7657848803711766



Correlation coefficient: -0.3170400981344824

- Higher sugar content creates a denser wine. Since these two have a strong correlation, we tested our accuracy numbers after dropping 'Sugar' or 'Density'. Surprisingly, both these actions led to lower accuracy numbers.

- The higher alcohol wines have a lower density. Alcohol inherently creates a less dense, less viscous drink.

SPLITTING AND FITTING AND CLASSIFYING....OH MY

```
# train test module on our standard data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
                                                    y,
                                                    random_state=1776,
                                                    stratify=y)
X_train.shape
```

```
(2317, 11)
```

- Use 'stratify=y' to maintain the same class distribution as the original dataset
- Initial numbers weren't great, so we attempted feature engineering as mentioned previously – all these negatively impacted our score, so we tried several different models and approaches to get better accuracies

```
# logistic regress
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='lbfgs',
                                max_iter=200,
                                random_state=1776)
classifier.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=200, random_state=1776)
```

```
# predictions and confusion matrix
testing_predictions = classifier.predict(X_test)
test_matrix = confusion_matrix(y_test, testing_predictions)
print(test_matrix)
```

```
[[ 78 133]
 [ 56 506]]
```

True Negatives (TN) False Negatives(FN)
False Positives (FP) True Positives (TP)

```
# classification report for our standard data
test_report = classification_report(y_test, testing_predictions)
print(test_report)
```

	precision	recall	f1-score	support
False	0.58	0.37	0.45	211
True	0.79	0.90	0.84	562
accuracy			0.76	773
macro avg	0.69	0.64	0.65	773
weighted avg	0.73	0.76	0.74	773

SPLITTING AND FITTING AND CLASSIFYING CONT'D

Extreme Gradient Boosting was our highest performing model with an overall accuracy of 84%

```
# xgboost model - our best performing model
from xgboost import XGBClassifier

xgb_clf = XGBClassifier(random_state=1776, scale_pos_weight=len(y_train[y_train == 0]) / len(y_train[y_train == 1]))
xgb_clf.fit(X_train, y_train)
predictions = xgb_clf.predict(X_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

       False       0.71      0.68      0.69       211
        True       0.88      0.90      0.89       562

    accuracy                           0.84       773
   macro avg       0.80      0.79      0.79       773
weighted avg       0.83      0.84      0.84       773
```

Average wines = True
High quality wines = False

- LogisticRegression and SVM had the lowest overall accuracy. We also tried RandomForest and StratifiedKFold (to preserve distribution) with XGBoost
- XGBoost is powerful due to its ability to learn from mistakes to prevent overfitting and optimize for efficiency. Each new tree focuses on correcting errors from the previous trees and each contributes to improving performance.

SVM

```
# running svm model
from sklearn.svm import SVC

svm_clf = SVC(kernel='rbf', C=1, gamma='scale', random_state=1776)
svm_clf.fit(X_train, y_train)
predictions = svm_clf.predict(X_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

       False       0.68      0.33      0.44       211
        True       0.79      0.94      0.86       562

    accuracy                           0.77       773
   macro avg       0.74      0.64      0.65       773
weighted avg       0.76      0.77      0.75       773
```

RF

```
# running random forest model
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100, random_state=1776)
rf_clf.fit(X_train, y_train)
predictions = rf_clf.predict(X_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

       False       0.79      0.55      0.65       211
        True       0.85      0.94      0.89       562

    accuracy                           0.84       773
   macro avg       0.82      0.75      0.77       773
weighted avg       0.83      0.84      0.83       773
```

StratifiedKFold with XGB

```
# classification report for the stratified data
strat_test_report = classification_report(y_test_strat, strat_predictions)
print(strat_test_report)
```

```
              precision    recall  f1-score   support

       False       0.64      0.66      0.65       169
        True       0.87      0.86      0.87       449

    accuracy                           0.81       618
   macro avg       0.76      0.76      0.76       618
weighted avg       0.81      0.81      0.81       618
```

VINOVISTA WINE ANALYSIS

```
# xgboost model - our best performing model
from xgboost import XGBClassifier

xgb_clf = XGBClassifier(random_state=1776, scale_pos_weig
xgb_clf.fit(X_train, y_train)
predictions = xgb_clf.predict(X_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

       False       0.71      0.68      0.69       211
        True       0.88      0.90      0.89       562

    accuracy                           0.84       773
   macro avg       0.80      0.79      0.79       773
weighted avg       0.83      0.84      0.84       773
```

precision: how many predicted positives were actually correct

recall: how many actual positives were identified

f1-score: combines precision and recall into a single number, particularly useful in imbalanced data like ours

Average wines = True
High quality wines = False

Our model performed better in identifying "average" wines (True), with a precision of 0.88 and a recall of 0.90, leading to an F1-score of 0.89.

For "good" wines (False), the precision was 0.71, recall was 0.68, and the F1-score was 0.69. These metrics indicate that while the model is more effective at identifying "bad" wines, it still performs reasonably well for "good" ones.

VINOVISTA WINE ANALYSIS

- Our dataset comes from the food industry, where quality ratings from tasters are inherently subjective. Despite this, we believe our model's results are very strong. Additionally, the overrepresentation of average wines in the data has also led to an accuracy imbalance, particularly when predicting good wines.

Ideas for how to use our model!

- It could be leveraged for rapid, small-batch research and development.
- Efficiently allocate for different wine groups. For example, delivering high-quality wine to upscale market sectors or average-quality wines to value-driven markets.
- Quick classifying of wines will save time and money on the identifying and packaging process, addresing both marketing and production strategies.
- Quality ratings from tasters take time to receive. Our model allows for swift detection and prediction of average wines that could be allocated as table wine batches, more efficiently targeting high-volume clients, or customers seeking affordable wines.