Wine Quality

Guy Kahana & Anat Peled

Project IV: Classification Models in ML & Advanced techniques

The Dataset

- The dataset was taken from Kaggle, but originally downloaded from the UCI Machine Learning Repository.
- The dataset refers to red and white variants of the Portuguese "Vinho Verde" wine. The reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).
- The goal of our project is to predict the wine quality using its features
- This prediction may assist wine makers in the producing process

Acknowledgements: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. *Modeling wine preferences by data mining from physicochemical properties*. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

The Dataset

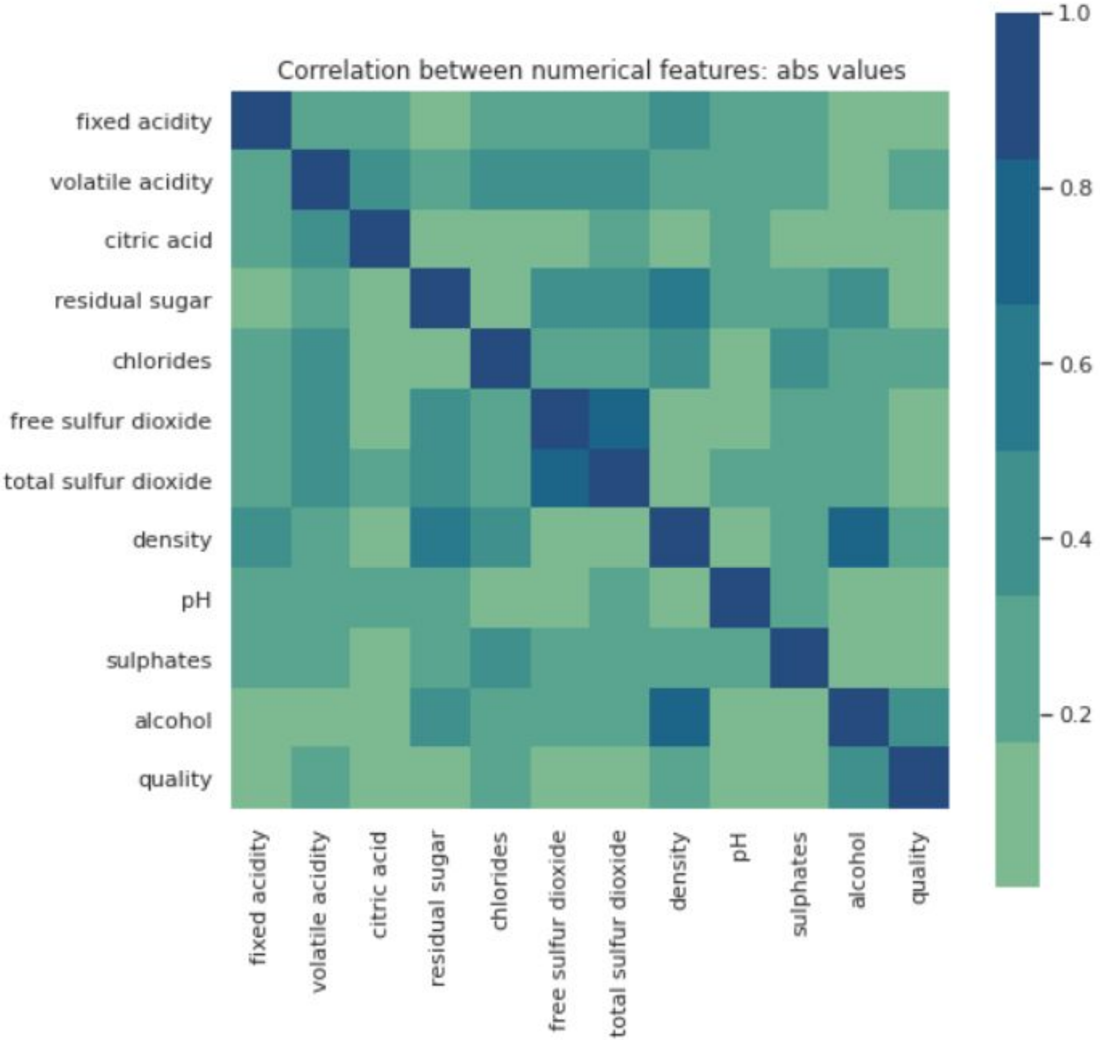
- Few nulls are present
- 12 numerical feature, 1 object (wine type)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

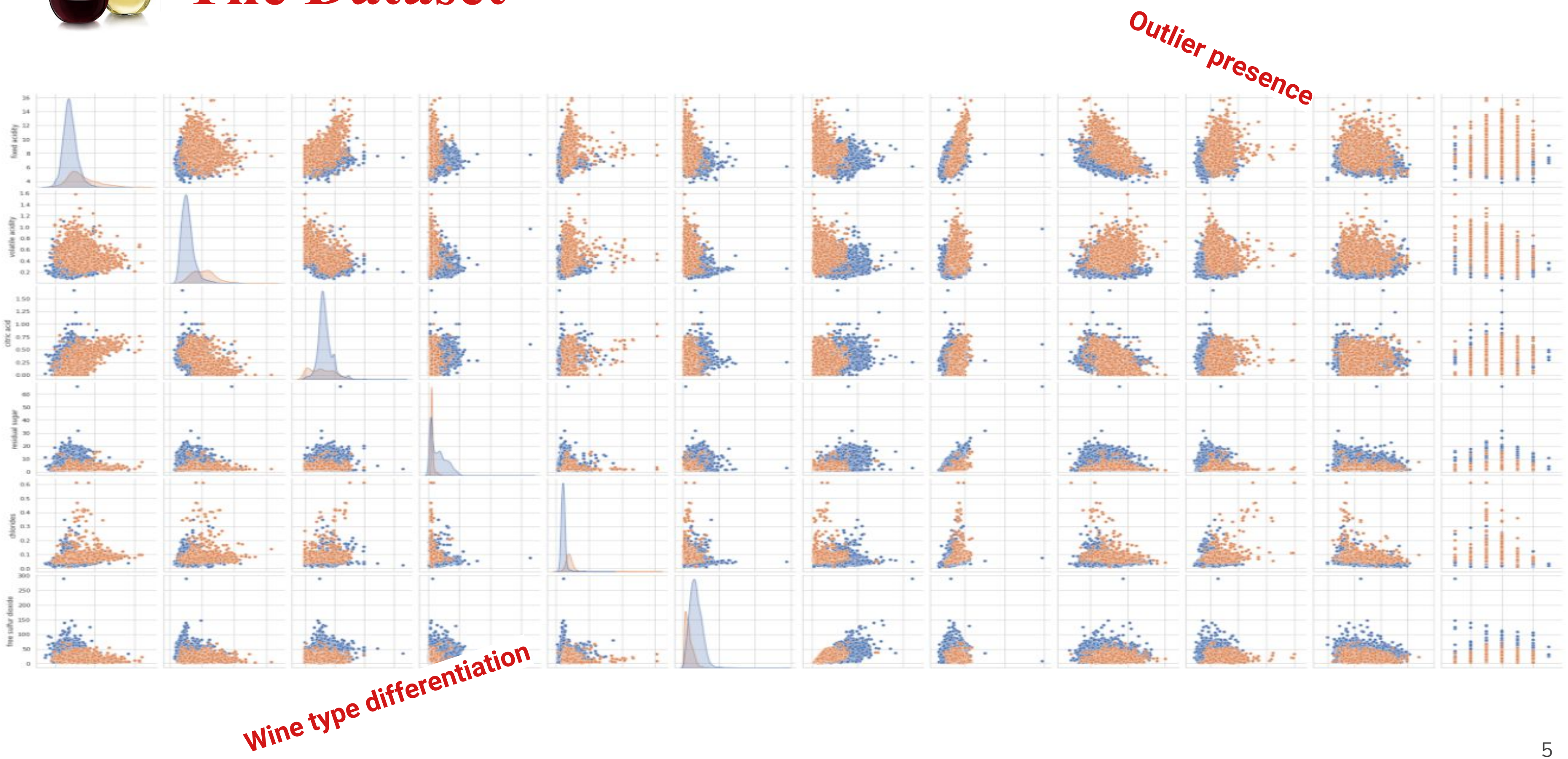
	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28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

The Dataset

- Linear correlation is limited
- Alcohol (0.44), density (0.31), volatile acidity (0.27) and chlorides (0.20) are the highest correlated features



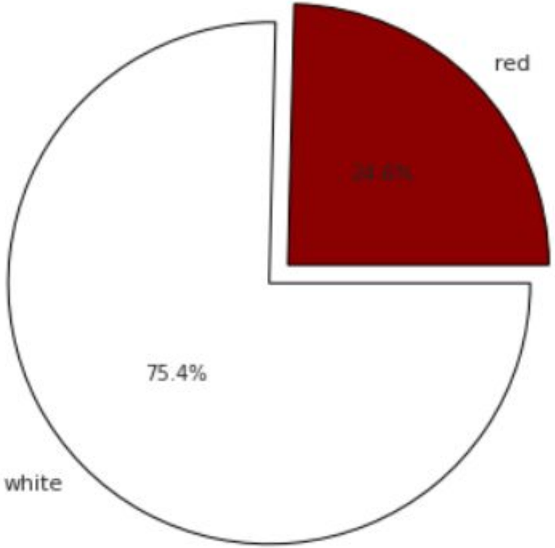
The Dataset



Type

.

- Type: Two types of wines: Red wine & white wine
- Ratio of 3:1 with white dominance

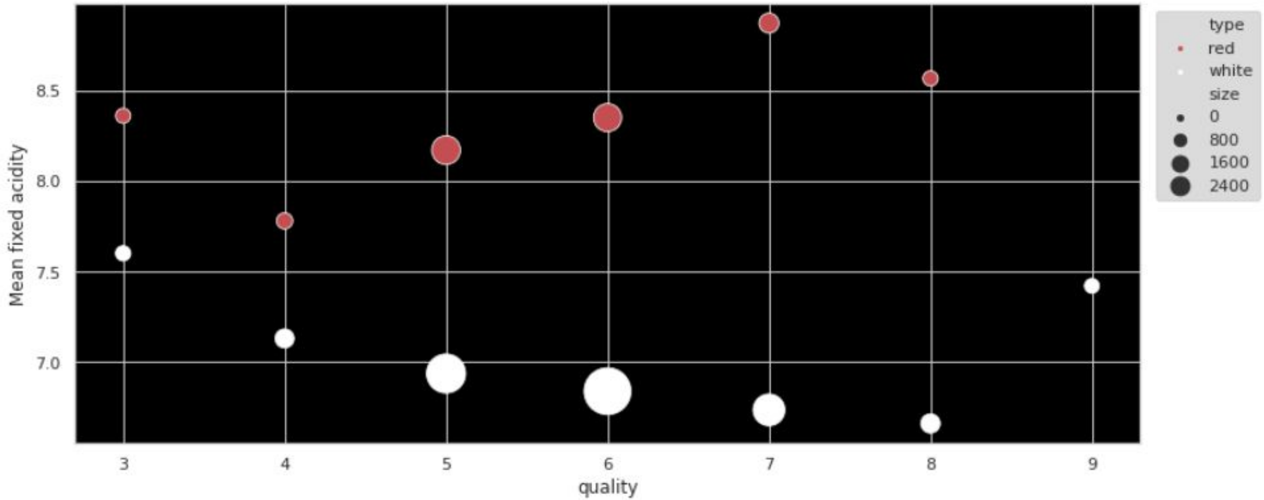


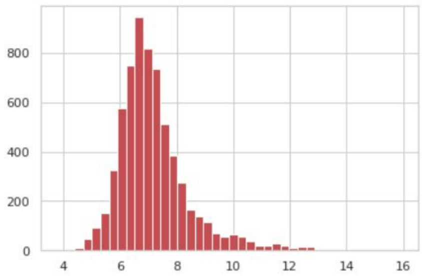
Fixed Acidity

- Fixed acids include tartaric, malic, citric, and succinic acids which are found in grapes . Reducing acids significantly might lead to wines tasting flat.

- There is consistent gap in mean values for red & white wines

count	6,487.00
mean	7.22
std	1.30
min	3.80
25%	6.40
50%	7.00
75%	7.70
max	15.90

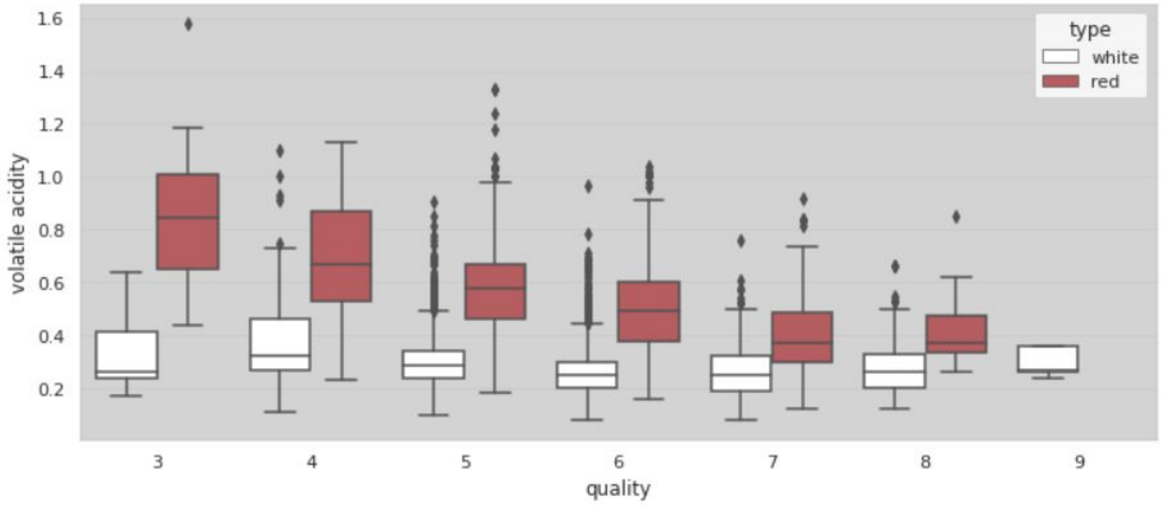




Volatile Acidity

count	6,489.00
mean	0.34
std	0.16
min	0.08
25%	0.23
50%	0.29
75%	0.40
max	1.58

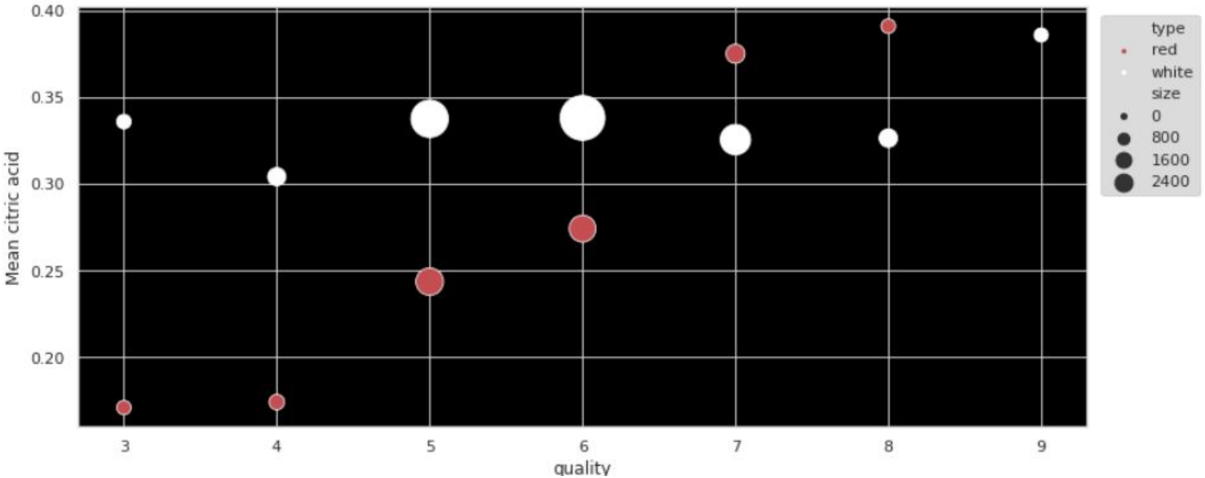
- Excess of volatile acids are undesirable and lead to unpleasant flavour
- Red wine volatile acidity is higher than white wines and tends to descent with quality
- Mean white wines fixed acidity is almost linear

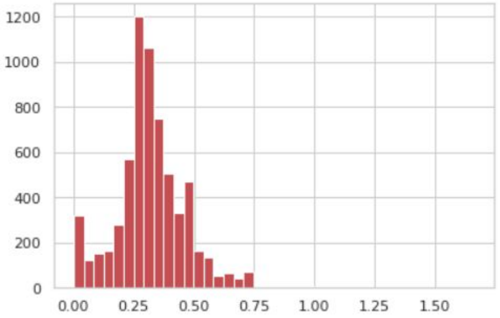


Citric Acid

- This is one of the fixed acids which gives a wine its freshness. Usually most of it is consumed during the fermentation process and sometimes it is added separately to give the wine more freshness.
- At quality levels <5 - mean citric acid levels are relatively different

count	6,494.00
mean	0.32
std	0.15
min	0.00
25%	0.25
50%	0.31
75%	0.39
max	1.66

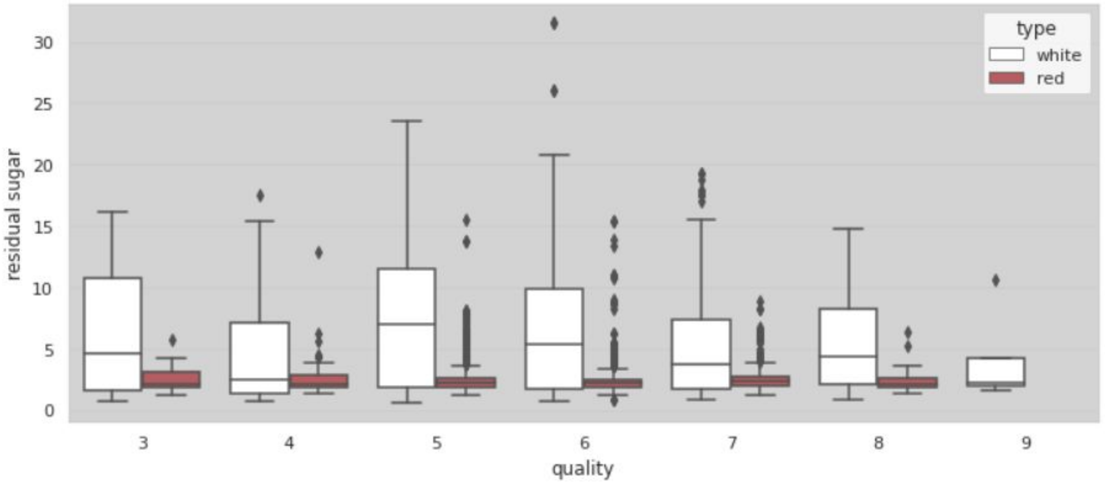


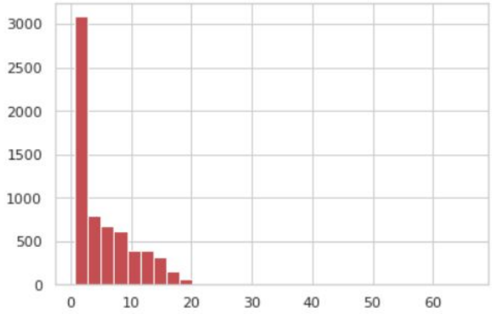


Residual Sugar

count	6,495.00
mean	5.44
std	4.76
min	0.60
25%	1.80
50%	3.00
75%	8.10
max	65.80

- This typically refers to the natural sugar from grapes which remains after the fermentation process stops, or is stopped.



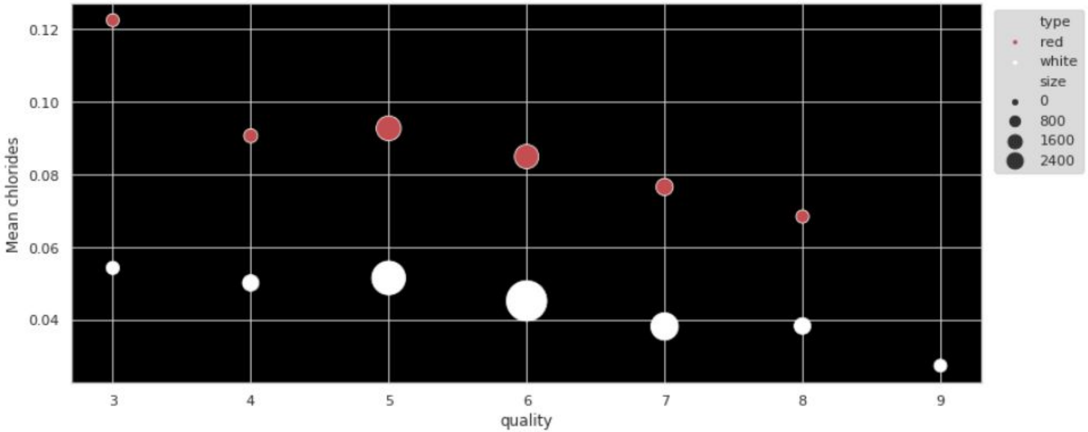


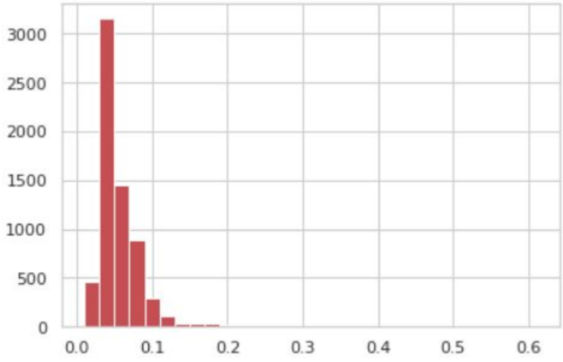
** outlier ~60 removed for plotting purposes*

Chloride**s**

count	6,495.00
mean	0.06
std	0.04
min	0.01
25%	0.04
50%	0.05
75%	0.07
max	0.61

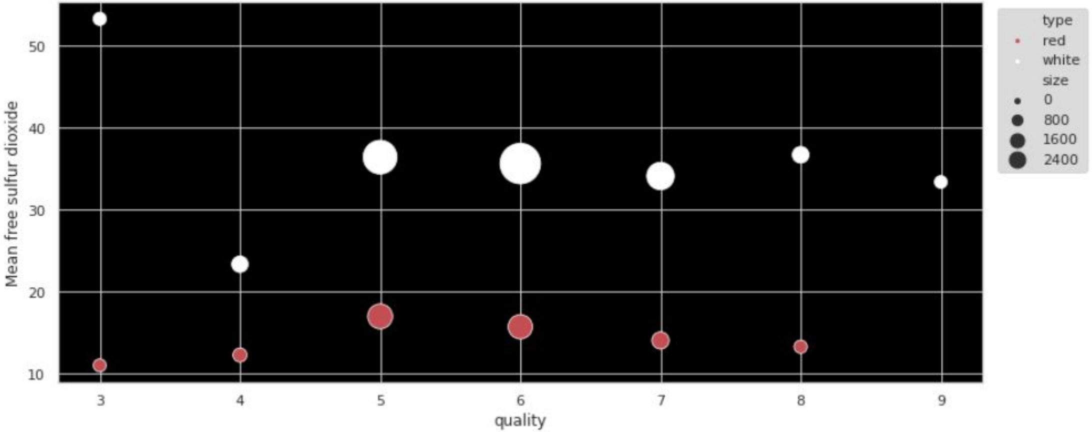
- This is the chloride concentration in the wine
- Mean chloride level decline with quality increase
- Red Wines have higher chloride levels than white wines

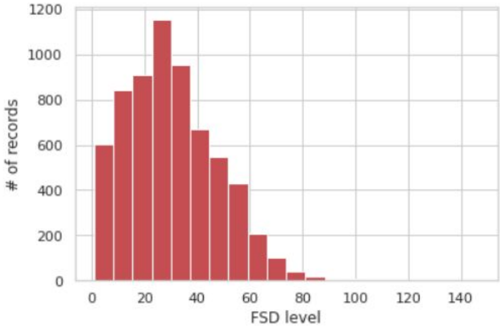




Free Sulfur Dioxide (SO_2)

- Also known as sulfites ,too much of it is undesirable and gives a pungent odour.
- White wines has higher free SO_2 levels
- Mean level of SO_2 in white wines with quality = 4 is showing anomalous

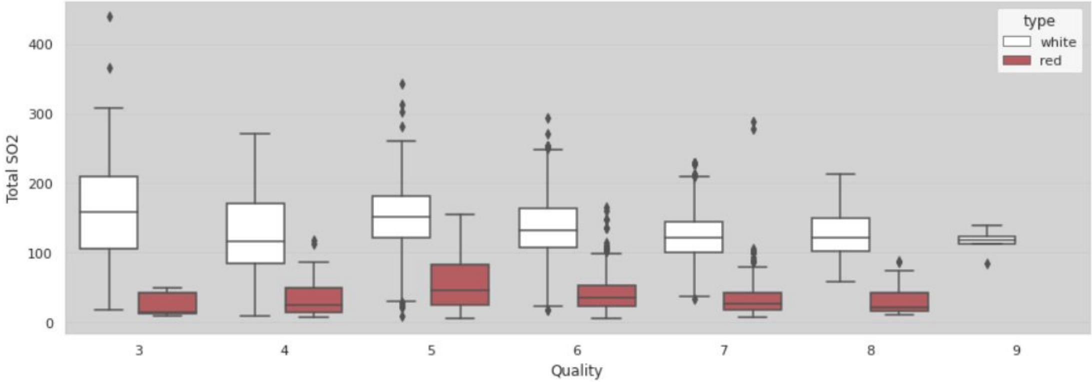


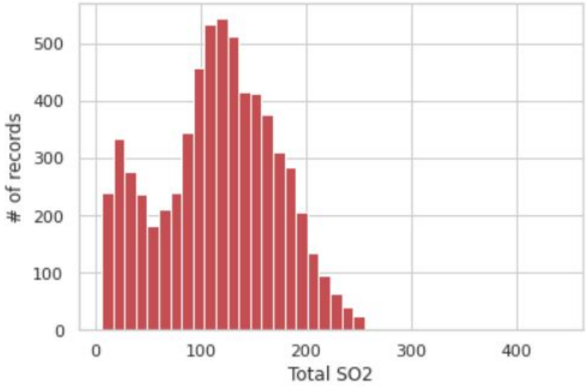


Total Sulfur Dioxide

- This is the sum total of the bound and the free sulfur dioxide. This is mainly added to kill harmful bacteria and preserve quality and freshness.

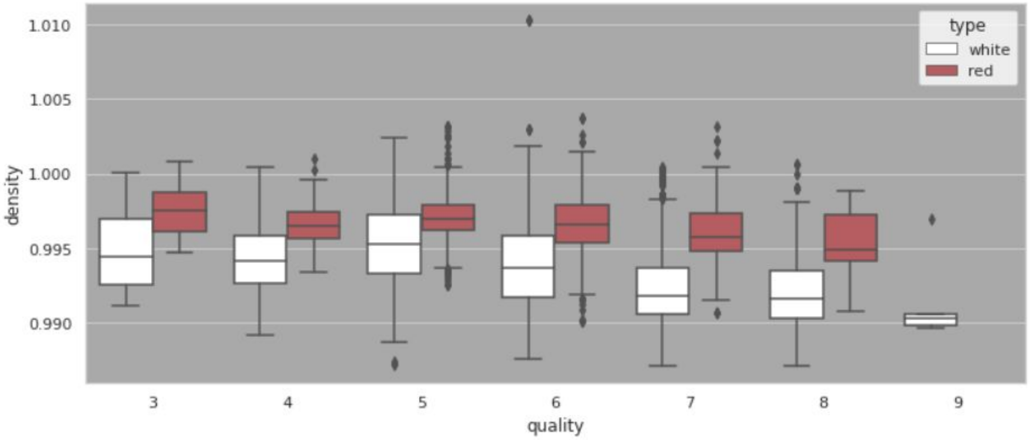
- Total mean SO_2 levels are higher in white wines than in red wines, but remain relatively constant among quality groups.

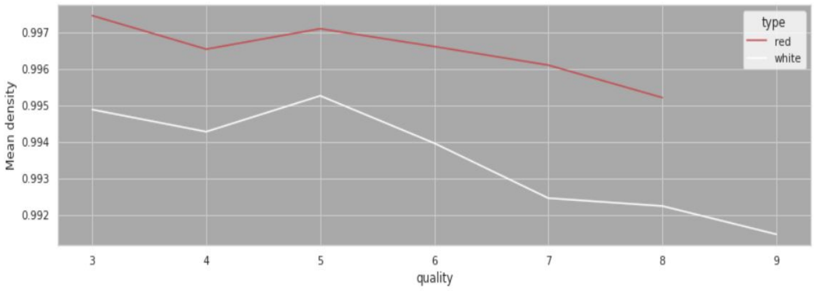




Density

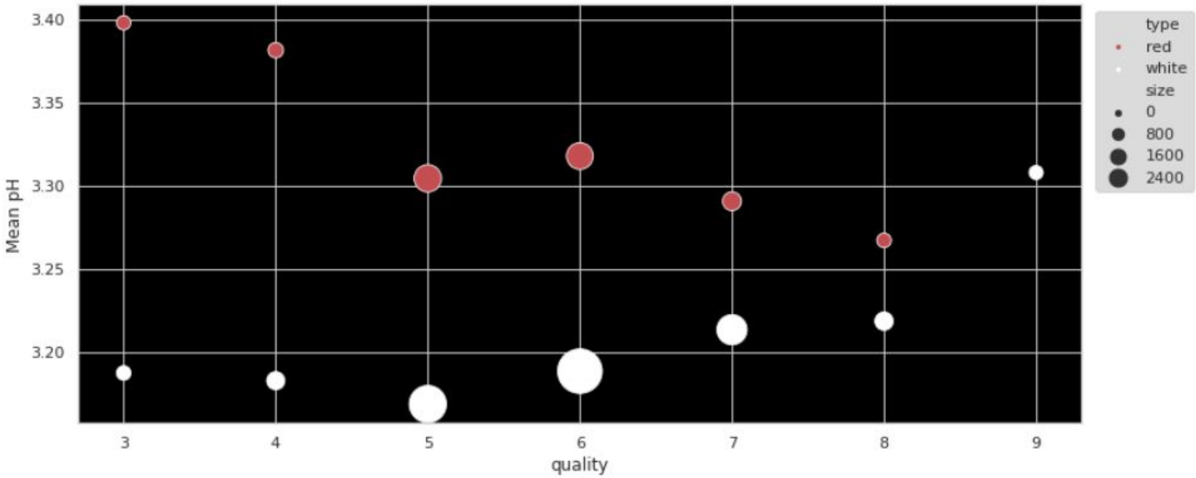
- This can be represented as a comparison of the weight of a specific volume of wine to an equivalent volume of water.
- Mean density values decline as quality increase in both types of wines.
- Majority of values between 0.99 and 1.005



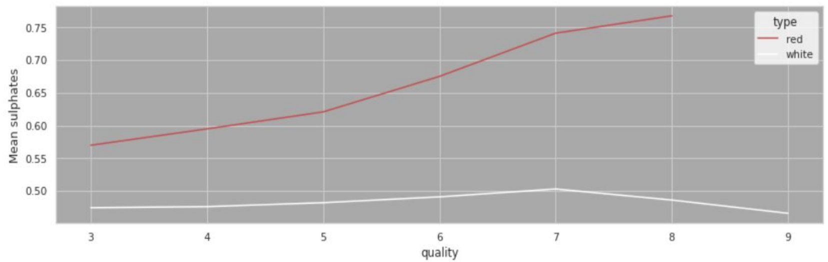


pH

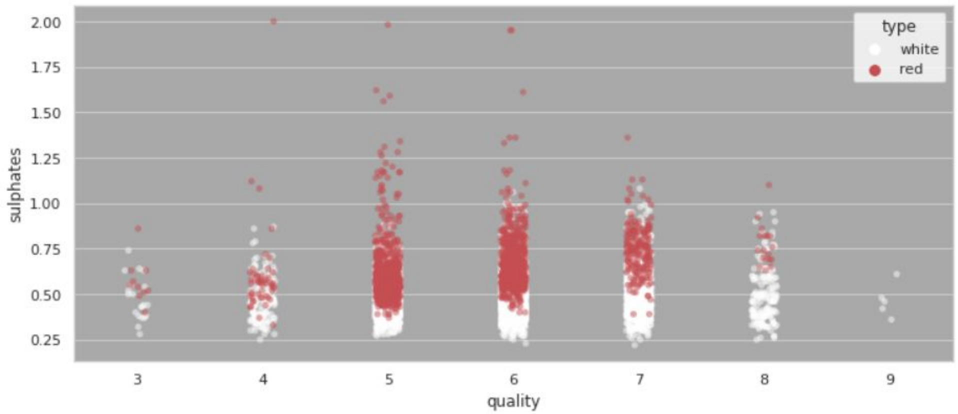
- Also known as the potential of hydrogen, this is a numeric scale to specify the acidity or basicity the wine. Most wines have a pH between 2.9 and 3.9 and are therefore acidic.
- Red wines are less acidic than white wines in average when it comes to low qualities levels



Sulphates



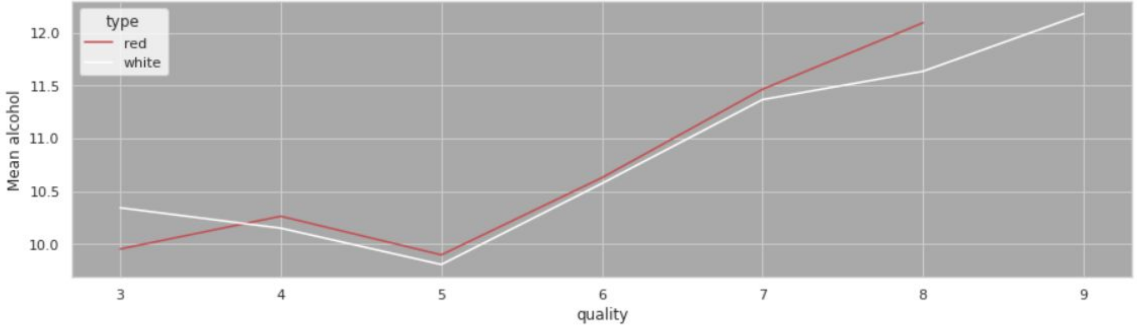
- These are mineral salts containing sulfur. They are connected to the fermentation process and affects the wine aroma and flavour.
- Red wines have higher sulphates values than white wines.
- Mean sulphates level is constant in white wines, and graduating in red wines.

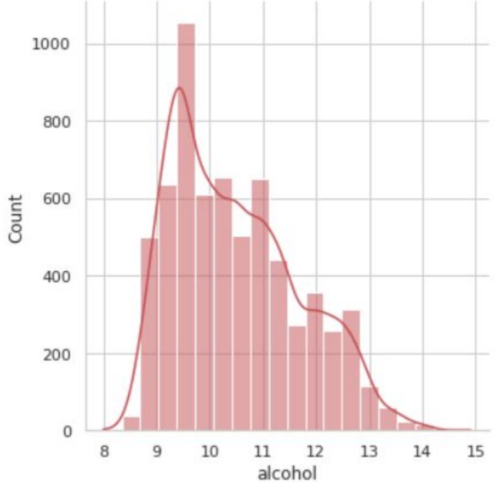


Alcohol

count	6497.000000
mean	10.491801
std	1.192712
min	8.000000
25%	9.500000
50%	10.300000
75%	11.300000
max	14.900000

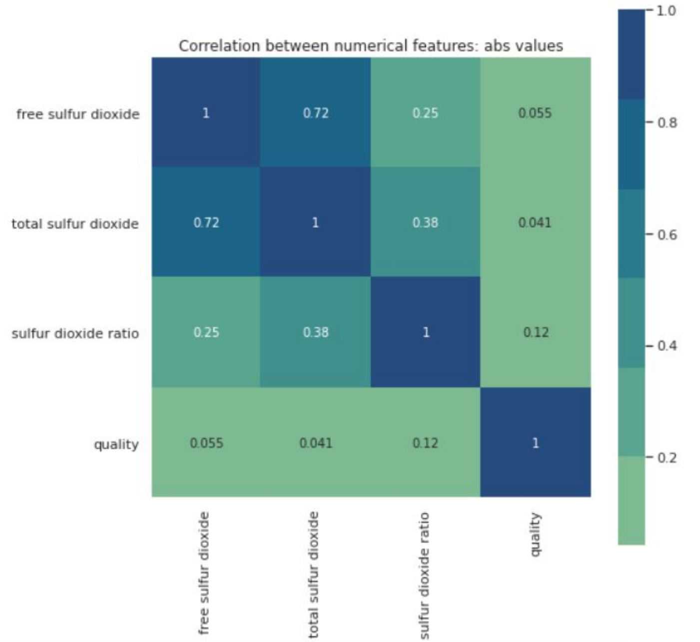
- It's usually measured in % vol or alcohol by volume (ABV).
- Alcohol % doesn’t vary between wine types, but tend to increase with quality

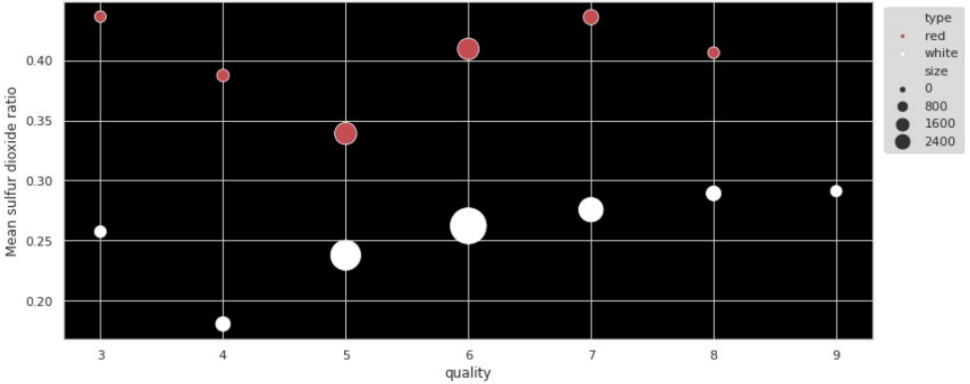


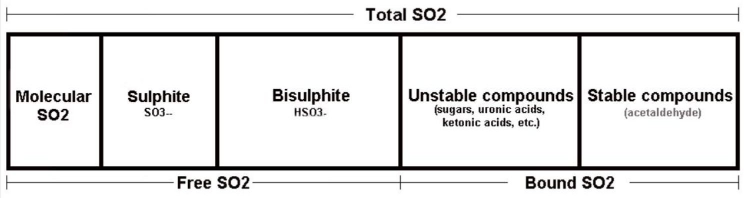


Sulfur Dioxide Ratio

- Free SO_2 / Total SO_2
- Higher ratio in red wines
- Stronger correlation to quality

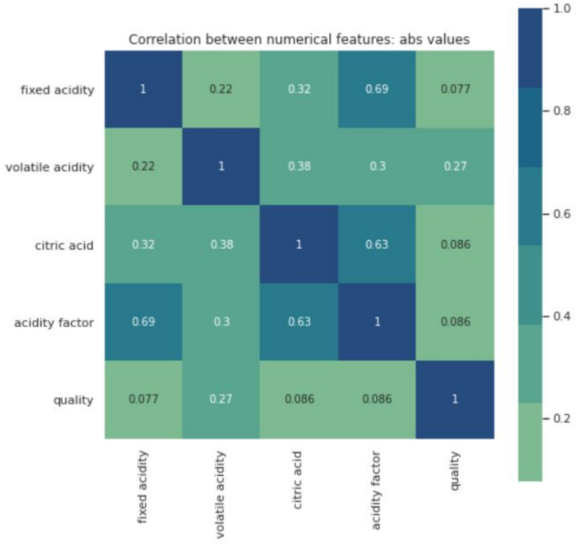


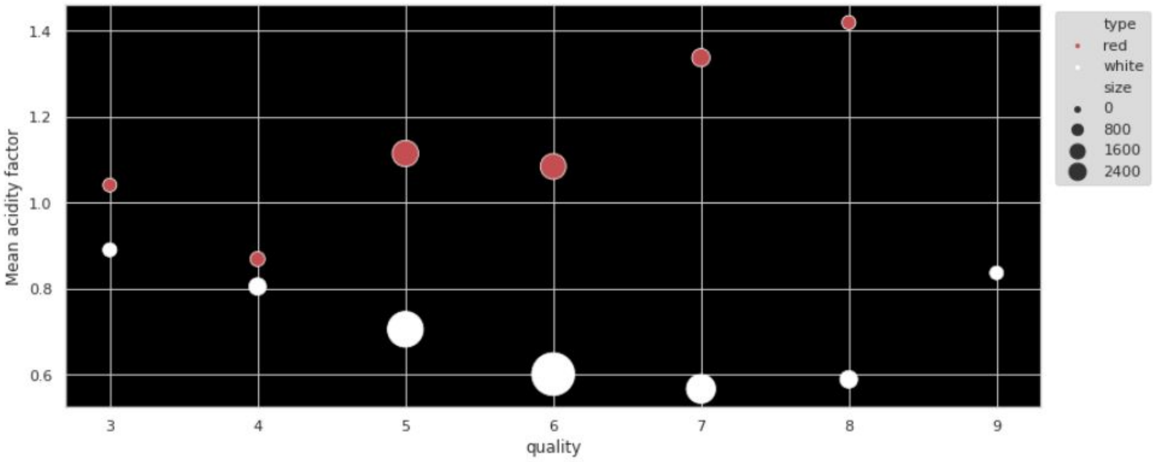




Acidity Factor

- Multiply of the 3 acid features to create a new feature
- Fixed at 3 & 4, drift apart at higher quality levels



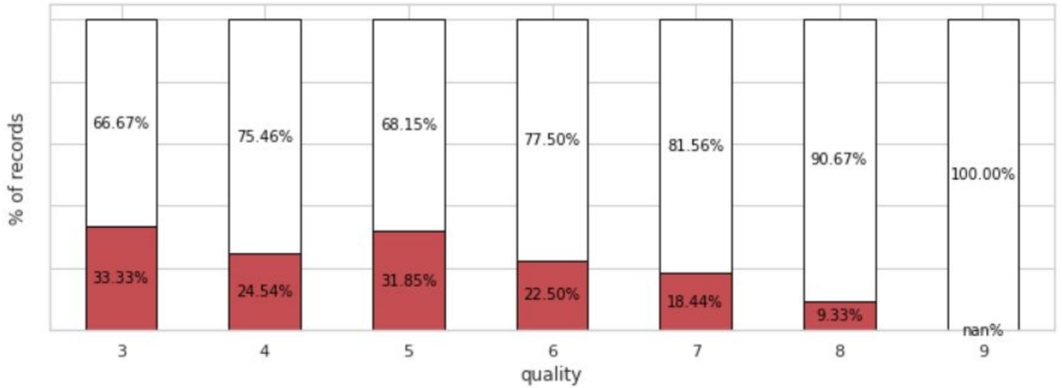


Quality (Target)

- Wine experts graded the wine quality between 0 (very bad) and 10 (very excellent). The eventual quality score is the median of at least three evaluations made by the same wine experts.
- imbalanced data
- 9 is applicable at white wines only

  (with 5 observations!)

	count	mean	std	min	25%	50%	75%	max
Red Wines	1599.0	5.636023	0.807569	3.0	5.0	6.0	6.0	8.0
White Wines	4898.0	5.877909	0.885639	3.0	5.0	6.0	6.0	9.0



Data Insights

- Most features show different behavior regarding the wine type, but the type by itself can’t predict the quality.
- Should we predict the wine type, one could expect high prediction rates.
- Quality prediction rate is expected to be low, due to imbalanced dataset
- The ability to predict categories 3, 4, 8 & 9 is expected to be very low due to small sample size
- Imputing nulls must take into consideration the wine type
- Outliers removals is essential in almost 50% of features
- Scaling is required due to different values scale (<1, 10s, 100s)
- The engineered features were aimed to unite similar indicative feature to a more solid feature. But while in sulfur we decided to drop the original features, at acidity we will use it all.
- **We suspect that applying different prediction models on white and red wines may get higher prediction score.**

Pre Processing

Pre-Processing

Apply in	Transformer	Alcohol	Acidity factor	citric acid	chlorides	Density	fixed acidity	Free Sulfur Dioxide
Train	Outlier_limit			V	V	V		V
Train + Test	SimpleImputer			V	V			
	MinMaxScaler	V	V	V	V	V	V	
	Other		eng. Feature					

Apply in	Transformer	Sulfur Dioxide Ratio	pH	residual sugar	Sulphates	Total Sulfur Dioxide	type	volatile acidity
Train	Outlier_limit			V	V	V		V
Train + Test	SimpleImputer		V	V				V
	MinMaxScaler		V	V	V			V
	Other	eng. Feature					[0,1]	

* Engineered feature

Pre-Processing: Outliers removal

- Dictionary with modified limits to features
- limit_value function
- FunctionTransformer applied

```
def limit_value(X, **val_dict):
    """This function recieves a dataframe and returns
       it without the outliers
    """
    for col, val in val_dict.items():
        X = X.loc[X[col] < val, :]
    return X
```

```
# Dictionary of outliers
outlier_dict = {'free sulfur dioxide': 150,
                'total sulfur dioxide': 400,
                'density': 1.01,
                'sulphates': 1.75,
                'volatile acidity':1.5,
                'citric acid': 1.2,
                'residual sugar': 50,
                'chlorides': 0.5}

# Clean outliers
outlier_limit = FunctionTransformer(limit_value, kw_args = outlier_dict)
df = outlier_limit.transform(df)
```

Pre-Processing: Feature selection & split

```python
col_to_drop = ['free sulfur dioxide', 'total sulfur dioxide']
df = df.drop(labels=col_to_drop, axis=1)
```

```python
# split the data
X_train, X_test, y_train, y_test =
      split(df.drop('quality', axis=1), df['quality'],
            test_size = 0.25,
            random_state = 12345,
            stratify=df['quality'])
```

```
X_Train:  (4849, 12) | y_Train:  4849
X_Test:   (1617, 12) | y_Test:  1617
```

Apply Imputer & Scaler

```
cols = ['fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'density', 'pH',
        'sulphates', 'alcohol', 'acidity factor']

imputer = SimpleImputer(strategy = 'mean')
scaler = MinMaxScaler()

X_train.loc[X_train['type'] == 0] = imputer.fit_transform(X_train.loc[X_train['type'] == 0])
X_train.loc[X_train['type'] == 1,:] = imputer.fit_transform(X_train.loc[X_train['type'] == 1])

X_train.loc[X_train['type'] == 0,cols] = scaler.fit_transform(X_train.loc[X_train['type'] == 0,cols])
X_train.loc[X_train['type'] == 1,cols] = scaler.fit_transform(X_train.loc[X_train['type'] == 1,cols])

X_test.loc[X_test['type'] == 0] = imputer.fit_transform(X_test.loc[X_test['type'] == 0])
X_test.loc[X_test['type'] == 1] = imputer.fit_transform(X_test.loc[X_test['type'] == 1])

X_test.loc[X_test['type'] == 0,cols] = scaler.fit_transform(X_test.loc[X_test['type'] == 0,cols])
X_test.loc[X_test['type'] == 1,cols] = scaler.fit_transform(X_test.loc[X_test['type'] == 1,cols])
```

Plan vs. Reality...

Plan

Over Sampling → Under Sampling → Build special estimator → Run split models

Reality

- Split models are worse!
- Estimator not working
- Over sampling doesn't work!
- (3 different methods!)
- And so was under-sampling
- Indexes don't much in X & Y

And then Dror comes to rescue...

Split Model

- Get 2 model - one for red, one for white
- In fit: fit each population with its own model
- In transform: predict the population, concat the data and return a reinexed y series

```python
class run_estimator (BaseEstimator, TransformerMixin):
  """ This transformer recives a DF(X) and a target(y),
      and split it to two populations: red wines and white wines
  """

  def __init__(self, model_r, model_w, classes=[0,1]):
      self.red_model_ = model_r
      self.white_model_ = model_w
      self.classes = classes


  def fit (self, X, y=None):
      X_red = X[X.type==0].copy()
      y_red = y[X.type==0].copy()
      self.red_model_.fit(X_red, y_red)
      X_white = X[X.type==1].copy()
      y_white = y[X.type==1].copy()
      self.white_model_.fit(X_white, y_white)
      return self

  def predict(self, X):
      X_red = X[X.type==0].copy()
      X_white = X[X.type==1].copy()
      y_red_pred = pd.Series(self.red_model_.predict(X_red),index=X_red.index)
      y_white_pred = pd.Series(self.white_model_.predict(X_white),index=X_white.index)
      y_pred = pd.concat([y_red_pred, y_white_pred], axis=0)
      return y_pred.reindex_like(X)
```

Cohen’s Kappa score

```
kappa_scorer = make_scorer(cohen_kappa_score)
```

$$k \equiv \frac{p_o - p_e}{1 - p_e}$$

Cohen's kappa coefficient (κ) is a statistic which measures inter-rater agreement for qualitative (categorical) items. It is generally thought to be a more robust measure than simple percent agreement calculation, as κ takes into account the possibility of the agreement occurring by chance.

		B: Yes	B: No
A	Yes	a	b
A	No	c	d

		B: Yes	B: No
A	Yes	10	15
A	No	20	05

$$p_{\text{Yes}} = \frac{a+b}{a+b+c+d} \cdot \frac{a+c}{a+b+c+d}$$

$$p_{\text{No}} = \frac{c+d}{a+b+c+d} \cdot \frac{b+d}{a+b+c+d}$$

$$p_e = p_{\text{Yes}} + p_{\text{No}}$$

The observed proportionate agreement is:

$$P_o = \frac{a+d}{a+b+c+d} = \frac{10+5}{50} = 0.3$$

Kappa value interpretation Landis & Koch (1977):

- *<0 No agreement*
- *0 — .20 Slight*
- *.21 — .40 Fair*
- *.41 — .60 Moderate*
- *.61 — .80 Substantial*
- *.81–1.0 Perfect*

Additional reading:https://towardsdatascience.com/interpretation-of-kappa-values-2acd1ca7b18f

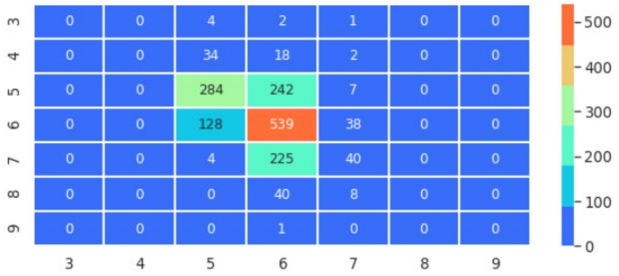
Baseline code without gridsearch

```python
# Run Baseline Logistic Regression with all data
model = LogisticRegression(multi_class = 'ovr')
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# CrossValidation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=123)
scores = cross_val_score(model, X_train, y_train, cv=cv, scoring=kappa_scorer)

# Print the results
print('CV:', scores)
print('CV mean:', scores.mean())
print(f'Train Cohen kappa score is: {cohen_kappa_score(y_train, y_train_pred):.3}')
print(f'Test Cohen kappa score is: {cohen_kappa_score(y_test, y_test_pred):.3}')
cm_plot(confusion_matrix(y_train, y_train_pred), model)
```

```python
def cm_plot(cm, model):
  fig, ax = plt.subplots(figsize = (10,4))
  df_cm = pd.DataFrame(cm, index = [i for i in model.classes_],
                    columns = [i for i in model.classes_])
  sns.set(font_scale=1.2)
  cmap = sns.color_palette("crest")
  ax = sns.heatmap(df_cm, cmap = cmap, annot=True,
                   fmt = 'd',
                   linecolor = 'w', linewidth = 1,
                   annot_kws={"size": 12})
  return plt.show()
```

Output

```
CV: [0.26404414 0.22636681 0.19986942 0.24408421 0.26525797]
CV mean: 0.23992450947633248
Train Cohen kappa score is: 0.251
Test Cohen kappa score is: 0.235
```



Model Results: LogisticRegrssion

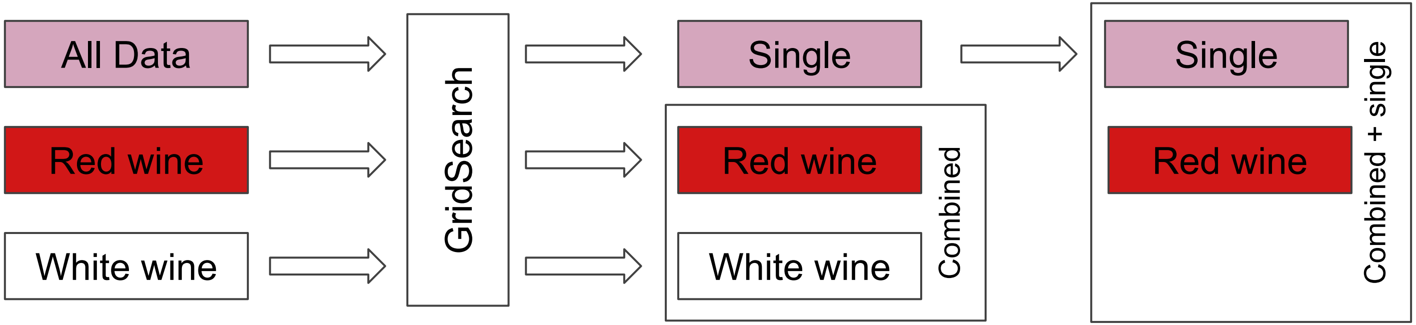
```
# Logistic Regression Classifier
model_w = LogisticRegression(multi_class = 'ovr', random_state=42, penalty='l2', C=1,
                             class_weight={3:2, 4:5, 5:6, 6:6, 7:8, 8:4, 9:1})
model_r = LogisticRegression(multi_class = 'ovr', random_state=42, penalty='l2', C=1.6,
                             class_weight={3: 1, 4: 3, 5: 5, 6: 7, 7: 9, 8: 1})
```

	Single: Baseline	Red	White	Combined
Train	0.251	0.348	0.279	0.301
Test	0.239	0.326	0.249	0.269
Mean CV	0.24	0.297	0.277	0.281

Model Results: VotingCalssifier

```
classifiers = [('LR', LogisticRegression(multi_class='ovr', random_state=42)),
               ('DTC', DecisionTreeClassifier(max_depth=5, random_state=42)),
               ('SVC', SVC(probability=True, random_state=42))]
```

	Single	Red	White	Combined	Combined+Single as white
Train	0.687	0.451	0.319	0.36	0.742
Test	0.322	0.326	0.261	0.281	0.306
Mean CV	0.33	0.308	0.26	0.281	0.331



Model Results: XGBoost

	Single	Red	White	Combined	Combined+Single as white
Train	0.999	0.374	0.735	0.658	0.864
Test	0.418	0.291	0.414	0.386	0.447
Mean CV	0.45	0.299	0.398	0.38	0.421

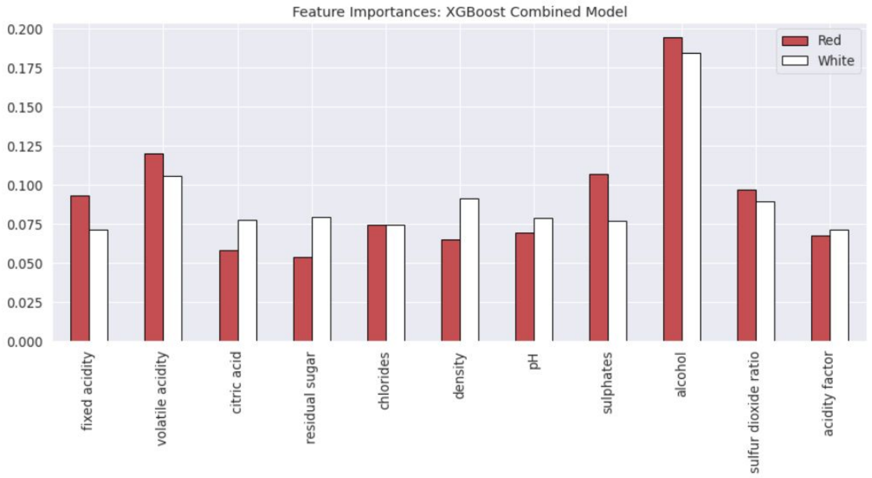
```
model_r = XGBClassifier(random_state=0, max_depth=2,
                        n_estimators=30, colsample_bytree=0.8,
                        subsample = 0.3)
model_w = XGBClassifier(random_state=0, max_depth=8, n_estimators=100,
                        colsample_bytree=0.9, alpha=1,
                        objective='multi:softmax')
```

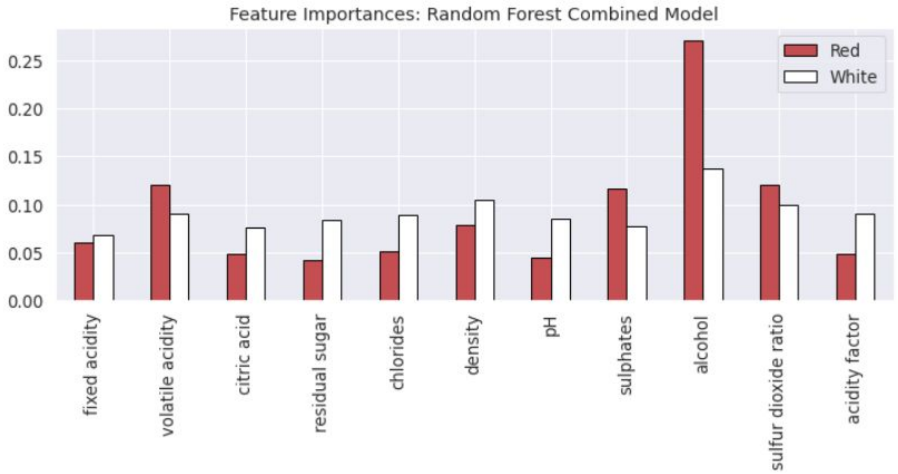
Model Results: RandomForest

	Single	Red	White	Combined	Combined+Single as white
Train	0.984	0.483	0.914	0.821	0.885
Test	0.438	0.317	0.461	0.429	0.44
Mean CV	0.45	0.299	0.438	0.416	0.431

```
model_r = RandomForestClassifier (n_estimators=120,
                    random_state=0, class_weight={3: 1, 4: 3, 5: 6, 6: 7, 7: 9, 8: 1},
                    max_depth = 8)
model_w = RandomForestClassifier(random_state=0, max_depth = 15, n_estimators=150,
                              class_weight={3:2, 4:2, 5:7, 6:6, 7:9, 8:4, 9:1})
```

Feature Importance

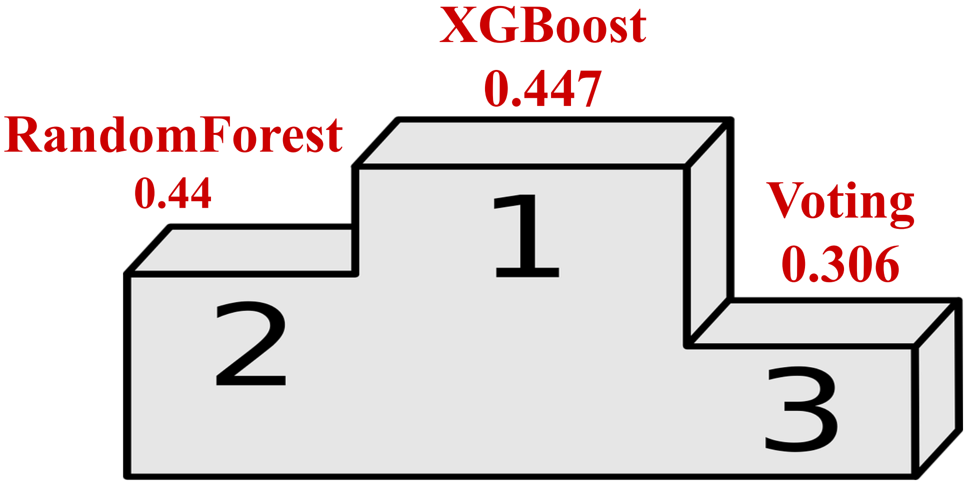




- Alcohol is dominant in both models
- red and wine have different importance
- Out engineered features proved to be valuable!

Conclusions

- Combined data failed to improve model scores
- Nevertheless, splitting the data into 2 populations was worth the effort, except for in VotingClassifier
- Applying grid search on each population and the joined data gave added value



Further exploring

1. Oversampling \ undersampling should be executed on the split model
2. Combining 2 different models (instead of one model with different hyper-parameters)
3. More time to GridSearch and improve the scores ;-)
4. Binning the quality target into good, medium and bad
5. Predicting wine type

Thank You