Kaggle Playground Season 3 Episode 5 - Wine Quality Data Competition

Part 1 - Data Exploration, Preparation and Model Spot-Checking

The Kaggle Playground competitions provide monthly opportunities to practice skills on realistic, real world type data sets.

This is my first try. The challenge is to predict wine quality ratings (between 3 and 8), based on wine chemistry data.

Prepare Problem

In [1]:

```
#load packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns

#check what packages are imported
import pkg_resources
import types
def get_imports():
    for name, val in globals().items():
        if isinstance(val, types.ModuleType):
            # Split ensures you get root package,
            # not just imported function
            name = val.__name__.split(".")[0]

        elif isinstance(val, type):
            name = val.__module__.split(".")[0]

        # Some packages are weird and have different
        # imported names vs. system names
        if name == "PIL":
            name = "Pillow"
        elif name == "sklearn":
            name = "scikit-learn"

        yield name
imports = list(set(get_imports()))

requirements = []
for m in pkg_resources.working_set:
    if m.project_name in imports and m.project_name!="pip":
        requirements.append((m.project_name, m.version))

for r in requirements:
    print("{}=={}".format(*r))
```

```
matplotlib==3.5.3
numpy==1.21.6
pandas==1.3.5
seaborn==0.12.2
```

Import Data

In [2]:

```
#load S3e5 Playground data

path = "/kaggle/input"

train = pd.read_csv(f"{path}/playground-series-s3e5/train.csv" )
train = pd.DataFrame(train)
print("Train data:\n", train.shape)
peek = train.head(5)
print(peek, "\n")
print(train.dtypes)

test = pd.read_csv(f"{path}/playground-series-s3e5/test.csv" )
test = pd.DataFrame(test)
print("Test data:\n", test.shape)
peek = test.head(5)
print(peek)
print(test.dtypes)
```

```
Train data:
 (2056, 13)
   Id  fixed acidity  volatile acidity  citric acid  residual sugar
\
0   0            8.0              0.50         0.39             2.2
1   1            9.3              0.30         0.73             2.3
2   2            7.1              0.51         0.03             2.1
3   3            8.1              0.87         0.22             2.6
4   4            8.5              0.36         0.30             2.3

   chlorides  free sulfur dioxide  total sulfur dioxide  density
pH  \
0      0.073                 30.0                  39.0  0.99572
3.33
1      0.092                 30.0                  67.0  0.99854
3.32
2      0.059                  3.0                  12.0  0.99660
3.52
3      0.084                 11.0                  65.0  0.99730
3.20
4      0.079                 10.0                  45.0  0.99444
3.20

   sulphates  alcohol  quality
0       0.77     12.1        6
1       0.67     12.8        6
2       0.73     11.3        7
3       0.53      9.8        5
4       1.36      9.5        6

Id                      int64
fixed acidity         float64
volatile acidity      float64
citric acid           float64
residual sugar        float64
chlorides             float64
free sulfur dioxide   float64
total sulfur dioxide  float64
density               float64
pH                    float64
sulphates             float64
alcohol               float64
quality                 int64
```

```
dtype: object
Test data:
 (1372, 12)
     Id  fixed acidity  volatile acidity  citric acid  residual sug
ar  \
0  2056            7.2             0.510         0.01
2.0
1  2057            7.2             0.755         0.15
2.0
2  2058            8.4             0.460         0.40
2.0
3  2059            8.0             0.470         0.40
1.8
4  2060            6.5             0.340         0.32
2.1

   chlorides  free sulfur dioxide  total sulfur dioxide  density
pH  \
0      0.077                 31.0                  54.0  0.99748
3.39
1      0.102                 14.0                  35.0  0.99586
3.33
2      0.065                 21.0                  50.0  0.99774
3.08
3      0.056                 14.0                  25.0  0.99480
3.30
4      0.044                  8.0                  94.0  0.99356
3.23

   sulphates  alcohol
0       0.59      9.8
1       0.68     10.0
2       0.65      9.5
3       0.65     11.7
4       0.48     12.8
Id                      int64
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
```

```
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
dtype: object
```

Summarise Data

In [3]:

```
#Data Info
train.info()
test.info()

#Data Description
pd.set_option('display.width', 100)
pd.set_option('precision', 3)
description = train.describe(include='all')
print("Data Description:\n", description, '\n')

#Data Imbalance/Class Distribution
class_counts = train.groupby('quality').size()
print("Class Distribution:\n", class_counts, '\n')

#Skew of Univariate Distributions
skew = train.skew()
print("Attribute skewness:\n", skew, '\n') #right(positive) or left(negative)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2056 entries, 0 to 2055
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Id                    2056 non-null   int64  
 1   fixed acidity         2056 non-null   float64
 2   volatile acidity      2056 non-null   float64
 3   citric acid           2056 non-null   float64
 4   residual sugar        2056 non-null   float64
 5   chlorides             2056 non-null   float64
 6   free sulfur dioxide   2056 non-null   float64
 7   total sulfur dioxide  2056 non-null   float64
 8   density               2056 non-null   float64
 9   pH                    2056 non-null   float64
 10  sulphates             2056 non-null   float64
 11  alcohol               2056 non-null   float64
 12  quality               2056 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 208.9 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Id                    1372 non-null   int64  
 1   fixed acidity         1372 non-null   float64
 2   volatile acidity      1372 non-null   float64
 3   citric acid           1372 non-null   float64
 4   residual sugar        1372 non-null   float64
 5   chlorides             1372 non-null   float64
 6   free sulfur dioxide   1372 non-null   float64
 7   total sulfur dioxide  1372 non-null   float64
 8   density               1372 non-null   float64
 9   pH                    1372 non-null   float64
 10  sulphates             1372 non-null   float64
 11  alcohol               1372 non-null   float64
dtypes: float64(11), int64(1)
memory usage: 128.8 KB
Data Description:
             Id  fixed acidity  volatile acidity  citric acid  resi
dual sugar  chlorides  \
count  2056.00       2056.000          2056.000     2056.000
```

```
2056.000   2056.000
mean   1027.50          8.365             0.528        0.265
2.399      0.082
std     593.66          1.705             0.173        0.188
0.859      0.024
min       0.00          5.000             0.180        0.000
1.200      0.012
25%     513.75          7.200             0.390        0.090
1.900      0.071
50%    1027.50          7.950             0.520        0.250
2.200      0.079
75%    1541.25          9.200             0.640        0.420
2.600      0.090
max    2055.00         15.900             1.580        0.760
14.000      0.414

       free sulfur dioxide  total sulfur dioxide   density        p
H  sulphates   alcohol  \
count             2056.000             2056.000  2056.000  2056.00
0   2056.000  2056.000
mean                16.956               49.237     0.997     3.31
1      0.641    10.415
std                 10.010               32.961     0.002     0.14
2      0.138     1.029
min                  1.000                7.000     0.990     2.74
0      0.390     8.700
25%                  8.000               22.000     0.996     3.20
0      0.550     9.500
50%                 16.000               44.000     0.997     3.31
0      0.610    10.100
75%                 24.000               65.000     0.998     3.39
0      0.720    11.000
max                 68.000              289.000     1.004     3.78
0      1.950    14.000

        quality
count  2056.000
mean      5.721
std       0.853
min       3.000
25%       5.000
50%       6.000
75%       6.000
```

```
max       8.000

Class Distribution:
 quality
3     12
4     55
5    839
6    778
7    333
8     39
dtype: int64

Attribute skewness:
 Id                     0.000
fixed acidity          0.960
volatile acidity       0.668
citric acid            0.247
residual sugar         3.757
chlorides              5.263
free sulfur dioxide    0.681
total sulfur dioxide   1.268
density                0.203
pH                     0.217
sulphates              1.803
alcohol                0.787
quality                0.266
dtype: float64
```

Notes

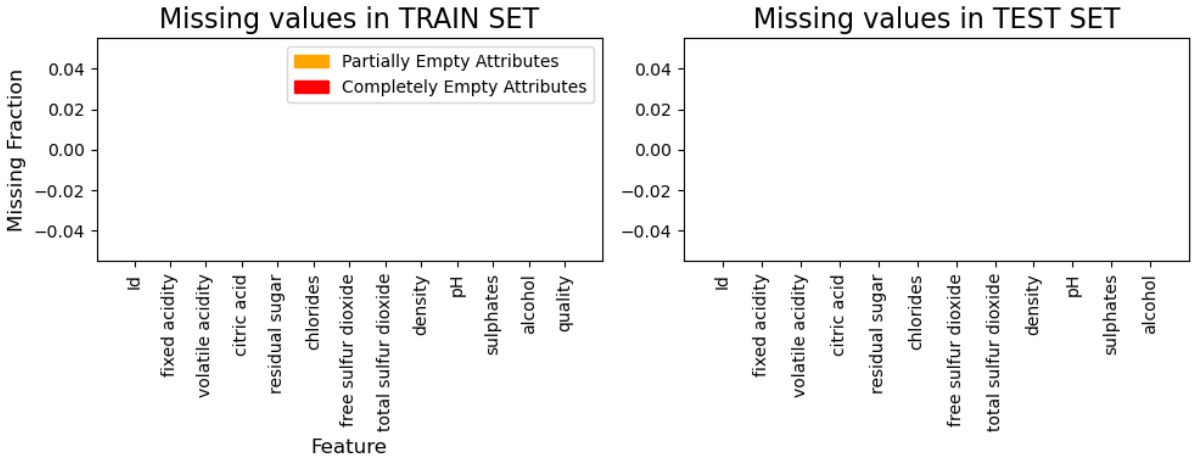
- The data is very imbalanced between the various quality groupings. Assume the categories can be 1 to 10? Only 3 to 8 is represented and most are in 5, 6, and 7.

Test for missing values and duplicates

None found

In [4]:

```
train_missing_ratios = train.isna().sum() / len(train)
test_missing_ratios = test.isna().sum() / len(test)
train_missing_ratios.head(20)

plt.figure(figsize=(10, 4))

# function to add value labels
def addlabels(x,y):
    for i in range(len(x)):
        if y[i]>0:
            plt.text(i,y[i],round(y[i], ndigits = 2))

plt.subplot(1, 2, 1)
plt.bar(train_missing_ratios.index,
        train_missing_ratios.values,
        color=['red' if ratio == 1 else 'orange' for ratio in train_missing_rati
os.values])
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Missing Fraction', fontsize=12)
plt.title('Missing values in TRAIN SET', fontsize=16)
plt.xticks(rotation=90)
plt.legend(handles=[mpatches.Patch(color='orange'),
                    mpatches.Patch(color='red')],
           labels=['Partially Empty Attributes', 'Completely Empty Attributes'])
addlabels(train_missing_ratios.index,
        train_missing_ratios.values)

plt.subplot(1, 2, 2)
plt.bar(test_missing_ratios.index,
        test_missing_ratios.values,
        color=['red' if ratio == 1 else 'orange' for ratio in test_missing_ratio
s.values])
plt.title('Missing values in TEST SET', fontsize=16)
plt.xticks(rotation=90)
addlabels(train_missing_ratios.index,
        train_missing_ratios.values)

plt.tight_layout()
plt.show()

train[train.duplicated()]
test[test.duplicated()]
```



Out[4]:

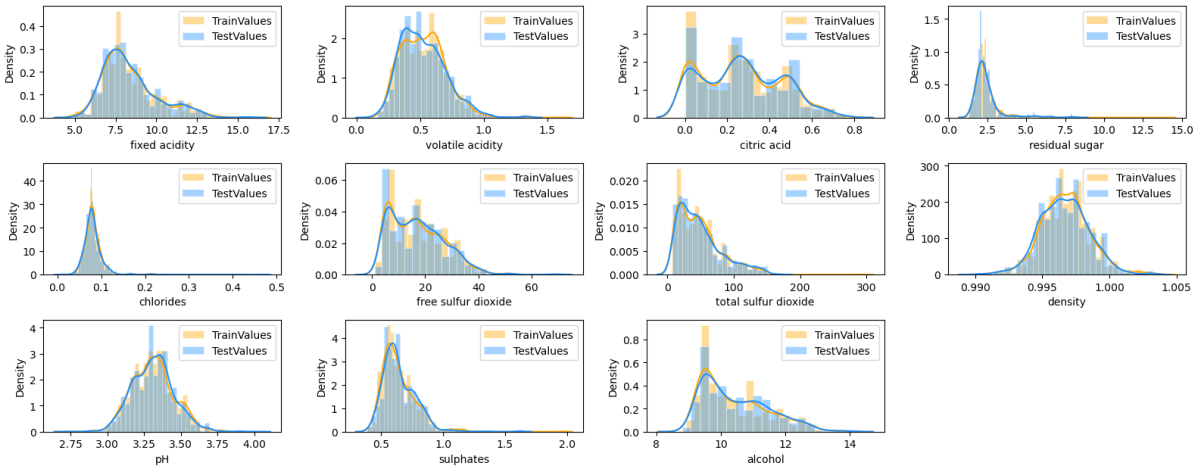
	Id	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate

Data Visualisation

Univariate Plots

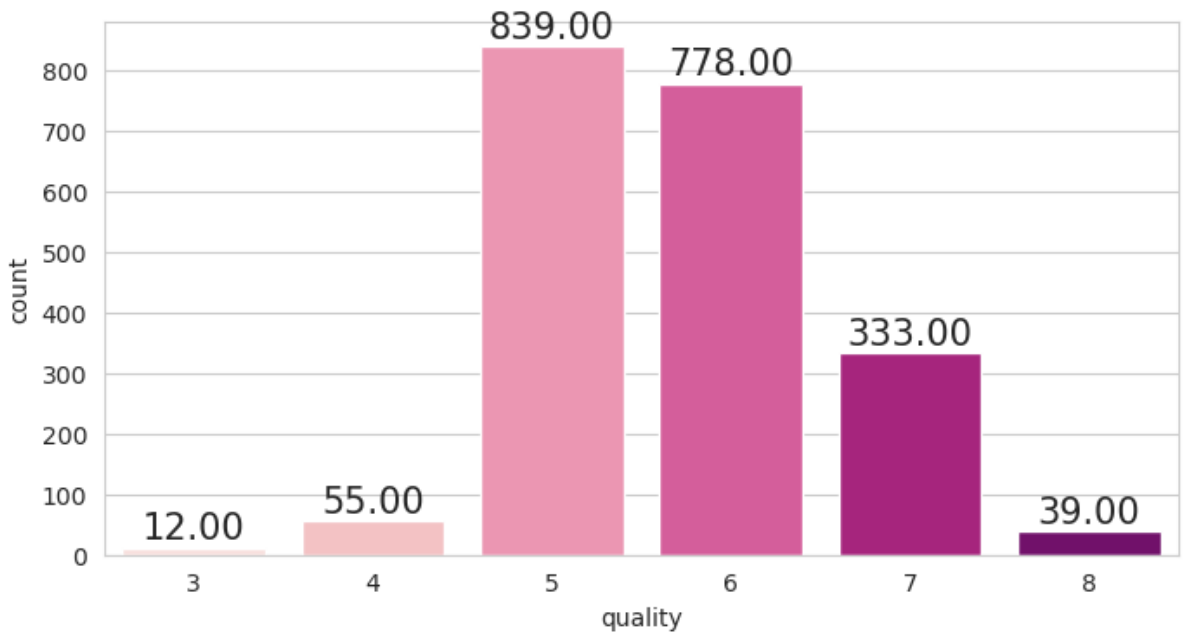
In [5]:

```
#see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
plt.figure(figsize=(16, 8))
for i, column in enumerate(train.columns[1:12], 1):
    plt.subplot(4,4,i)
    #plt.subplots(figsize=(7,6), dpi=100)
    sns.histplot( train[column], color="orange", kde=True, stat="density", kde_kws=dict(cut=3), alpha=.4, edgecolor=(1, 1, 1, .4), label="TrainValues")
    sns.histplot( test[column], color="dodgerblue", kde=True, stat="density", kde_kws=dict(cut=3), alpha=.4, edgecolor=(1, 1, 1, .4), label="TestValues")

    plt.legend()
    plt.tight_layout()
```



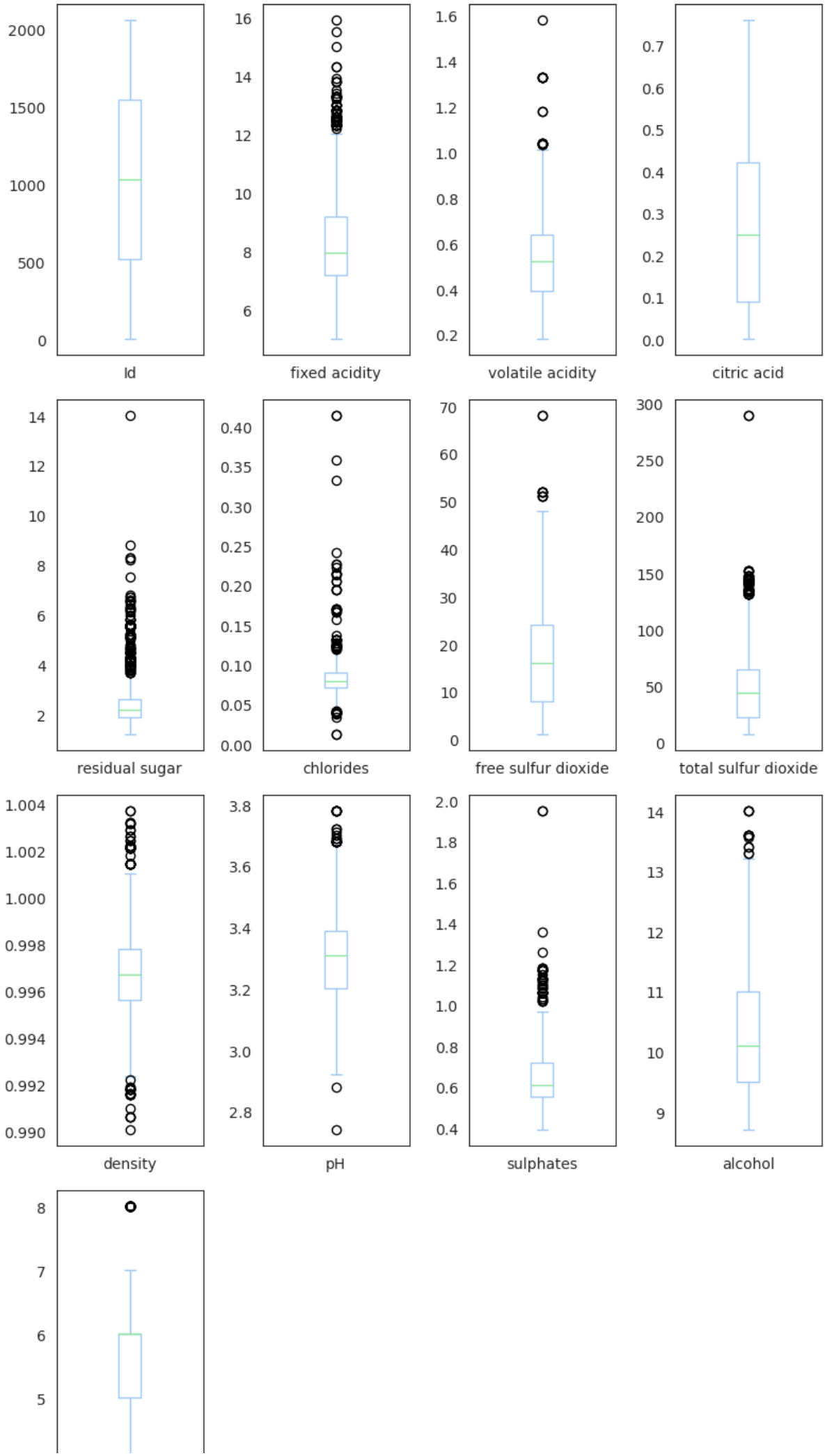
In [6]:

```
#show the balance of the classes
outcome = "quality"

plt.figure(figsize=(8, 4))
sns.set_style("whitegrid")

#https://www.codecademy.com/article/seaborn-design-ii
sns.set_palette("RdPu") #use SNS defauls (Deep, Muted, Bright, Pastel, Dark, Colorblind) or colorbrewer palettes

ax_train=sns.countplot(x=train[outcome], data=train);
for p in ax_train.patches:
    ax_train.annotate(format(p.get_height(), '.2f'), (p.get_x()+p.get_width()/2, p.get_height()), ha='center', va='center', size=15, xytext=(0, 8), textcoords='offset points')
```



In [7]:

```
#box and whisker
sns.set_style("white")
sns.set_palette("pastel")
train.plot(kind='box', subplots=True, layout=(4, 4), sharex=False, sharey=False,
figsize=(8,15))
plt.tight_layout()
plt.show()
```


Id
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol

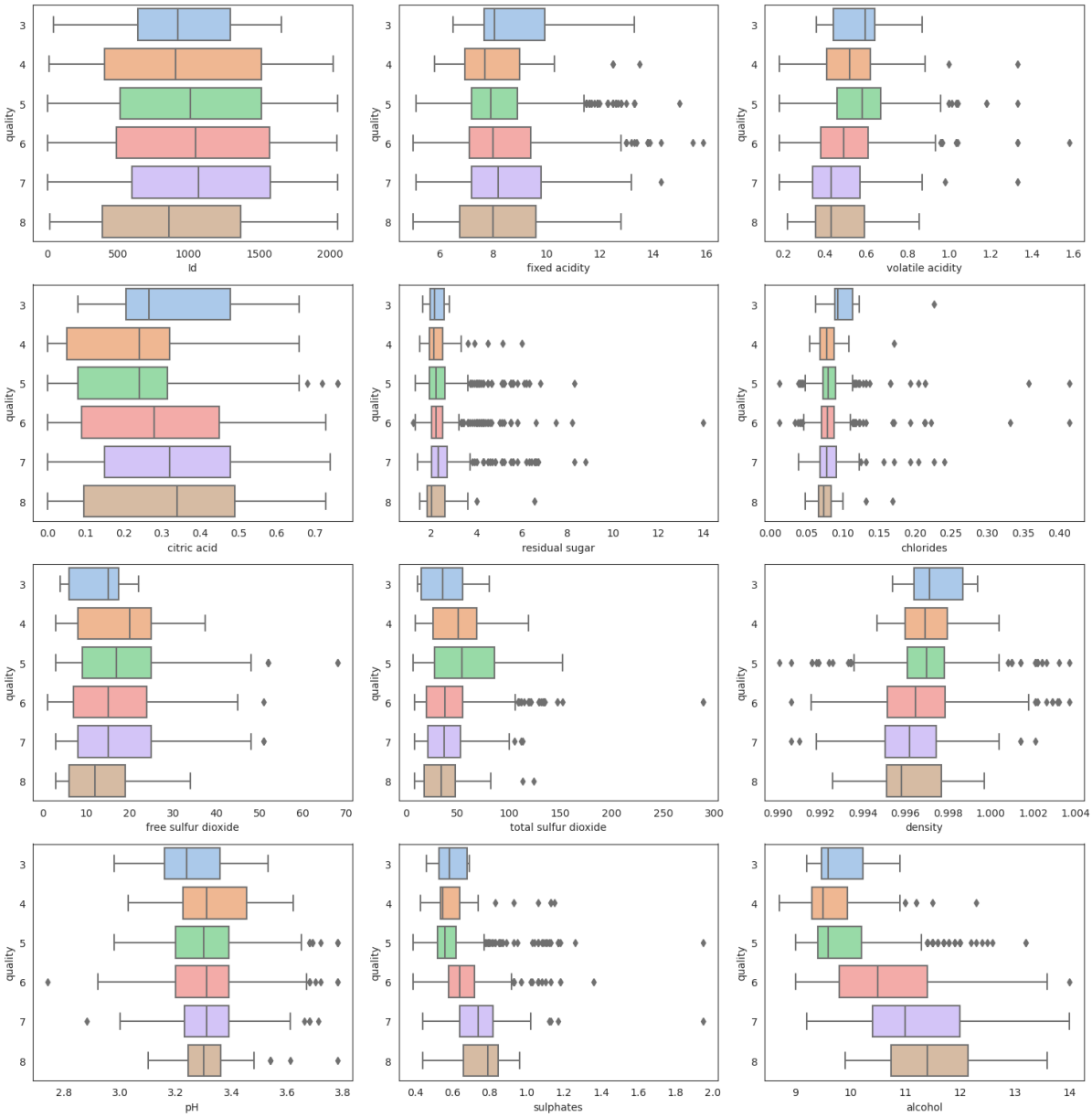
- Lots of outliers for many of the variables

Box Plot Grouped by Target Variable

In [8]:

```
def plot_features_by_target(df, num_features):
    """Display all columns in df except TARGET group by TARGET.
    """
    saved_type = df[TARGET].dtype
    df[TARGET] = df[TARGET].astype('category')
    columns = [c for c in df.columns if c != TARGET]
    ncols = 3
    nrows = np.ceil(len(columns)/ncols).astype(int)
    fig, axs = plt.subplots(ncols=ncols, nrows=nrows, figsize=(15,nrows*4))
    for c, ax in zip(columns, axs.flatten()):
        if c in num_features:
            sns.boxplot(data=df, x=c, y=TARGET, ax=ax)
        else:
            sns.countplot(data=df, x=c, hue=TARGET, ax=ax)
    fig.suptitle('Distribution of variables grouped by the target variable', fon
tsize=20)
    plt.tight_layout(rect=[0, 0, 1, 0.98])
    df[TARGET] = df[TARGET].astype(saved_type)


TARGET = "quality"
num_features = train.columns
plot_features_by_target(train, num_features)
```

Distribution of variables grouped by the target variable



We note:

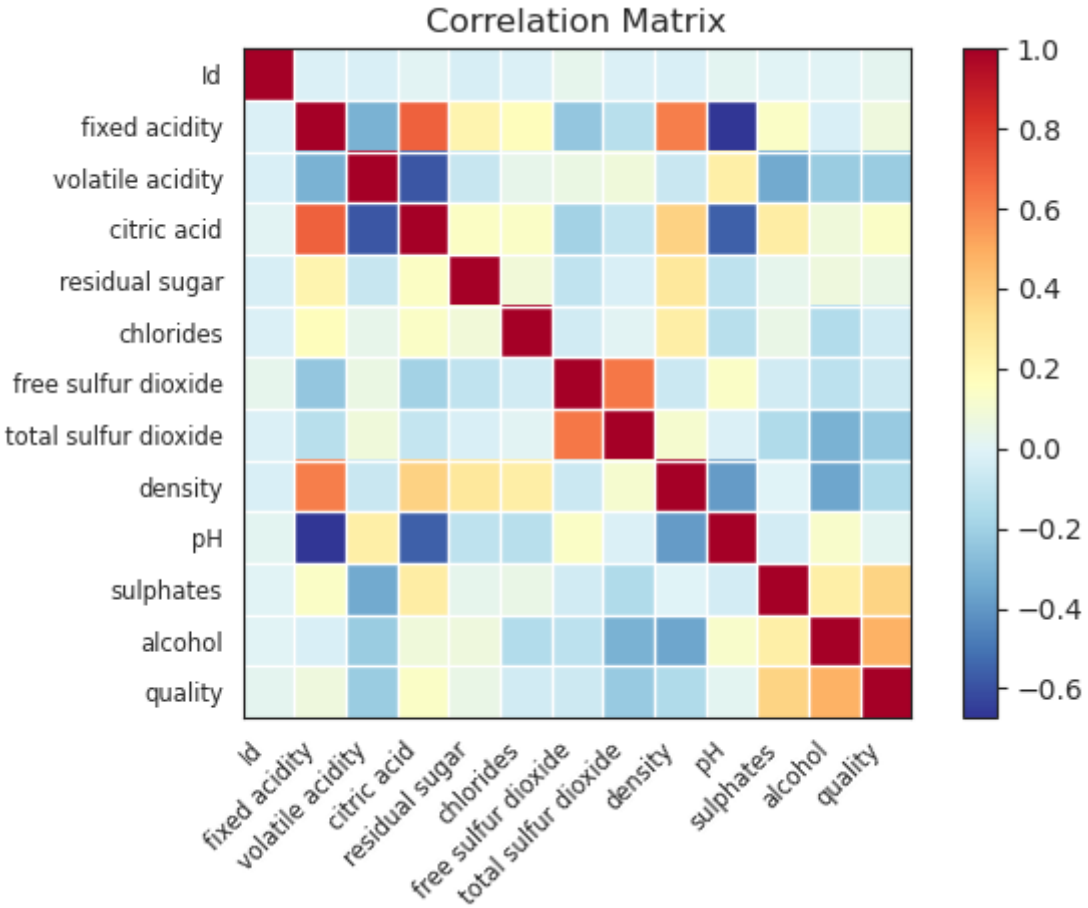
- A positive correlation between alcohol and quality.
- A positive correlation betwee sulphates and quality.
- With both of these, its strongest between class 5,6 and 7.
- Chloride level could be useful for picking out class 3.

Multivariate Plots

Correlation Plot

In [9]:

```
import statsmodels.graphics.api as smg

corr_matrix = np.corrcoef(train.T)
names = list(train.columns)
smg.plot_corr(corr_matrix, xnames=names)
plt.show()
```



We can see a positive correlation between quality and:

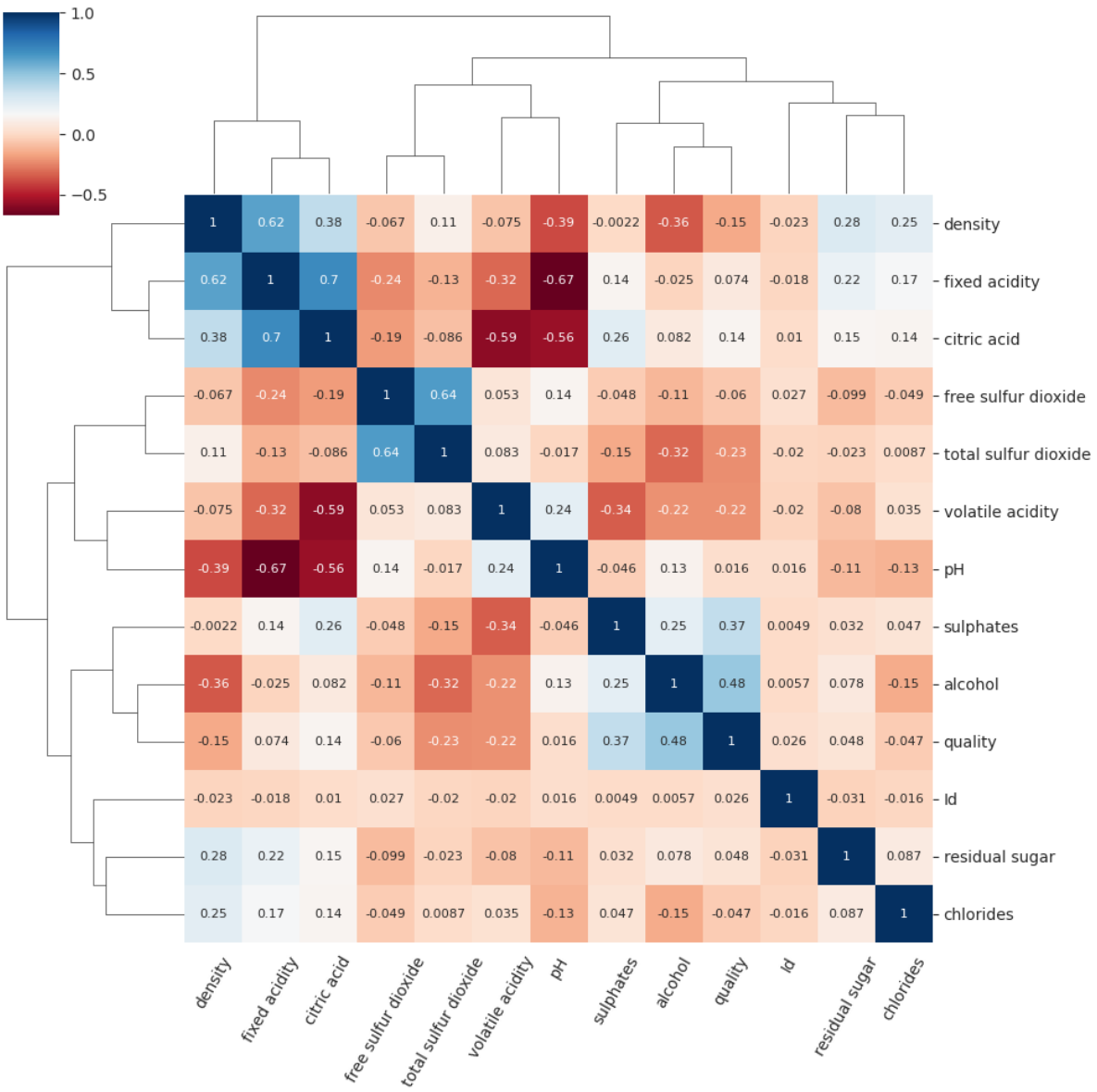
- alcohol
- sulphates
- citric acid

negative correlation to:

- volatile acidity
- total sulfur density

In [10]:

```
g = sns.clustermap(train.corr(),
                   method = 'complete',
                   cmap   = 'RdBu',
                   annot  = True,
                   annot_kws = {'size': 8})
plt.setp(g.ax_heatmap.get_xticklabels(), rotation=60);
```



Prepare Data

Split data into Train and Validation Sets

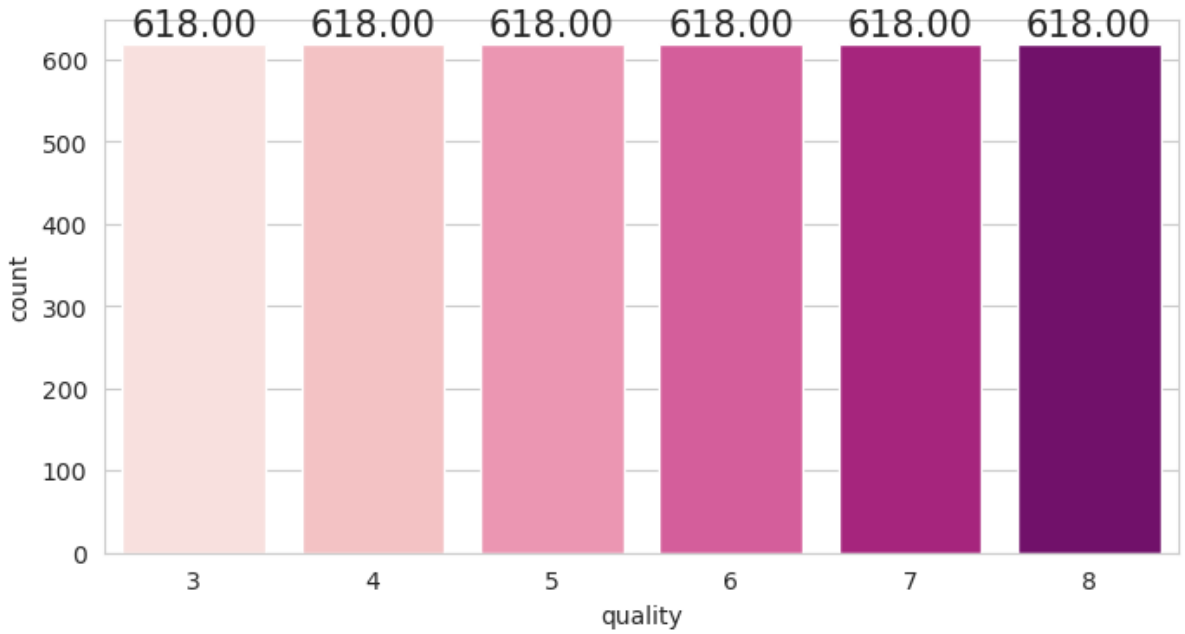
In [11]:

```
# Evaluate using a train and a test set
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

#numerical predictor (NP) and response locations
NP_first_loc = 'fixed acidity' #ignore col 0 which is the ID
NP_last_loc = 'alcohol'
outcome_loc = 'quality'

NP_first_iloc = train.columns.get_loc(NP_first_loc) #ignore col 0 which is the ID
#print("NP_first_iloc:",NP_first_iloc, "\n" )
NP_last_iloc = train.columns.get_loc(NP_last_loc)
#print("NP_last_iloc:",NP_last_iloc, "\n" )
outcome_iloc = train.columns.get_loc(outcome_loc)

#Original Data
X = train.loc[:,NP_first_loc:NP_last_loc]
#print("\nX shape: ", X.shape)
#print("First two rows of X:\n", X.iloc[0:2, :], "\n")
Y = train.loc[:,outcome_loc]
#print("\n\nFist 10 rows of Y: \n", Y.iloc[0:10])

#Split Data
test_size = 0.25
seed = 100 #re-evaluate with different seeds to estimate variance?
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=test_size,rand
om_state=seed)
```

Data Balancing

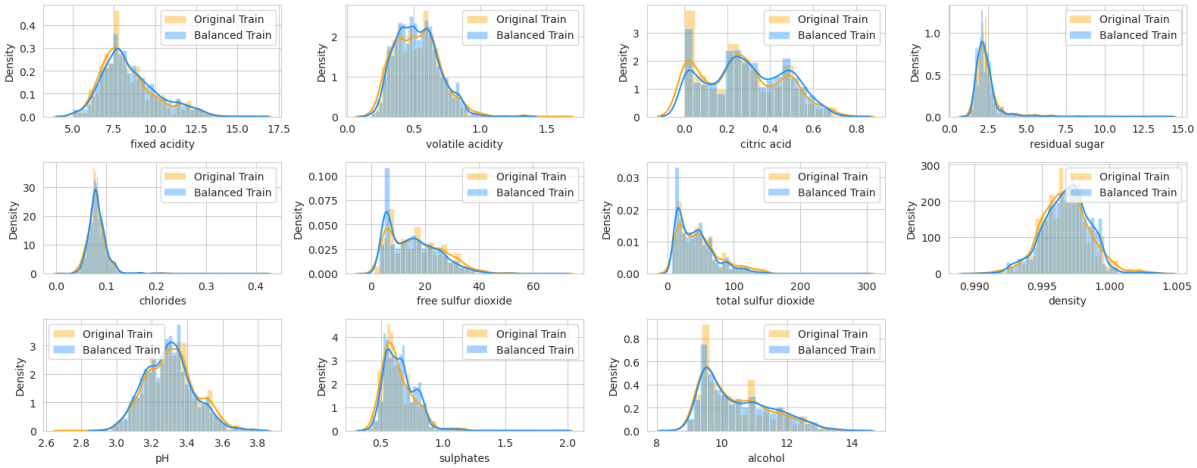
In [12]:

```
from imblearn.over_sampling import SMOTE

outcome = 'quality'
#train = pd.DataFrame(train)

#Only use the training set for balancing
#don't use the full x set and split the balanced data later, because we will be va
ilidating on the Y set split from the orignial data and dont want to leek data fro
m the val set into the balanced training set

X_train_b, Y_train_b = SMOTE().fit_resample(X_train, Y_train)
train_balanced = X_train_b.reset_index(drop=True).join(Y_train_b)

#replot the class distribution
plt.figure(figsize=(8, 4))
sns.set_style("whitegrid")

#https://www.codecademy.com/article/seaborn-design-ii
sns.set_palette("RdPu") #use SNS defauls (Deep, Muted, Bright, Pastel, Dark, Colo
rblind) or colorbrewer palettes

ax_train=sns.countplot(x=train_balanced[outcome], data=train_balanced);
for p in ax_train.patches:
    ax_train.annotate(format(p.get_height(), '.2f'), (p.get_x()+p.get_width()/2,
p.get_height()), ha='center', va='center', size=15, xytext=(0, 8), textcoords='o
ffset points')
```


618.00
618.00
618.00
618.00
618.00
618.00
600
500
400
300
200
100
0
count
3
4
5
6
7
8
quality

In [13]:

```
#redo the density/histograms to compare original data to resampled

plt.figure(figsize=(16, 8))
for i, column in enumerate(train.columns[1:12], 1):
    plt.subplot(4,4,i)
    #plt.subplots(figsize=(7,6), dpi=100)
    sns.histplot( train[column], color="orange", kde=True, stat="density", kde_kws=dict(cut=3), alpha=.4, edgecolor=(1, 1, 1, .4), label="Original Train")
    sns.histplot( train_balanced[column], color="dodgerblue", kde=True, stat="density", kde_kws=dict(cut=3), alpha=.4, edgecolor=(1, 1, 1, .4), label="Balanced Train")

    plt.legend()
    plt.tight_layout()
```



We note that the distributions look a bit differen for the balanced data. Let's visualise by group.

In [14]:

```
#show only one quality class at a time
qual = 8

train_sub_qual = train.loc[(train["quality"] == qual)]
# peek = train_sub_qual.head(5)
#print(peek)
train_balanced_sub_qual = train.loc[(train_balanced["quality"] == qual)]

plt.figure(figsize=(16, 8))
for i, column in enumerate(train_sub_qual.columns[1:12], 1):
    plt.subplot(4,4,i)
    sns.histplot( train_balanced_sub_qual[column], color="dodgerblue", kde=True, stat="density", kde_kws=dict(cut=3),line_kws=dict(linewidth=5), alpha=.4, edgecolor=(1, 1, 1, .4), label="Balanced Train")
    sns.histplot( train_sub_qual[column], color="orange", kde=True, stat="density", kde_kws=dict(cut=3),line_kws=dict(linewidth=2.5), alpha=.4, edgecolor=(1, 1, 1, .4), label="Original Train")


    plotTitle = "Wine Quality Rating: " + str(qual)
    plt.title(plotTitle)
    plt.legend()
    plt.tight_layout()
```

It can be shown that the SMOTE oversampling perfrectly matched the original distributions by outcome class. Hence the overal distribution looks a bit different since the counts per class are adjusted.

Standardise data

In [15]:

```
# Standardize data (0 mean, 1 stdev)

from sklearn.preprocessing import StandardScaler

#standardize Original data - all data including validation set
scaler = StandardScaler().fit(X)
X_std = scaler.transform(X)
X_std = pd.DataFrame(X_std, index=X.index, columns=X.columns)

# summarize transformed data
np.set_printoptions(precision=3)
print("\nRescaled original train data peek:\n",X_std.iloc[0:5,:])

#standardize Original data training and validation
scaler = StandardScaler().fit(X_train) #evaluate the set to figure out how to tra
nsform - use only training data
X_train_std = scaler.transform(X_train) #apply transformation
X_train_std = pd.DataFrame(X_train_std, index=X_train.index, columns=X_train.col
umns)
X_val_std = scaler.transform(X_val)
X_val_std = pd.DataFrame(X_val_std, index=X_val.index, columns=X_val.columns)
#standardize Balanced data -  traing and validation set
scaler = StandardScaler().fit(X_train_b) #evaluate the set to figure out how to t
ransform - use only training data
X_train_b_std = scaler.transform(X_train_b)  #apply transformation
X_train_b_std = pd.DataFrame(X_train_b_std, index=X_train_b.index, columns=X_tra
in_b.columns)
#X_val_b_std = scaler.transform(X_val_b) - no need to balance the X validation se
t. validate on the orignial or standardised X val data

# summarize transformed data
np.set_printoptions(precision=3)
print("\nRescaled balanced train data peek:\n", X_train_b_std.iloc[0:5,:])
```

```
Rescaled original train data peek:
    fixed acidity  volatile acidity  citric acid  residual sugar  c
hlorides  free sulfur dioxide  \
0         -0.214            -0.159        0.664          -0.232
-0.373                1.303
1          0.548            -1.315        2.470          -0.115
0.428                1.303
2         -0.742            -0.102       -1.249          -0.348
-0.963               -1.395
3         -0.156             1.978       -0.239           0.234
0.090               -0.595
4          0.079            -0.968        0.186          -0.115
-0.120               -0.695

   total sulfur dioxide  density     pH  sulphates  alcohol
0                -0.311   -0.563  0.137      0.933    1.638
1                 0.539    0.981  0.066      0.208    2.319
2                -1.130   -0.081  1.472      0.643    0.860
3                 0.478    0.302 -0.777     -0.807   -0.598
4                -0.129   -1.263 -0.777      5.211   -0.890

Rescaled balanced train data peek:
    fixed acidity  volatile acidity  citric acid  residual sugar  c
hlorides  free sulfur dioxide  \
0          0.654             0.583        1.090          -1.256
0.709               -1.122
1         -0.588             0.456       -1.539          -0.056
-0.710               1.486
2         -0.588            -0.301        0.214          -0.856
-0.304               1.051
3         -0.411            -2.069        0.324           1.809
-0.659               0.073
4         -0.648             1.593       -1.593          -0.056
-0.811               -0.035

   total sulfur dioxide  density     pH  sulphates  alcohol
0                 0.661    0.475 -0.702     -0.359   -1.025
1                 0.102   -0.925  0.399     -0.833    0.010
2                 0.242   -0.843  0.105      3.675   -0.460
3                 0.591    0.827  0.986     -0.596   -1.025
4                -0.038    0.920  1.572     -1.070   -0.743
```

Feature Selection

Add some features based on cluster analysis

Univariate Feature Selection

In [16]:

```
# Feature Selection with Univariate Statistical Tests
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from tabulate import tabulate

# choose which data to use - origninal? balanced? standardised?
X_s = X
Y_s = Y

# Univariate Feature Selection - feature extraction
test = SelectKBest(score_func=f_classif, k=4)
fit_UV = test.fit(X_s, Y_s)
np.set_printoptions(precision=3)
UVscores = fit_UV.scores_

# summarize scores
features = fit_UV.transform(X_s)

#get a list of the numberical column names
#colNameList = list(train.iloc[:,NP_first_iloc:NP_last_iloc+1].columns.values)#hav
e to add 1 for the slice to include the last index with iloc
colNameList = list(train.loc[:,NP_first_loc:NP_last_loc].columns.values)
#print(colNameList)

# summarize selected features
table = [colNameList,
         UVscores, ]

print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
3.72697	25.7363	11.1406	3.38504	3.27028	3.92142	33.0478	10.5045	0.975361	73.6244	134.458

The highest values shows the highest correlation with the outcome/target variable, according to the ANOVA F-value method (appropriate for numerical inputs and categorical data). The following variables come out tops, in order:

- alcohol
- sulphates
- total sulfur dioxide
- volatile acidity
- citric acid
- density

Interestingly, when using the oversampled (balanced) data, this changes to:

- alcohol
- sulphates
- chlorides
- total sulfur dioxide
- density
- free sulfur dioxide
- volatile acidity
- citric acid

The standardisation didn't make any difference.

Recursive Feature Elimination

We'll play around a bit to see the different between using regression and classification models. This problem can be treated as either, but I think regression will be a bit more appropriate, since the quality levels to mean something as they ascend in value. They are not just arbitrary bins.

In [17]:

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from tabulate import tabulate
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor


# choose which data to use - origninal? balanced? standardised?
X_s = X_std
Y_s = Y

#how many features to compare
topN = 1
print("Find %d best variables.\n " % topN)

# Recursive Feature Elimination - feature extraction
#model = LogisticRegression(solver='liblinear')
#model = DecisionTreeClassifier()
model = DecisionTreeRegressor()
rfe = RFE(estimator = model, n_features_to_select = topN)
fit_RFE = rfe.fit(X_s, Y_s)
print("Selected Features: %s" % fit_RFE.support_)
print("Feature Ranking: %s" % fit_RFE.ranking_)

# summarize selected features
RFEscores = fit_RFE.ranking_

#get a list of the numberical column names
#colNameList = list(train.iloc[:,NP_first_iloc:NP_last_iloc+1].columns.values)#hav
e to add 1 for the slice to include the last index with iloc
colNameList = list(train.loc[:,NP_first_loc:NP_last_loc].columns.values)
#print(colNameList)

table = [colNameList,
         RFEscores, ]

print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

Find 1 best variables.

Selected Features: [False False False False False False False False False False True]
Feature Ranking: [4 6 9 7 8 10 3 5 11 2 1]

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
4	6	9	7	8	10	3	5	11	2	1

For the original data, the following come out highest:

- sulphates
- density
- pH
- alchohol
- volatile acidity
- citric acid

For the balanced data:

- chlorides
- sulphates
- citric acid
- volatile acidity
- desnity
- pH

This changed significantly when the standardised data was used, as follows:

For the original data, the following come out highest:

- alchohol
- sulphates
- total sulfur dioxide
- free sulfur dioxide
- citric acid
- volatile acidity

For the balanced data:

- alcohol
- chlorides
- sulphates
- total sulfur dioxide
- free sulfur dioxide
- citric acid

These results also changed drastically depending on the chosen classification or regression model.

Feature Importance

Use a random forest model to get some shapley estimators

In [18]:

```
# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestRegressor


# choose which data to use - origninal? balanced? standardised?
X_s = X_std
Y_s = Y

# feature extraction
#model = ExtraTreesClassifier(n_estimators=100)
model = RandomForestRegressor(max_depth=6, random_state=0, n_estimators=10)
model.fit(X_s, Y_s)
importances = pd.DataFrame(data={
    'Attribute': X_s.columns,
    'Importance': model.feature_importances_
})
importances = importances.sort_values(by='Importance', ascending=False)

print("Feature Importance Results using original data:\n")
print(importances)

model.fit(X_s, Y_s)

importances = pd.DataFrame(data={
    'Attribute': X_s.columns,
    'Importance': model.feature_importances_
})
importances = importances.sort_values(by='Importance', ascending=False)
print("Feature Importance Results using balanced data:\n")
print(importances)
```

```
Feature Importance Results using original data:

               Attribute  Importance
10               alcohol       0.466
9              sulphates       0.296
0          fixed acidity       0.043
7                density       0.036
4              chlorides       0.034
8                     pH       0.030
6   total sulfur dioxide       0.024
1       volatile acidity       0.020
3         residual sugar       0.019
2            citric acid       0.019
5    free sulfur dioxide       0.014
Feature Importance Results using balanced data:

               Attribute  Importance
10               alcohol       0.466
9              sulphates       0.296
0          fixed acidity       0.043
7                density       0.036
4              chlorides       0.034
8                     pH       0.030
6   total sulfur dioxide       0.024
1       volatile acidity       0.020
3         residual sugar       0.019
2            citric acid       0.019
5    free sulfur dioxide       0.014
```

These results correspond fairly well. No big differnce when using the standardised data.

Comparing all the results it seems to say that the following variables are most important:

- alcohol
- sulphates
- total sulfur dioxide
- chlorides
- free sulfur dioxide

These ones not as much:

- residual sugar
- fixed acidity
- pH

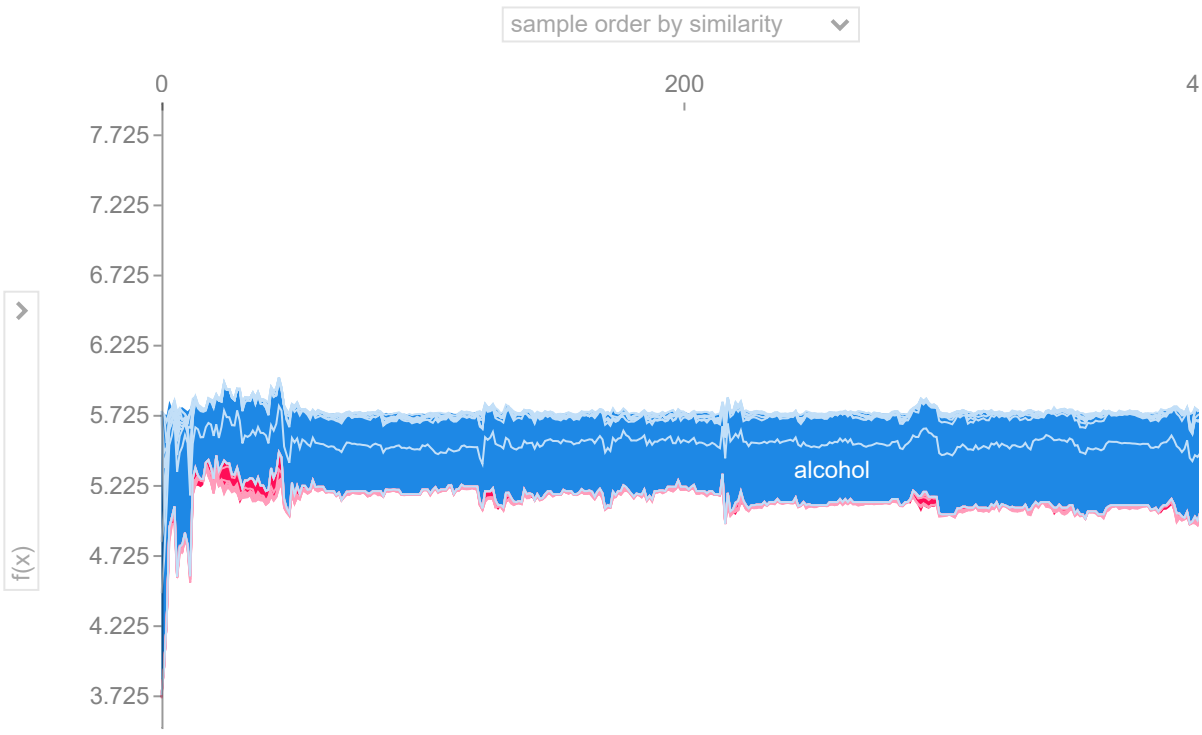
Shapley Values

In [19]:

```
# import shap library
import shap
colNameList = list(train.loc[:,NP_first_loc:NP_last_loc].columns.values)
print(colNameList)

# explain the model's predictions using SHAP
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_s)

# visualize the training set predictions
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values, X_s)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'de
nsity', 'pH', 'sulphates', 'alcohol']
```

Out[19]:

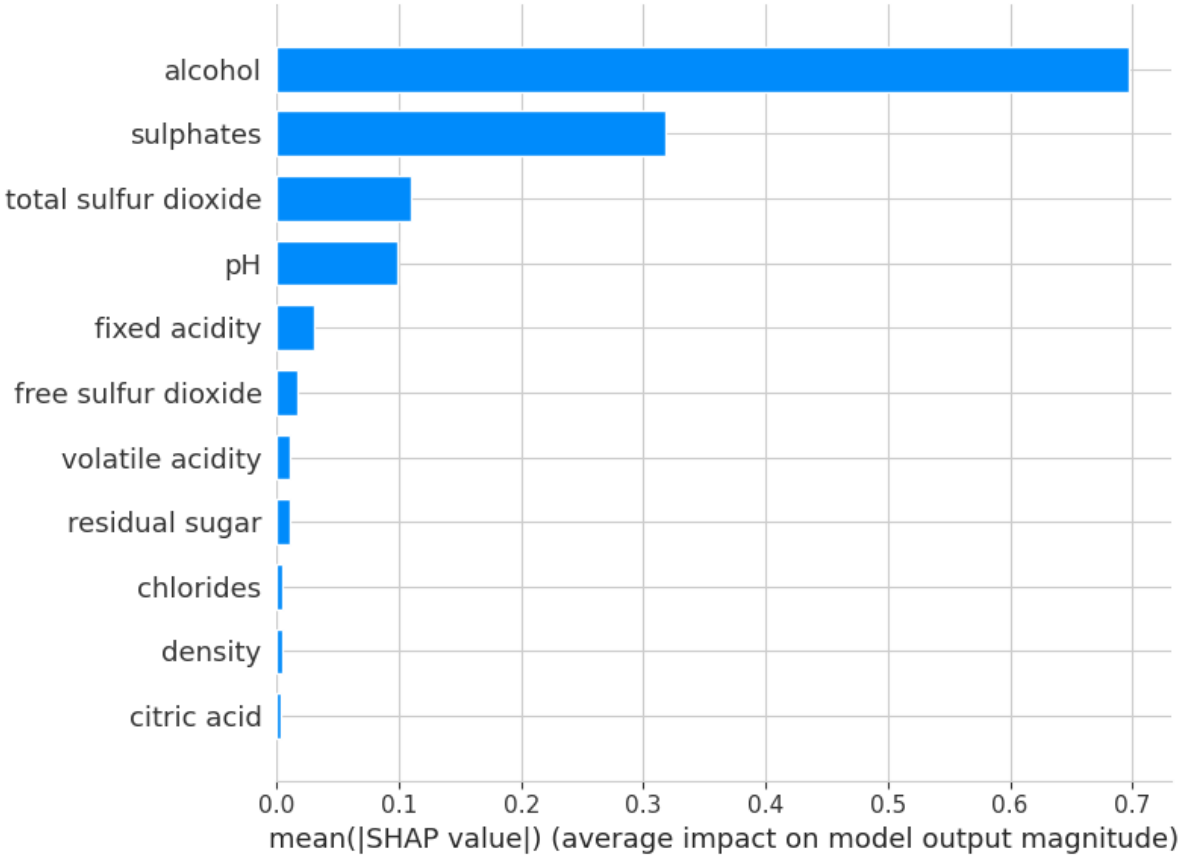


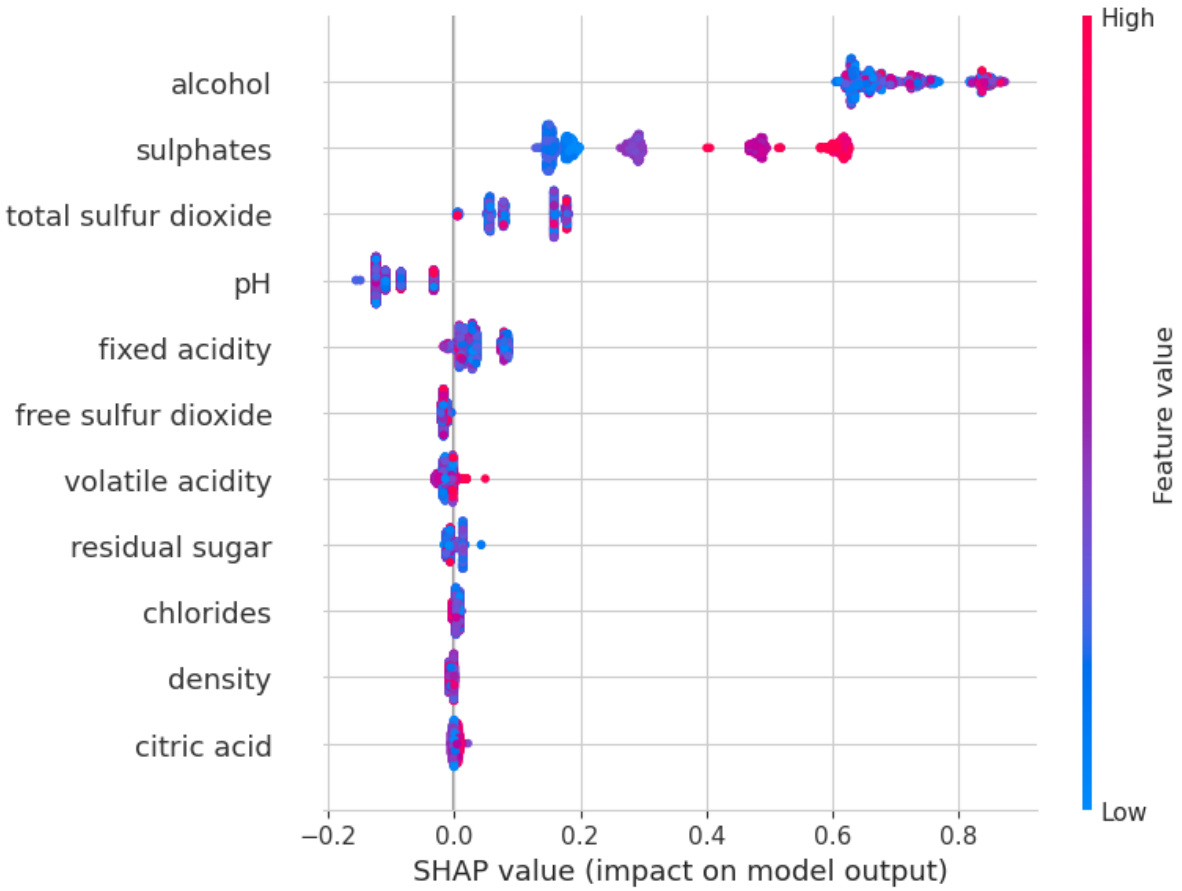
Features that features heavily are:

- Alcohol (feature 10)
- Chlorides (feature 4)
- Density (feature 7)
- Sulphates (feature 9)

In [20]:

```
shap_values = shap.TreeExplainer(model).shap_values(X_train)
shap.summary_plot(shap_values, X_train, plot_type="bar")
shap.summary_plot(shap_values, X_train)
```


alcohol
sulphates
total sulfur dioxide
pH
fixed acidity
free sulfur dioxide
volatile acidity
residual sugar
chlorides
density
citric acid
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
mean(|SHAP value|) (average impact on model output magnitude)


alcohol
sulphates
total sulfur dioxide
pH
fixed acidity
free sulfur dioxide
volatile acidity
residual sugar
chlorides
density
citric acid
−0.2
0.0
0.2
0.4
0.6
0.8
SHAP value (impact on model output)
High
Feature value
Low

Algorithm Spot-Checking and Comparison

We'll do some initial spot checking to see which algorithms look most promising. We'll try some linear and non-linear algorithms. We'll use 10-fold cross validation.

Spotcheck Function Definition

In [21]:

```
def modelSpotCheck(X_s, Y_s, X_v, Y_v, dataDescription, models, scoring, ylim_mi
n = None, ylim_max = None):

    #lists defined outside of the loop to gather all results throughout iterations
    predictions = [[] for i in range(len(X_s))]
    absError = [[] for i in range(len(X_s))]
    modfits  = [[] for i in range(len(X_s))]

    #overall loop through the various data sets passed (orignial, standardised et
c.)
    for i in range(len(X_s)):

        #initialise and clear lists on each iteration
        #summary, results, names = []
        summary = []
        results = []
        names = []

        # evaluate each model in turn
        for name, model in models:
            kfold = KFold(n_splits=10, random_state=7, shuffle=True)
            cv_results = cross_val_score(model, X_s[i], Y_s[i], cv=kfold, scorin
g=scoring)
            results.append(cv_results)
            names.append(name)
            modSum = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
            summary.append(modSum)
            model.fit(X_s[i], Y_s[i])
            modfits[i].append(model)
            modelpred = model.predict(X_v[i])
            predictions[i].append(modelpred)
            Err = abs(Y_v[i] - np.round(modelpred))
            absError[i].append(Err)

        #Print summary of performance (mean and std dev according to metric chose
n)
        print("Results according to ", dataDescription[i], "data:\n", summary )

        # boxplot algorithm comparison
        ax = plt.subplot(1, len(X_s),(i+1))
        ax.set_title(dataDescription[i])
        if (not ylim_min == None and not ylim_max == None):
            ax.set_ylim(ylim_min, ylim_max)
```

```
        sns.boxplot(results, ax = ax)
        ax.set_xticklabels(names,rotation=90)
        #plt.show()

    return names, predictions, absError, modfits
```

Regression Approach

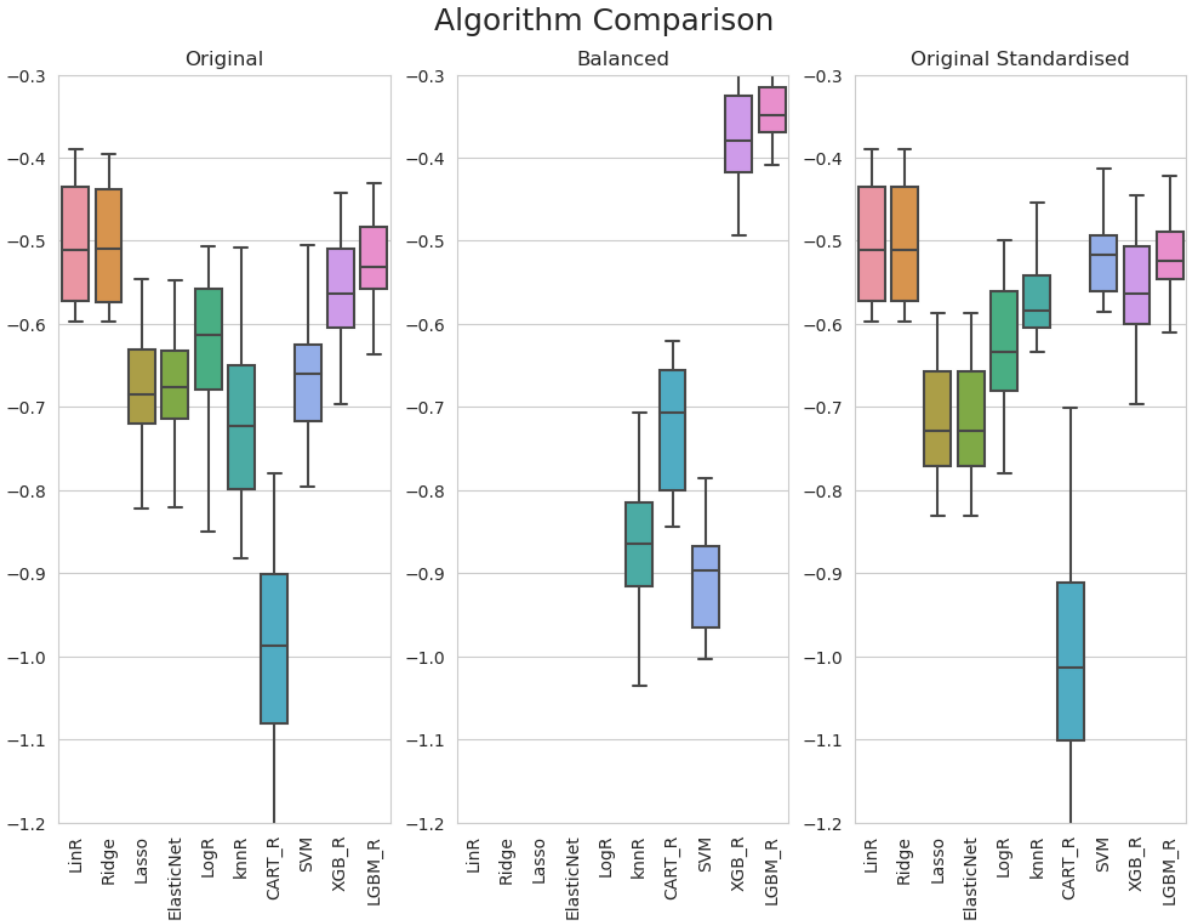
We'll treat this as a regression problem and use the following regresion metric: Mean Absolute Error

In [22]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from xgboost.sklearn import XGBRegressor
from lightgbm import LGBMRegressor

# select which data to use - origninal? balanced? standardised?
dataDescription_R = ["Original","Balanced", "Original Standardised"] #Ensure "St
andardised" is included for standardised data
X_s = [pd.DataFrame(X_train.copy()), pd.DataFrame(X_train_b.copy()), pd.DataFram
e(X_train_std.copy())]#predictors to train on
Y_s = [             Y_train.copy(),               Y_train_b.copy(),
Y_train.copy()] #outcome to train on
X_v = [pd.DataFrame(X_val.copy()), pd.DataFrame(X_val.copy()), pd.DataFrame(X_va
l_std.copy())] #predictors to validate on
Y_v = [             Y_val.copy(),               Y_val.copy(),               Y_va
l.copy()] #outcome to validate on

#define plot size
plt.figure(figsize=(12,8))
plt.suptitle('Algorithm Comparison',fontsize=18, y=0.95)

# prepare models
models_R = []
models_R.append(('LinR', LinearRegression()))
models_R.append(('Ridge', Ridge()))
models_R.append(('Lasso', Lasso()))
models_R.append(('ElasticNet', ElasticNet()))
models_R.append(('LogR', LogisticRegression(solver='liblinear')))
models_R.append(('knnR', KNeighborsRegressor()))
models_R.append(('CART_R', DecisionTreeRegressor()))
models_R.append(('SVM', SVR(gamma='auto')))
models_R.append(('XGB_R', XGBRegressor()))
models_R.append(('LGBM_R', LGBMRegressor()))
```

```
#Run the check
names_R, predictions_R, absError_R, modfits_R = modelSpotCheck(X_s, Y_s, X_v, Y_
v,dataDescription_R, models_R, scoring = 'neg_mean_squared_error', ylim_min = -
1.2, ylim_max= -0.3)
#scoring = 'r2', 'neg_mean_absolute_error', 'neg_mean_squared_error'
```

```
Results according to  Original data:
 ['LinR: -0.504966 (0.074338)', 'Ridge: -0.505329 (0.072869)', 'Las
so: -0.681115 (0.072868)', 'ElasticNet: -0.677071 (0.071292)', 'Log
R: -0.630364 (0.095082)', 'knnR: -0.718555 (0.109418)', 'CART_R: -
1.000746 (0.152563)', 'SVM: -0.660439 (0.089866)', 'XGB_R: -0.56271
5 (0.069656)', 'LGBM_R: -0.524735 (0.056411)']
Results according to  Balanced data:
 ['LinR: -1.424141 (0.099425)', 'Ridge: -1.437078 (0.104783)', 'Las
so: -2.575330 (0.105795)', 'ElasticNet: -2.109788 (0.101119)', 'Log
R: -2.116405 (0.219188)', 'knnR: -0.871220 (0.092796)', 'CART_R: -
0.724093 (0.081752)', 'SVM: -0.906942 (0.071673)', 'XGB_R: -0.37731
7 (0.057530)', 'LGBM_R: -0.348787 (0.037275)']
Results according to  Original Standardised data:
 ['LinR: -0.504966 (0.074338)', 'Ridge: -0.504953 (0.074334)', 'Las
so: -0.713355 (0.075453)', 'ElasticNet: -0.713355 (0.075453)', 'Log
R: -0.629736 (0.083267)', 'knnR: -0.569169 (0.050935)', 'CART_R: -
1.014390 (0.180570)', 'SVM: -0.516883 (0.051533)', 'XGB_R: -0.56166
0 (0.068969)', 'LGBM_R: -0.519431 (0.050289)']
```



Notes

From the graphs above, it seems like some of the models trained on the balanced data did really well. But we need to remember that the outcome variable was also oversampled and the results might not be as good when the model is tested on the validation set that was separated from the original data. We'll have a look.

We also notice that some of the models (KNN, SVM and CART to a lesser degree) did show improvement when modelled on standardised data.

In [23]:

```
#build confusion matrices for rounded regression values for specified models

from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

#models4cm = ["LinR", "Ridge", "LogR", "SVM", "LGBM_R"]
#k =          [0,     0,        0,      1,     1] #choose which data set to use
models4cm = ["LinR", "LGBM_R", "XGB_R", "LGBM_R", "Ridge"]
k =          [0,      0,        1,        1,        2] #choose which data set to
use


for i in range(len(models4cm)):
    df_confusion = 0
    dfCropped = 0
    N = names_R.index(models4cm[i])
    #if model was fit on standardised data then also use standardised validationio
n data
    if "Standardised" in dataDescription_R[k[i]]:
        modelpred = modfits_R[k[i]][N].predict(X_val_std)
    else:
        modelpred = modfits_R[k[i]][N].predict(X_val)
    #cm_pred = modelpred.copy()
    #cm_pred = np.round(modelpred.copy())\
    cm_pred = np.round(predictions_R[k[i]][N].copy()) #use predictions passed by
spotcheck function

    #Print heading and build confusion matrix
    print(modfits_R[k[i]][N])
    print(dataDescription_R[k[i]], "Data")
    df_confusion = pd.crosstab(Y_val, cm_pred, rownames=['Actual'], colnames=['P
redicted'], margins=True)
    df_confusion = df_confusion.reindex(columns=[3,4,5,6,7,8, "All"],fill_value=
0)
    print(df_confusion)

    #Calculate accuracy and print
    dfCropped = df_confusion.iloc[0:6,0:6].copy()
    diag = pd.Series(np.diag(dfCropped), index=[dfCropped.index, dfCropped.colum
ns])
    diagSum = sum(diag)
    acc = diagSum/(df_confusion.loc['All','All'].copy())
    print("\nNumber of correct predictions: ",diagSum)
```

```
    print("Accuracy: ",acc, "\n")
    i= i+1
```

```
LinearRegression()
Original Data
Predicted  3  4    5    6   7  8  All
Actual
3          0  0    3    2   0  0    5
4          0  0    8    6   0  0   14
5          0  0  154   67   0  0  221
6          0  0   42  130  10  0  182
7          0  0    8   52  21  0   81
8          0  0    0    8   3  0   11
All        0  0  215  265  34  0  514

Number of correct predictions:  305
Accuracy:  0.5933852140077821

LGBMRegressor()
Original Data
Predicted  3  4    5    6   7  8  All
Actual
3          0  0    3    2   0  0    5
4          0  0   10    4   0  0   14
5          0  0  163   54   4  0  221
6          0  0   45  115  22  0  182
7          0  0    9   52  20  0   81
8          0  0    0    9   2  0   11
All        0  0  230  236  48  0  514

Number of correct predictions:  298
Accuracy:  0.5797665369649806

XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_byt
ree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='dep
thwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_one
hot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child
_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=1
00, n_jobs=0,
```

```
             num_parallel_tree=1, predictor='auto', random_state=0,
reg_alpha=0,
             reg_lambda=1, ...)
Balanced Data
Predicted  3   4    5    6    7  8  All
Actual
3          0   0    3    2    0  0    5
4          1   1    7    4    1  0   14
5          0  21  123   62   15  0  221
6          0  11   47   77   46  1  182
7          0   1   10   32   37  1   81
8          0   1    1    3    6  0   11
All        1  35  191  180  105  2  514

Number of correct predictions:  238
Accuracy:  0.46303501945525294

LGBMRegressor()
Balanced Data
Predicted  3   4    5    6   7  8  All
Actual
3          0   0    5    0   0  0    5
4          0   0    8    6   0  0   14
5          0  14  148   51   7  1  221
6          0   3   57   84  34  4  182
7          0   2    9   34  36  0   81
8          0   0    0    5   6  0   11
All        0  19  227  180  83  5  514

Number of correct predictions:  268
Accuracy:  0.5214007782101168

Ridge()
Original Standardised Data
Predicted  3  4    5    6   7  8  All
Actual
3          0  0    3    2   0  0    5
4          0  0    8    6   0  0   14
5          0  0  154   67   0  0  221
6          0  0   42  130  10  0  182
7          0  0    8   52  21  0   81
8          0  0    0    8   3  0   11
All        0  0  215  265  34  0  514
```

```
Number of correct predictions:  305
Accuracy:  0.5933852140077821
```

Notes

As expected, when ran on the validation set, the results are much worse.

Classification

Show hidden markdown

Show hidden markdown

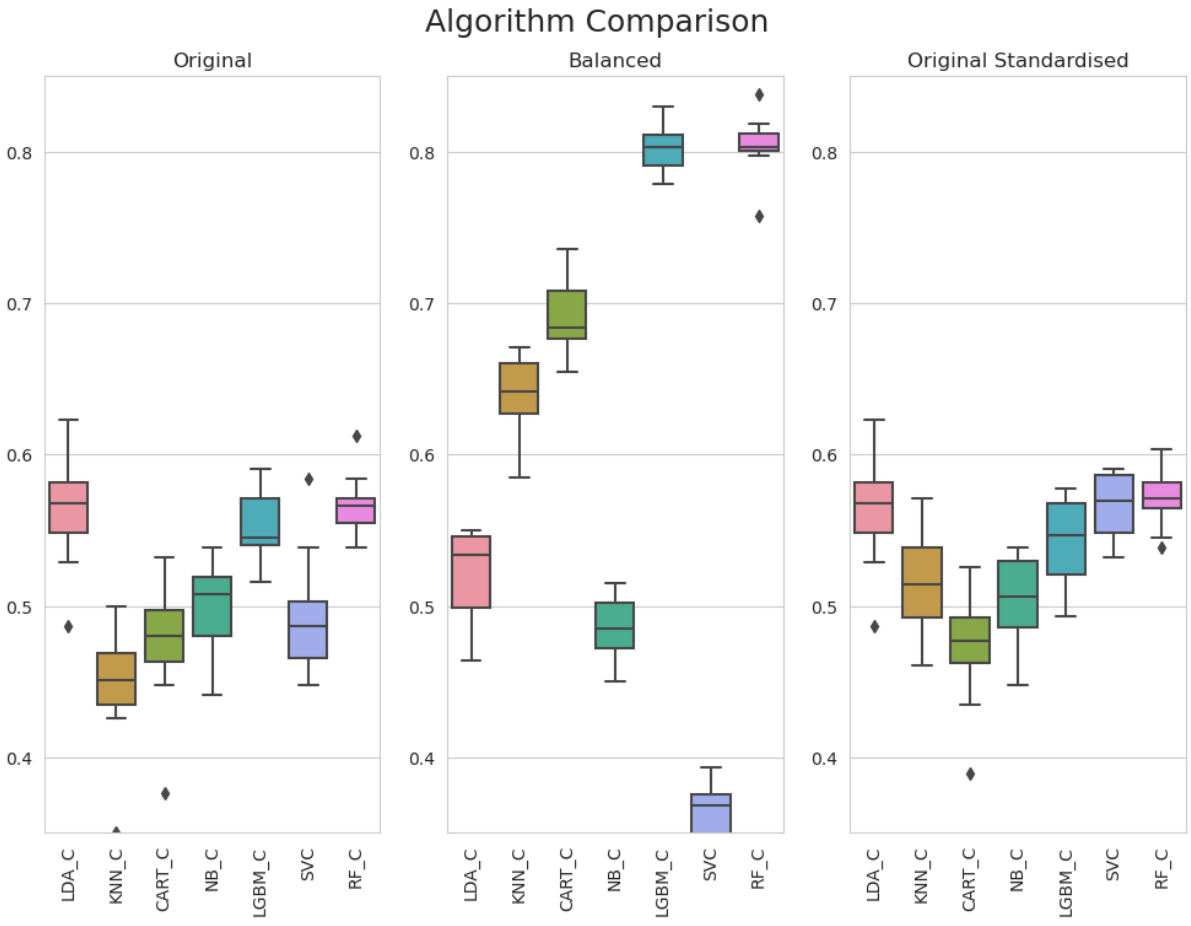
Show hidden markdown

In [24]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier

# select which data to use - origninal? balanced? standardised?
dataDescription_C = ["Original","Balanced", "Original Standardised"] #Ensure "St
andardised" is included for standardised data
X_s = [pd.DataFrame(X_train.copy()), pd.DataFrame(X_train_b.copy()), pd.DataFram
e(X_train_std.copy())]#predictors to train on
Y_s = [             Y_train.copy(),               Y_train_b.copy(),
Y_train.copy()] #outcome to train on
X_v = [pd.DataFrame(X_val.copy()), pd.DataFrame(X_val.copy()), pd.DataFrame(X_va
l_std.copy())] #predictors to validate on
Y_v = [             Y_val.copy(),               Y_val.copy(),               Y_va
l.copy()] #outcome to validate on

#define plot size
plt.figure(figsize=(12,8))
plt.suptitle('Algorithm Comparison',fontsize=18, y=0.95)

# prepare models
#add: lightGBM, XGBoost, RF
models_C = []
models_C.append(('LDA_C', LinearDiscriminantAnalysis()))
models_C.append(('KNN_C', KNeighborsClassifier()))
models_C.append(('CART_C', DecisionTreeClassifier()))
models_C.append(('NB_C', GaussianNB()))
models_C.append(('LGBM_C', LGBMClassifier()))
models_C.append(('SVC', SVC()))
models_C.append(('RF_C', RandomForestClassifier()))

#Run the check
names_C, predictions_C, absError_C, modfits_C = modelSpotCheck(X_s, Y_s, X_v, Y_
v, dataDescription_C, models_C, scoring = 'accuracy', ylim_min = 0.35, ylim_max
```

```
= 0.85)
# 'accuracy', 'neg_log_loss', 'roc_auc'
```

```
Results according to  Original data:
 ['LDA_C: 0.563569 (0.035694)', 'KNN_C: 0.446196 (0.038407)', 'CART
_C: 0.474051 (0.039312)', 'NB_C: 0.498056 (0.032165)', 'LGBM_C: 0.5
51885 (0.022954)', 'SVC: 0.494801 (0.039342)', 'RF_C: 0.567415 (0.0
19834)']
Results according to  Balanced data:
 ['LDA_C: 0.519972 (0.031217)', 'KNN_C: 0.640782 (0.024866)', 'CART
_C: 0.692287 (0.023841)', 'NB_C: 0.485169 (0.019269)', 'LGBM_C: 0.8
01250 (0.015283)', 'SVC: 0.360844 (0.022418)', 'RF_C: 0.803941 (0.0
19328)']
Results according to  Original Standardised data:
 ['LDA_C: 0.563569 (0.035694)', 'KNN_C: 0.516230 (0.033985)', 'CART
_C: 0.471458 (0.035788)', 'NB_C: 0.501295 (0.031603)', 'LGBM_C: 0.5
42778 (0.027326)', 'SVC: 0.566783 (0.020943)', 'RF_C: 0.571965 (0.0
19639)']
```



Compare the regression performance to the classification models by calculating a regression metric for both

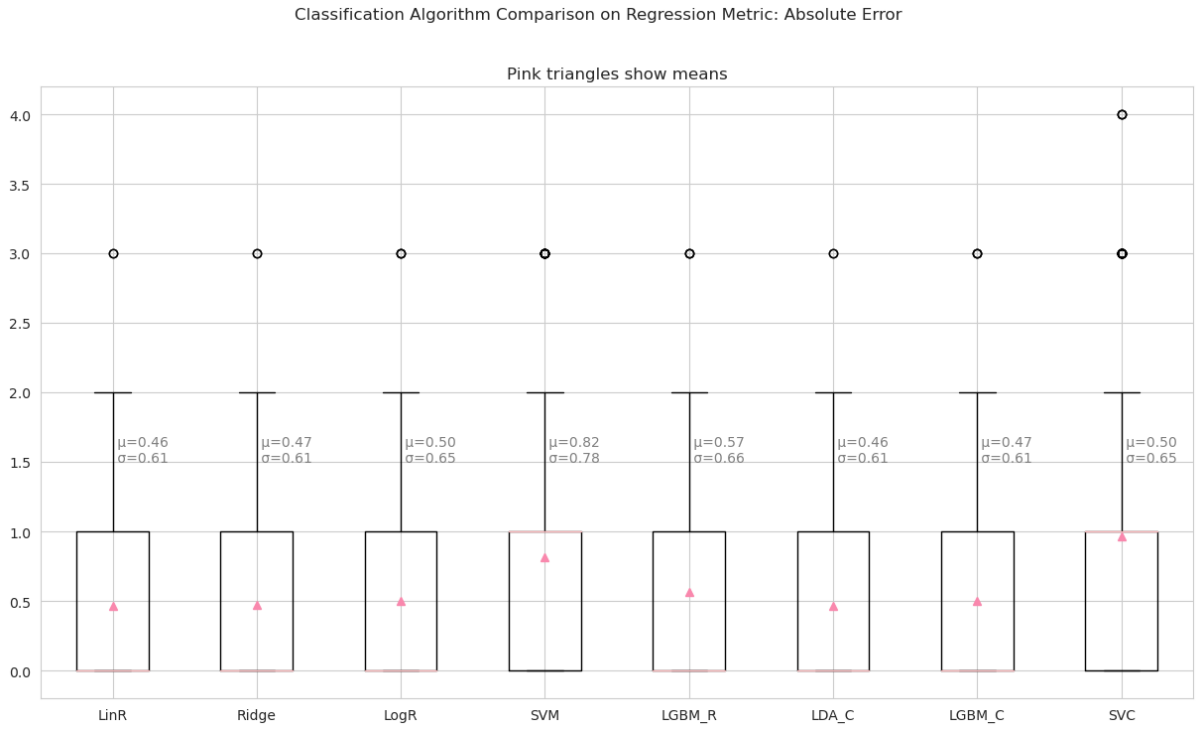
Only pick the best ones from each

In [25]:

```
# compare classification and regression results based on regression metric


#Select Regression models for comparison
models4cm_R = ["LinR", "Ridge", "LogR", "SVM", "LGBM_R"]
k_R =          [0,     0,        0,      1,     1] #choose which data set to use
N_R = [i for i in range(len(names_R)) if names_R[i] in models4cm_R] #find indece
s of chosen models
print(N_R)

#Select Classification models for comparison
models4cm_C = ["LDA_C", "LGBM_C", "SVC"]
k_C =          [0,      0,       1] #choose which data set to use
N_C = [i for i in range(len(names_C)) if names_C[i] in models4cm_C] #find indece
s of chosen models
print(N_C)

absError_combined = []
m = []
st = []

for i in range(len(models4cm_R)):
    absError_combined.append(absError_R[k_R[i]][N_R[i]]) #add all the abs_Erro
r's for the selected models, from the chosen data transforms
    m.append(np.mean(absError_combined[i]))
    st.append(np.std(absError_combined[i]))

for i in range(len(models4cm_C)):
    absError_combined.append(absError_R[k_C[i]][N_C[i]]) #add all the abs_Erro
r's for the selected models, from the chosen data transforms
    m.append(np.mean(absError_combined[i]))
    st.append(np.std(absError_combined[i]))


# boxplot algorithm comparison

style = dict(size=10, color='gray')
fig = pyplot.figure(figsize=(15, 8))
fig.suptitle('Classification Algorithm Comparison on Regression Metric: Absolute
Error')
ax = fig.add_subplot(111)
bp = pyplot.boxplot(absError_combined, showmeans=True)
ax.set_xticklabels(models4cm_R + models4cm_C)
```

```
ax.set_title('Pink triangles show means')

for i in range(len(absError_combined)):
    ax.text(i+1, 1.5, ' μ={:.2f}\n σ={:.2f}'.format(m[i], st[i]), **style)

pyplot.show()
```

```
[0, 1, 4, 7, 9]
[0, 4, 5]
```



In [26]:

```
#build confusion matrices for rounded regression values for specified models

from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

models4cm = ["LDA_C", "KNN_C","CART_C", "LGBM_C", "LGBM_C", "RF_C", "RF_C"]
k =          [0,        1,      1,       0,          1,       0,       1] #choos
e which data set to use

for i in range(len(models4cm)):
    df_confusion = 0
    dfCropped = 0
    N = names_C.index(models4cm[i])

    #if model was fit on standardised data then also use standardised validationio
n data
    if "Standardised" in dataDescription_C[k[i]]:
        modelpred = modfits_C[k[i]][N].predict(X_val_std)
    else:
        modelpred = modfits_C[k[i]][N].predict(X_val)

    #Print heading and build confusion matrix
    print(modfits_C[k[i]][N])
    print(dataDescription_C[k[i]], "Data")
    #cm_pred = modelpred.copy()
    cm_pred = predictions_C[k[i]][N].copy() #use predictions passed by spotcheck
function
    df_confusion = pd.crosstab(Y_val, cm_pred, rownames=['Actual'], colnames=['P
redicted'], margins=True)
    df_confusion = df_confusion.reindex(columns=[3,4,5,6,7,8, "All"],fill_value=
0)
    print(df_confusion)

    #Work out the accuracy and print
    dfCropped = df_confusion.iloc[0:6,0:6].copy()
    diag = pd.Series(np.diag(dfCropped), index=[dfCropped.index, dfCropped.colum
ns])
    diagSum = sum(diag)
    acc = diagSum/(df_confusion.loc['All','All'].copy())
    print("\nNumber of correct predictions: ",diagSum)
    print("Accuracy: ",acc, "\n")

    i= i+1
```

```
LinearDiscriminantAnalysis()
Original Data
Predicted  3  4    5    6   7  8  All
Actual
3          0  0    4    1   0  0    5
4          0  0   10    4   0  0   14
5          1  0  172   47   1  0  221
6          0  0   58  106  18  0  182
7          0  0   12   43  26  0   81
8          0  0    1    6   3  1   11
All        1  0  257  207  48  1  514

Number of correct predictions:  305
Accuracy:  0.5933852140077821

KNeighborsClassifier()
Balanced Data
Predicted   3   4    5   6   7   8  All
Actual
3           0   0    3   2   0   0    5
4           0   5    4   1   3   1   14
5          27  46   88  28  28   4  221
6          40  22   34  38  32  16  182
7           7  12    7  24  22   9   81
8           3   0    3   2   2   1   11
All        77  85  139  95  87  31  514

Number of correct predictions:  154
Accuracy:  0.29961089494163423

DecisionTreeClassifier()
Balanced Data
Predicted  3   4    5    6   7   8  All
Actual
3          0   3    1    0   1   0    5
4          0   2    7    1   4   0   14
5          3  35  104   61  17   1  221
6          2  13   43   60  42  22  182
7          2   7   12   24  23  13   81
8          1   0    1    6   0   3   11
All        8  60  168  152  87  39  514

Number of correct predictions:  192
```

```
Accuracy:  0.3735408560311284

LGBMClassifier()
Original Data
Predicted  3  4    5    6   7  8  All
Actual
3          0  0    4    1   0  0    5
4          0  0   12    2   0  0   14
5          0  0  161   54   6  0  221
6          0  0   53  105  24  0  182
7          0  0   13   48  19  1   81
8          0  0    0    8   3  0   11
All        0  0  243  218  52  1  514

Number of correct predictions:  285
Accuracy:  0.5544747081712063

LGBMClassifier()
Balanced Data
Predicted  3  4    5    6   7   8  All
Actual
3          0  0    2    3   0   0    5
4          0  0    9    5   0   0   14
5          0  7  148   57   7   2  221
6          5  1   53   84  36   3  182
7          1  0   14   33  28   5   81
8          0  0    0    4   7   0   11
All        6  8  226  186  78  10  514

Number of correct predictions:  260
Accuracy:  0.5058365758754864

RandomForestClassifier()
Original Data
Predicted  3  4    5    6   7  8  All
Actual
3          0  0    3    2   0  0    5
4          0  0   12    2   0  0   14
5          0  0  169   50   2  0  221
6          0  0   48  119  15  0  182
7          0  0   13   50  18  0   81
8          0  0    0    8   3  0   11
All        0  0  245  231  38  0  514
```

```
Number of correct predictions:  306
Accuracy:  0.5953307392996109

RandomForestClassifier()
Balanced Data
Predicted  3   4    5    6   7   8  All
Actual
3          0   0    4    1   0   0    5
4          0   0   11    2   1   0   14
5          1  17  148   46   7   2  221
6          4   6   45   83  38   6  182
7          1   0   12   30  33   5   81
8          0   0    0    4   5   2   11
All        6  23  220  166  84  15  514

Number of correct predictions:  266
Accuracy:  0.5175097276264592
```

Notes:

We see the following with both the regression and classification approach: Certain models trained on the balanced data doesn't perform anywhere near as well as the impressive looking cross-validation results predicted (which were based of course on the oversampled outcome data insetad of the real validation data). We can see from the confusion matrices that these models struggeled with classes 3,4 and 8. These were the classes with very little data to start with (12, 55 and 39 observations respectively compared to between 333 and 830 in the other classes). We conclude that these models (KNN, CART, LGBM, RF, XGB) do well with balanced data, but our SMOTE oversampling didn't work with based on the small amount of data available.

The performance between the best regression models (Linear Regression, Ridge Regression, SVM and LGBM) and classification models (LDA, RF, LGBM) were comparable. Note of these models were tuned so this performance is only indicative as a spot check, which helps us determine where to focus our efforts going forward.