Wine-Pandas

September 21, 2019

[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

sns.set()
```

1 1 - Loading the data

[3]:
```
file_path = '../data/wine/winequality-red.csv'
raw_data  = pd.read_csv(file_path, delimiter=';')
raw_data.head(5)
```

[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

2 2 - Cleaning the data

For this particular dataset, the cleaning steps will consist of:

1. Flagging and deleting rows containing missing values, if any
2. Flagging each cell that is not a float
3. Flagging and deleting duplicate rows, if any
4. Flagging and flagging outliers in the data
5. Deleting flagged errors

The different flags will be:

- 'ok' (no issue found with row)
- 'not a float'
- 'duplicate'
- 'potential outlier'

[4]:
```
# Add a flag column to the pandas dataset, will be used to flag potential␣
 ↪problems in the data
raw_data['flag'] = 'ok' # by default, flag everything as OK, this may change as␣
 ↪problems are found
raw_data.head(5)
```

[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

	alcohol	quality	flag
0	9.4	5	ok
1	9.8	5	ok
2	9.8	5	ok
3	9.8	6	ok
4	9.4	5	ok

[5]:
```
columns_to_clean = [c for c in raw_data.columns if c != 'flag']
print(columns_to_clean)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
```

```
'sulphates', 'alcohol', 'quality']
```

2.0.1 1 - Flag missing values

[6]:
```
def flag_missing(df):
    print(df.isna().sum())

flag_missing(raw_data)
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
flag                    0
dtype: int64
```

As we can see above, there are no NaN values in the red wine dataframe. We can move to the next step.

2.0.2 2 - Make sure every cell is a float

[7]:
```
def isFloat(string):
    try:
        float(string)
        return True
    except ValueError:
        return False

def flag_not_float(df, column):
    for i in df.index:
        if not isFloat(df[column][i]):
            df.at[i, 'flag'] = 'not a float'
    return df

# Test that all columns are made of float values
for column in columns_to_clean:
    raw_data = flag_not_float(raw_data, column)
```

2.0.3 3 - Identify potential duplicate rows

[8]:
```
def flag_duplicates(df):

    duplicate_rows_df = df[df.duplicated()]
    if duplicate_rows_df.shape[0] == 0:
        print("No duplicates found in dataframe")
        return raw_data

    # If we reach this part of the function then we do have duplicates in the␣
 ↪data
    print("Number of duplicate rows = %s" % duplicate_rows_df.shape[0])
    for i in duplicate_rows_df.index:
        raw_data.at[i, 'flag'] = 'duplicate'
    return raw_data


raw_data = flag_duplicates(raw_data)
raw_data.head()
```

```
Number of duplicate rows = 240
```

[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	flag
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	ok
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	ok
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	ok
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	ok
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	duplicate

As we can see here, rows #0 and #4 are indeed duplicates. Duplicate row(s) are now flagged and will be removed from the dataframe in the last step of the data cleaning.

2.0.4 4 - Flag outlier data

First we can plot the boxplots to see how each variable is represented in the dataset. Then, assuming the distributions are Gaussian, one could flag as an outlier any value that falls outside of 3 standard deviations from the mean. Since we don't know if the Gaussian assumption holds, we will run the models with different versions of the dataset to determine whether outlier flagging helped improve the predictions.

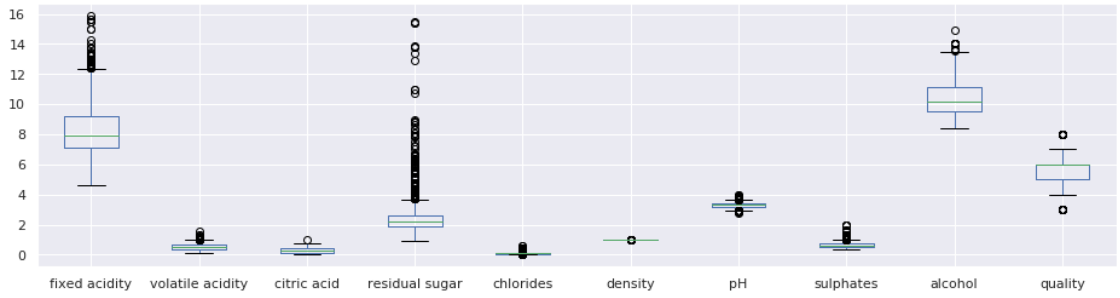
1. after deleting the duplicate rows only, and
2. after deleting the duplicate rows *as well as* what we have considered to be outlier values

To evaluate which version performs best.

Boxplots

[9]:
```
def draw_boxplots(raw_data, columns_to_clean):
    plt.figure(figsize=(16, 4))
    raw_data[columns_to_clean].boxplot()
    plt.show()

# Sulfur variables have much higher values than the rest, so we seperated them
sulfur_vars = ['free sulfur dioxide', 'total sulfur dioxide']
other_vars  = [v for v in columns_to_clean if not v in sulfur_vars]

draw_boxplots(raw_data, sulfur_vars)
```

[10]:
```
draw_boxplots(raw_data, other_vars)
```



Flag potential outliers

[11]:
```
def flag_outliers(df, columns_to_clean, n_std=3):
    """
    n_std: number of standard deviations above which a value is considered to␣
 ↪be an outlier
    If a given row will be flagged if it contains at least one outlier
    """

    for col in columns_to_clean:
        # sumamry statistics
        col_mean, col_std = np.mean(df[col]), np.std(df[col])
        # identify outliers
        cut_off = col_std * n_std
        # this is the range of acceptable data
        lower, upper = col_mean - cut_off, col_mean + cut_off

        # outliers
        outliers = df[(df[col]<lower) | (df[col]>upper)]
        for i in outliers.index:
            df.at[i, 'flag'] = 'potential outlier'
    return df

raw_data = flag_outliers(raw_data, columns_to_clean)
```

2.0.5 5 - Delete flagged data

Summary of the flagging : we have 223 duplicates and 148 rows containing at least one outlier value.

[12]:
```
summary = raw_data.groupby('flag').count()['quality']
summary
```

[12]:
```
flag
duplicate             223
ok                   1228
potential outlier     148
Name: quality, dtype: int64
```

Cleanup

[13]:
```
raw_data.head(5)
```

[13]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality       flag
0      9.4        5         ok
1      9.8        5         ok
2      9.8        5         ok
3      9.8        6         ok
4      9.4        5  duplicate
```

[14]:
```
def get_clean_dataframe(df, delete_outliers=False):
    """
    The 'delete_outliers' attribute can be set to True to also remove rows␣
 ↪flagged as outliers
    """
    if delete_outliers is True:
        clean_data = df[df['flag']=='ok']
    else:
        clean_data = df[(df['flag']=='ok') | (df['flag']=='potential outlier')]
    # Drop the flag column before returning the clean data
    return clean_data.drop('flag', axis=1)


# Get the clean data
clean_df   = get_clean_dataframe(raw_data) # this is to skip␣
 ↪SettingWithCopyWarning from Pandas
clean_data = clean_df.copy()
# Create the binary y column
clean_data['y'] = np.where(clean_df['quality'] >= 6.0, 1.0, 0.0)
clean_data.head()
```

[14]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
```

```
5            7.4              0.66         0.00             1.8      0.075

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
5                 13.0                  40.0   0.9978  3.51       0.56

   alcohol  quality    y
0      9.4        5  0.0
1      9.8        5  0.0
2      9.8        5  0.0
3      9.8        6  1.0
5      9.4        5  0.0
```

3 3 - Exploratory data analysis

This section includes:

1. Histograms of each variable
2. Scatterplots (xi,y)
3. The new boxplots

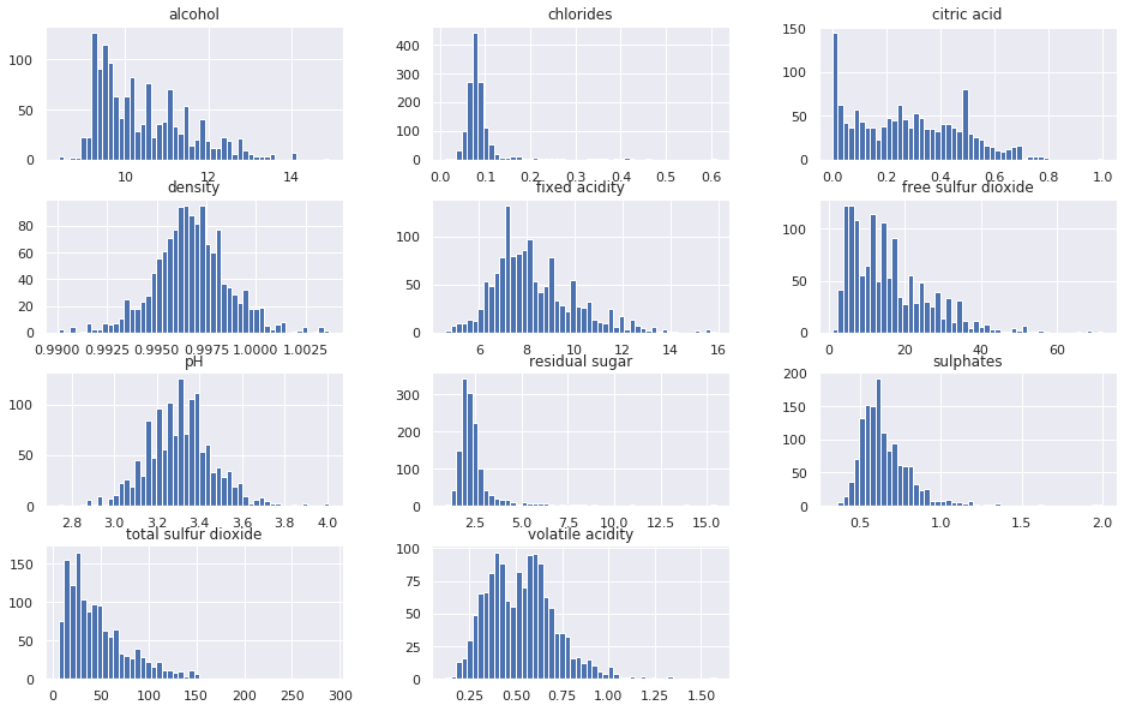
[15]:
```
y_vars = ['quality', 'y']
x_vars = [var for var in clean_data.columns.tolist() if not var in y_vars]

print(x_vars)
print(y_vars)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
'sulphates', 'alcohol']
['quality', 'y']
```

3.1 1 - Histograms

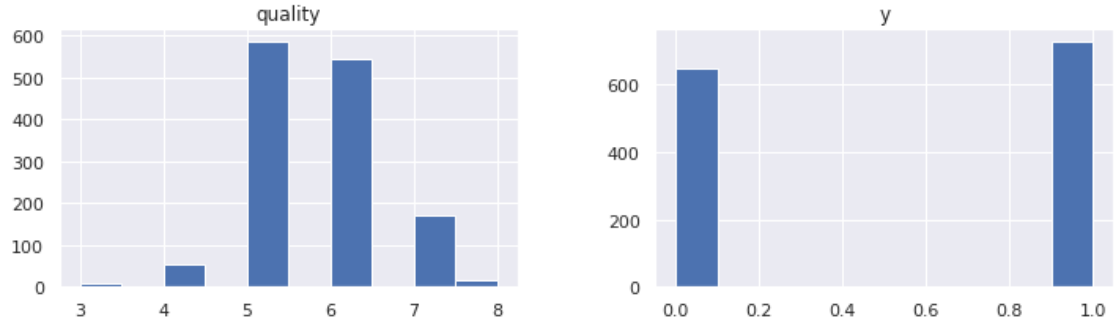
3.1.1 Input variables

[16]:
```
def draw_histogram(df, bins=50, width=12, height=5):
    df.hist(bins=bins, figsize=(width,height))

draw_histogram(clean_data[x_vars], bins=50, width=16, height=10)
```



3.1.2 Outputs variables

[17]:
```
draw_histogram(clean_data[y_vars], bins=10, width=12, height=3)
```



3.2 2 - Distribution of the input variables, by quality category

[67]:
```
def plot_dist_by_category(df, x, y_cat, y_cont):

    fig = plt.figure(figsize=(16,4))
```

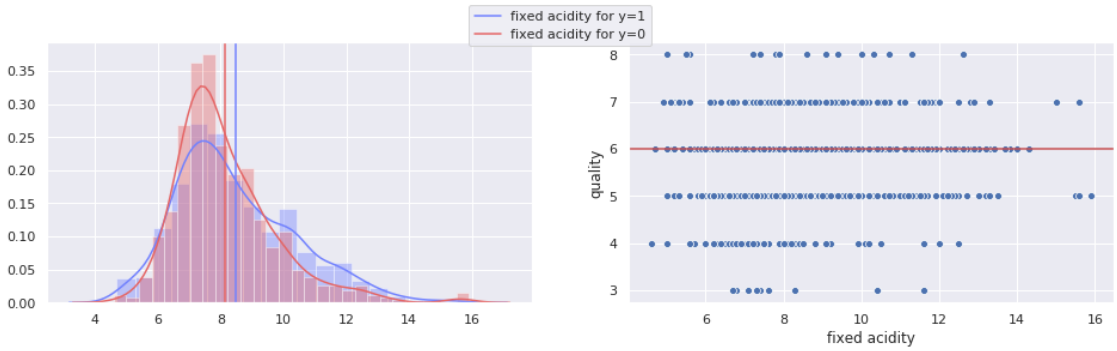
```
    #-------------------------------------------------------------
    # First plot : P(Xi|Y=0) versus P(Xi|Y=1)
    #-------------------------------------------------------------
    plt.subplot(1, 2, 1)

    pos_class = df[df[y_cat] == 1.0] # positive class
    neg_class = df[df[y_cat] == 0.0] # negative class

    pos_mean = float(np.mean(pos_class[[x]])) # mean, positive class
    neg_mean = float(np.mean(neg_class[[x]])) # mean, negative class

    sns.distplot(pos_class[[x]], color='#7282ff')
    sns.distplot(neg_class[[x]], color='#e56666')

    plt.axvline(pos_mean, color='#7282ff')
    plt.axvline(neg_mean, color='#e56666')

    fig.legend(labels=['%s for y=1' % x,'%s for y=0' % x], loc='upper center')

    #-------------------------------------------------------------
    # Seconf plot : Scatterplots
    #-------------------------------------------------------------
    plt.subplot(1, 2, 2)
    sns.scatterplot(x=x, y=y_cont, data=df)
    plt.axhline(y=6.0, color='r') # Horizontal line that separates good and bad␣
↪quality wine
    plt.show()

var_list = clean_data.columns.tolist()
print(var_list)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
'sulphates', 'alcohol', 'quality', 'y']
```

3.2.1 fixed acidity

From the important overlap between the two distributions below it is unclear that fixed acidity varies significantly between wines of quality=1 and wines of quality=0.

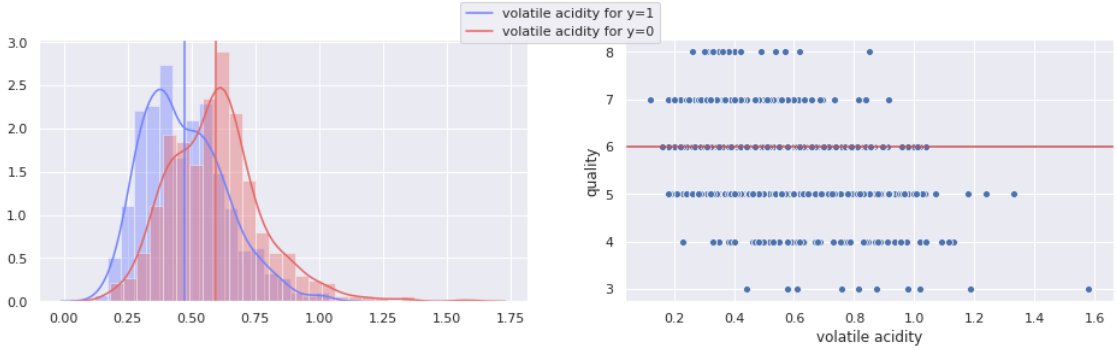
[68]:
```
plot_dist_by_category(clean_data, 'fixed acidity', 'y', 'quality')
```



3.2.2 volatile acidity

Despite the overlap, the means of the two distributions below seem to differ between wines of quality=1 and wines of quality=0.

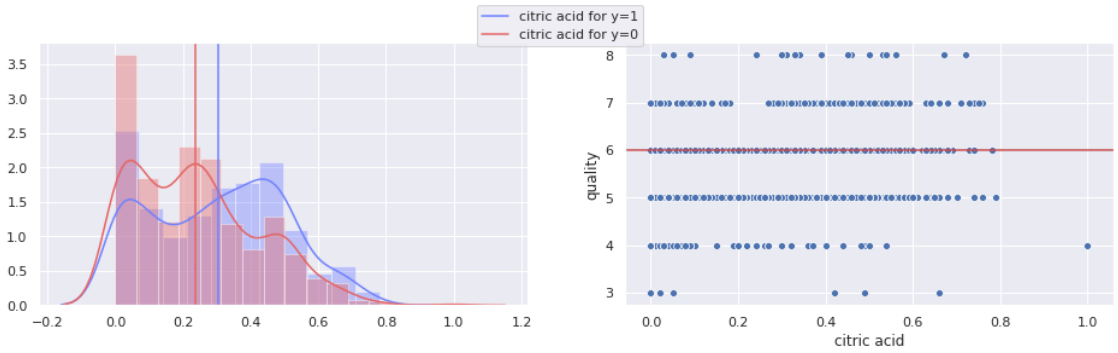
[69]:
```
plot_dist_by_category(clean_data, 'volatile acidity', 'y', 'quality')
```



3.2.3 citric acid

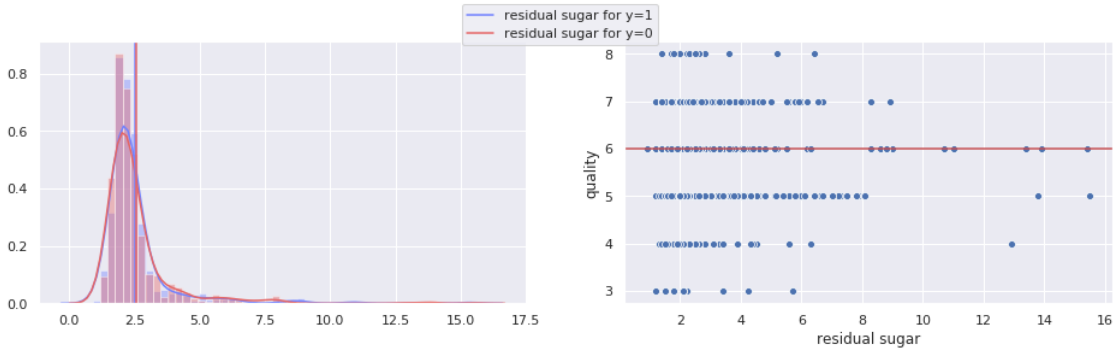
Despite the overlap, the means of the two distributions below seem to differ between wines of quality=1 and wines of quality=0.

[70]:
```
plot_dist_by_category(clean_data, 'citric acid', 'y', 'quality')
```



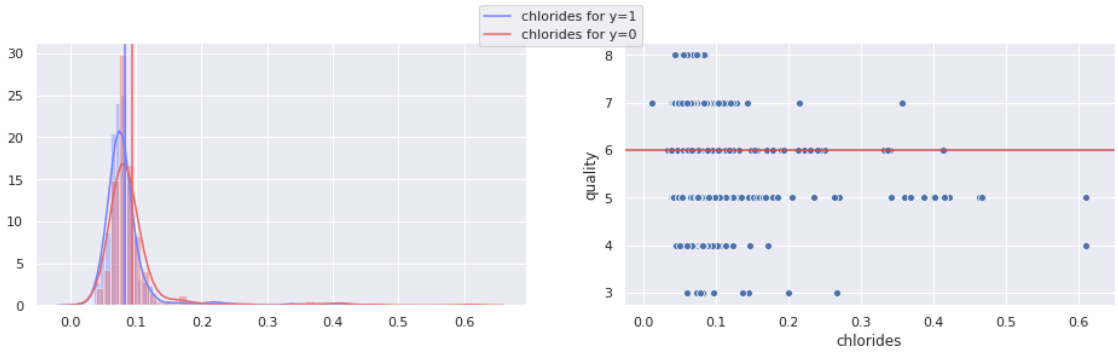
3.2.4 residual sugar

[71]:
```
plot_dist_by_category(clean_data, 'residual sugar', 'y', 'quality')
```



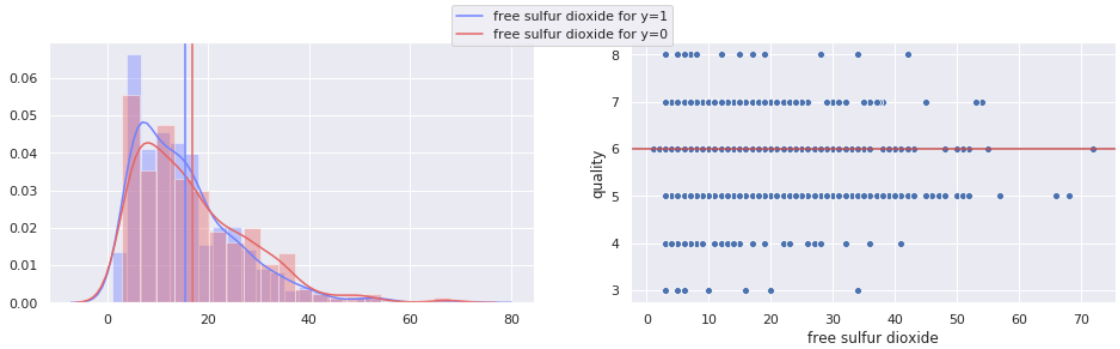
3.2.5 chlorides

[72]:
```
plot_dist_by_category(clean_data, 'chlorides', 'y', 'quality')
```



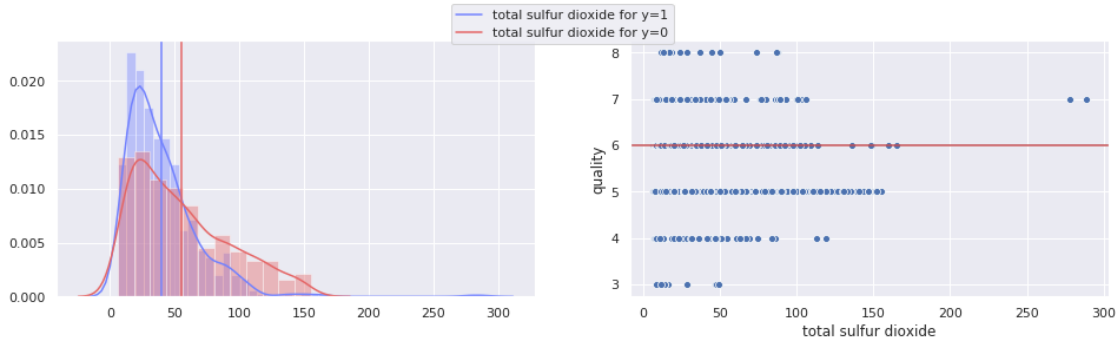
3.2.6 free sulfur dioxide

[73]:
```
plot_dist_by_category(clean_data, 'free sulfur dioxide', 'y', 'quality')
```



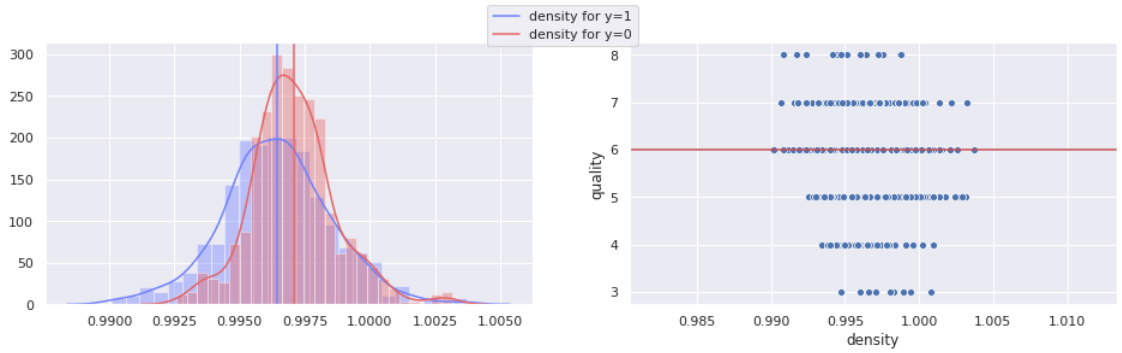
3.2.7 total sulfur dioxide

[74]:
```
plot_dist_by_category(clean_data, 'total sulfur dioxide', 'y', 'quality')
```



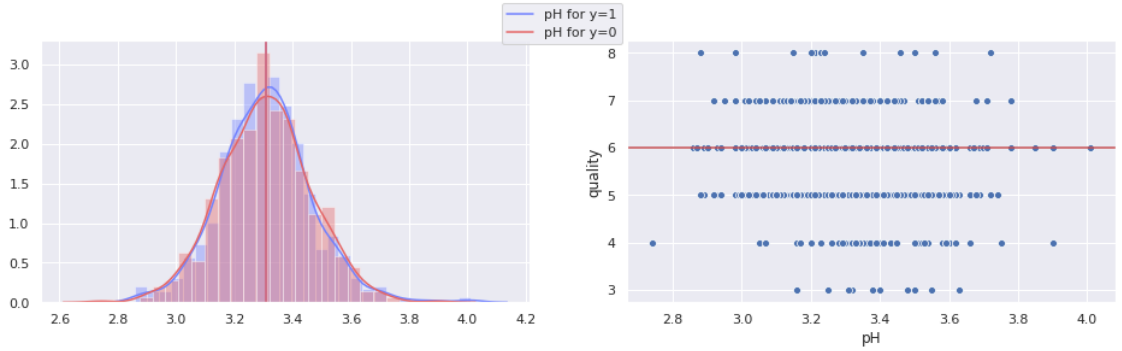
3.2.8 density

[75]:
```
plot_dist_by_category(clean_data, 'density', 'y', 'quality')
```



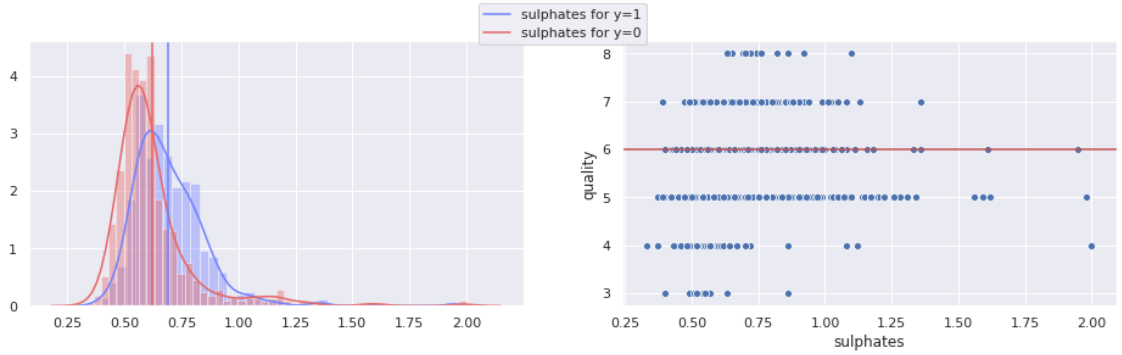
3.2.9 pH

[76]:
```
plot_dist_by_category(clean_data, 'pH', 'y', 'quality')
```



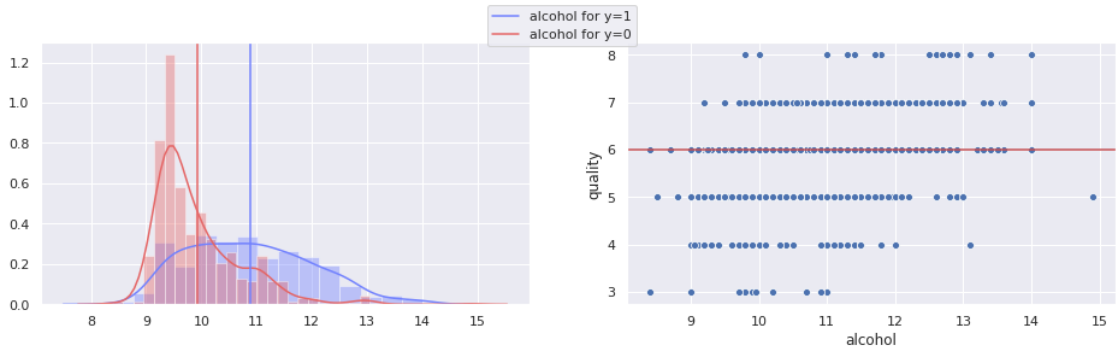
3.2.10 sulphates

[77]:
```
plot_dist_by_category(clean_data, 'sulphates', 'y', 'quality')
```



3.2.11 alcohol

[78]:
```
plot_dist_by_category(clean_data, 'alcohol', 'y', 'quality')
```



3.3 3 - Correlation analysis

[89]:
```
corr = clean_data[x_vars + ['y']].corr()
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

[89]:
```
<pandas.io.formats.style.Styler at 0x7ff5e590c4e0>
```

The strongest absolute correlations with the y value are with **alcohol**, **total sulfure dioxide** and **volatile acidity**, which reflects the conclusions of the analysis by histogram and scatterplot.

4 4 - Turn the dataframe into a Numpy array for the rest of the project

[83]:
```
def dataframe_to_narray(df, x_vars, y_var):

    X = df[x_vars].to_numpy()
    y = df[y_var].to_numpy()

    return X,y

X, y = dataframe_to_narray(clean_data, x_vars, 'y')

print(X)
```

```
print(y)
```

```
[[ 7.4    0.7    0.    …  3.51   0.56   9.4  ]
 [ 7.8    0.88   0.    …  3.2    0.68   9.8  ]
 [ 7.8    0.76   0.04  …  3.26   0.65   9.8  ]
 …
 [ 5.9    0.55   0.1   …  3.52   0.76  11.2  ]
 [ 5.9    0.645  0.12  …  3.57   0.71  10.2  ]
 [ 6.     0.31   0.47  …  3.39   0.66  11.   ]]
[0. 0. 0. … 1. 0. 1.]
```

[ ]: