Final project Statistical Learning: red and white wines datasets

Introduction

In [1]:
```
# Useful libraries
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import math
import numpy as np
from scipy.stats import norm
from IPython.core.display import HTML
```

In [149]:
```
# Import of data from csv files
wqw = pd.read_csv("winequality-white.csv",sep=";")
wqr = pd.read_csv("winequality-red.csv",sep=";")
display(HTML("Let's have a first glance at  each dataset (one for red wine and one for white
wine):"))
display(HTML("<h3 style='text-align: center;'>Red wine<br></h3>"),wqr.head(), HTML("<h3 styl
e='text-align: center;'>White wine<br></h3>"), wqw.head())
```

Let's have a first glance at each dataset (one for red wine and one for white wine):

Red wine

	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

White wine

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

In [3]:
```
display(HTML("The given dataset is composed of two sub-datasets: one for red wine and one wh
ite wine. The white wine one is composed of {} rows and the red wine one of {} rows.".format
(len(wqw.index), len(wqr.index))
     +"<br>They both have the same columns (features): <br><br>{}.".format((wqr.columns[:].t
olist()))
     +"<br><br>The inputs include objective tests (e.g. PH values) and the output is based o
n sensory data (median of at least 3 evaluations made by wine experts).<br>Each expert grade
d the wine quality between 0 (very bad) and 10 (very excellent). It corresponds to the last
 column of the previously displayed data."))
```

The given dataset is composed of two sub-datasets: one for red wine and one white wine. The white wine one is composed of 4898 rows and the red wine one of 1599 rows.
They both have the same columns (features):

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

The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts).
Each expert graded the wine quality between 0 (very bad) and 10 (very excellent). It corresponds to the last column of the previously displayed data.

Question we wish to address:

We suppose we are a mass distribution brand - like Carrefour - and that our main goal is to **select wines amongst our suppliers' proposals**. We wish to propose qualitative wines to our clients, so we would like to select the best wines. **We do not have any wine experts in our staff so we wish to objectify wine quality based on objective data**. For a given wine, the goal is thus to analyze its characteristics and based on it assign it a quality score (0-10). Therefore we wish **to predict wine quality** depending on its characteristics. We would also like to **give some guidelines to our suppliers** in order to make better wines.

Document structure

The document is structured in the following way:

- **Introduction**
- **1. Exploratory analysis**
  - **1.1. Comparison of datasets statistics: can we merge both datasets ?**
  - **1.2. First dataset analysis: a study of correlation between features**
- **2. Classification and prediction task: can we predict wine quality based on its objective characteristics ?**
  - **2.1. Classification and prediction with kNN**
  - **2.2. Classification and prediction with QDA**
- **3. Classification and prediction: can we predict wine quality with only two classes ?**
- **4. Regression methods: can we give any guidelines to our suppliers to produce better wines ?**
  - **4.1. White wine regressions on density**
  - **4.2. Red wine regressions on volatile acidity**
- **5. Further possible developments**
- **Conclusion**

1. Exploratory analysis

1.1 Comparison of datasets statistics

Data has been splitted in two separate datasets. Before going further in our analysis we wish to analyze **if there seems to be a reason for such a choice.** As the first step we would like to study **the statistics of each features** of the datasets and compare them. Hereafter, a comparison of each feature's statistics for white and red wine.

In [4]:
```
def stat_comparison(d, indent=0):
    for key, value in d.items():
        print('\t' * indent + str(key)+":")
        if isinstance(value, dict):
            stat_comparison(value, indent+1)
        else:
            print('\t' * (indent+1) + str(value))

stat_comp = {}
for c in wqr.columns:
    stat_comp[c]= {
            'mean' :{'red': wqr[c].mean(),
                     'white' : wqw[c].mean()},
            'median' : {'red' : wqr[c].median(),
                        'white' : wqw[c].median()},
            'std' : {'red' : wqr[c].std(),
                     'white' : wqw[c].std()}}

stat_comparison(stat_comp)
```

```
fixed acidity:
        mean:
                red:
                        8.31963727329581
                white:
                        6.854787668436097
        median:
                red:
                        7.9
                white:
                        6.8
        std:
                red:
                        1.7410963181276953
                white:
                        0.843868227687513
volatile acidity:
        mean:
                red:
                        0.5278205128205128
                white:
                        0.27824111882400976
        median:
                red:
                        0.52
                white:
                        0.26
        std:
                red:
                        0.17905970415353537
                white:
                        0.10079454842486534
citric acid:
        mean:
                red:
                        0.2709756097560976
                white:
                        0.33419150673744386
        median:
                red:
                        0.26
                white:
                        0.32
        std:
                red:
                        0.19480113740531857
                white:
                        0.12101980420298249
residual sugar:
        mean:
                red:
                        2.53880550343965
                white:
                        6.391414863209474
        median:
                red:
                        2.2
                white:
                        5.2
        std:
                red:
                        1.4099280595072798
                white:
                        5.072057784014881
chlorides:
        mean:
                red:
                        0.08746654158849279
                white:
                        0.04577235606369946
        median:
                red:
                        0.079
                white:
                        0.043
        std:
                red:
                        0.0470653020100901
                white:
                        0.021847968093728798
free sulfur dioxide:
        mean:
                red:
                        15.874921826141339
                white:
                        35.30808493262556
        median:
                red:
                        14.0
                white:
                        34.0
        std:
                red:
                        10.460156969809725
                white:
                        17.00713732523259
total sulfur dioxide:
        mean:
                red:
                        46.46779237023139
                white:
                        138.36065741118824
        median:
                red:
                        38.0
                white:
                        134.0
        std:
                red:
                        32.895324478299074
                white:
                        42.49806455414291
density:
        mean:
                red:
                        0.9967466791744841
                white:
                        0.9940273764801959
        median:
                red:
                        0.99675
                white:
                        0.99374
        std:
                red:
                        0.0018873339538425554
                white:
                        0.0029909069169369337
pH:
        mean:
                red:
                        3.3111131957473416
                white:
                        3.1882666394446715
        median:
                red:
                        3.31
                white:
                        3.18
        std:
                red:
                        0.15438646490354277
                white:
                        0.1510005996150668
sulphates:
        mean:
                red:
                        0.6581488430268917
                white:
                        0.48984687627603113
        median:
                red:
                        0.62
                white:
                        0.47
        std:
                red:
                        0.16950697959010996
                white:
                        0.1141258339488323
alcohol:
```

```
        mean:
                red:
                        10.422983114446529
                white:
                        10.514267047774602
        median:
                red:
                        10.2
                white:
                        10.4
        std:
                red:
                        1.0656675818473946
                white:
                        1.230620567757318
quality:
        mean:
                red:
                        5.6360225140712945
                white:
                        5.87790935075541
        median:
                red:
                        6.0
                white:
                        6.0
        std:
                red:
                        0.807569439734705
                white:
                        0.8856385749678312
```

Statistics seem to be quite different from one dataset to the other so we better explore them separately. We deem this qualitative approach sufficient here - given the obvious discrepencies between datasets.

1.2 First analysis of the dataset: correlation between features

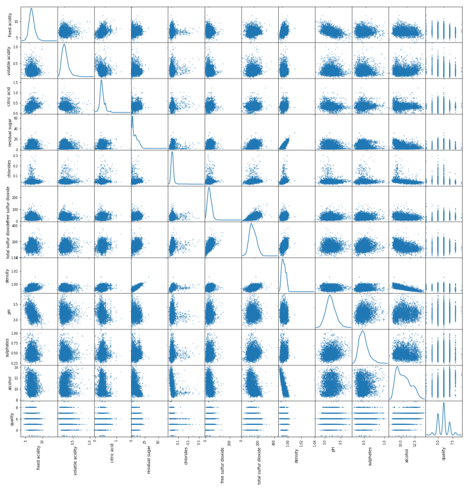
We wish to analyze **correlation between features** to detect interdependent features. In order to do so we first plot of each pairwise density and the compute correlation matrices to search for correlated features.

1.2.1 Scatter plots

Hereafter, the scatter plots allowing to visually see correlation between features and - on the diagonal - features' density estimation by kernel density estimation.

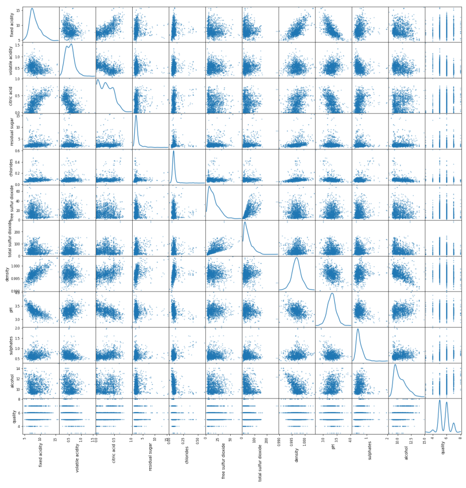
In [5]:
```
from pandas.plotting import scatter_matrix
spw = scatter_matrix(
    wqw,
    figsize  = [20, 20],
    marker   = "o",
    s        = 5,
    diagonal = "kde"
)

for ax in spw.ravel():
    ax.set_xlabel(ax.get_xlabel(), fontsize = 10, rotation = 90)
    ax.set_ylabel(ax.get_ylabel(), fontsize = 10, rotation = 90)
display(HTML("<h3 style='text-align: center;'>White wine scatter plots</h3>"))
plt.show()
```

White wine scatter plots



In [10]:
```
spr = scatter_matrix(
    wqr,
    figsize  = [20, 20],
    marker   = "o",
    s        = 5,
    diagonal = "kde"
)

for ax in spr.ravel():
    ax.set_xlabel(ax.get_xlabel(), fontsize = 10, rotation = 90)
    ax.set_ylabel(ax.get_ylabel(), fontsize = 10, rotation = 90)
display(HTML("<h3 style='text-align: center;'>Red wine scatter plots</h3>"))
plt.show()
```

Red wine scatter plots



For white wine we can observe that there seems to be **a positive correlation between** the following features:

- residual sugar and density
- density and fixed acidity
- free sulfure dioxide and total sulfure dioxide

and **a negative correlation** between:

- density and alcohol

For red wine we can observe that there seems to be **a positive correlation between** the following features:

- fixed acidity and citric acid
- fixed acidity and density
- free sulfure dioxide and total sulfure dioxide

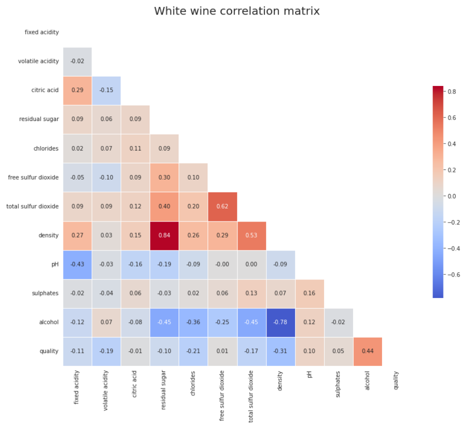
and **a negative correlation between**:

- fixed acidity and pH
- cictric acid and volatile acidity
- citric acid and pH
- density and alcohol

1.2.2 Correlation matrices

To complete the previous visual interpretation we also computed correlation matrices to quantify the degree of dependence between variables.

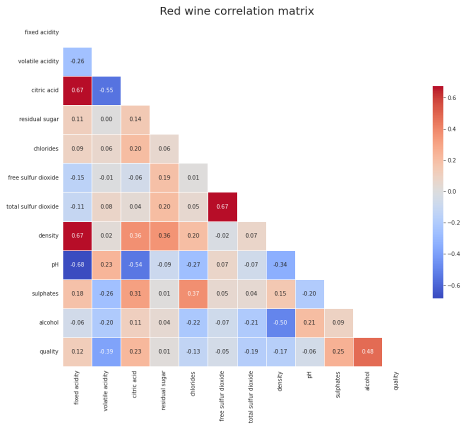
In [23]:
```
Cor = wqw.corr()
ones = np.ones(Cor.shape)
msk1 = np.triu(ones.astype(bool))

with sns.axes_style("white"):
    fg1, x1 = plt.subplots(figsize=(14, 14))
    sns.heatmap(Cor, cmap="coolwarm", mask=msk1, center=0, square=True, annot=True, linewidt
hs=0.5, fmt=".2f", cbar_kws={"shrink": .5})
plt.title("White wine correlation matrix", fontsize=20)
plt.show()
```


White wine correlation matrix

For white wine, we can infer from the correlation matrix above that there appears to be **a strong positive association between 'density' and 'residual sugar' (0.84)**. For white wine, this may be the direct result of a causal relationship between sugar content and wine density.There are also moderate positive correlations between 'total sulfur dioxide' and 'free sulfur dioxide' (0.62), and 'total sulfur dioxide' and 'density' (0.53).

Furthermore, we note that there is **a strong negative association between 'density' and 'alcohol'**. We may surmise that for white wine, there may exist a (negative) causal relationship between alcohol content and wine density.

In [24]:
```
Cor = wqr.corr()
ones = np.ones(Cor.shape)
msk1 = np.triu(ones.astype(bool))

with sns.axes_style("white"):
    fg1, x1 = plt.subplots(figsize=(14, 14))
    sns.heatmap(Cor, cmap="coolwarm", mask=msk1, center=0, square=True, annot=True, linewidt
hs=0.5, fmt=".2f", cbar_kws={"shrink": .5})
plt.title("Red wine correlation matrix", fontsize=20)
plt.show()
```


Red wine correlation matrix

For red wine, we can infer from the correlation matrix below that there appears to be **a moderate positive association between 'citric acid' and 'fixed acidity' (0.67)**, 'density' and 'fixed acidity' (0.67), 'total sulfure dioxyde' and 'free sulfure dioxide' (0.67). There is also a moderate negative association between 'pH' and 'fixed acidity' (-0.68), 'citric acid' and 'pH' (-0.54), and 'density' and 'alcohol' (-0.50).

Perhaps surpisingly, we note that there appears to be moderately **negative associations between 'citric acid' and 'volatile acidity' (-0.55)**. This seems to be counter-intuitive as one would expect positive associations between them.

2. Classification task

The idea of this section is to **try and predict wine quality based on its components**, the idea being that with a given composition we would be able to predict wine quality. We will use three different prediction methods based on :

- **kNN**
- **QDA**

We decided to apply the methods to the same datasets (splitted the same way). We chose not to use any random shuffling and set the size of the training set to be 80% of the data. It seems to be a frequent rule of thumb based on *Pareto principle*.

2.1 Classification and prediction with kNN classifier

The main parameter in kNN classifier - as its name suggests - is **the k parameter.** To select it we used an experimental procedure by trying to **minimize the error rate** as well as **maximizing the accuracy.** For each wine we plot the resulting graphs and select k graphically.

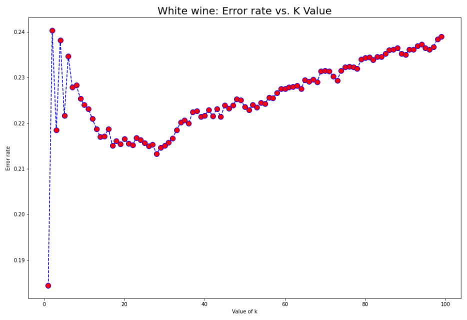
We proceeded the following way to choose the k-value:

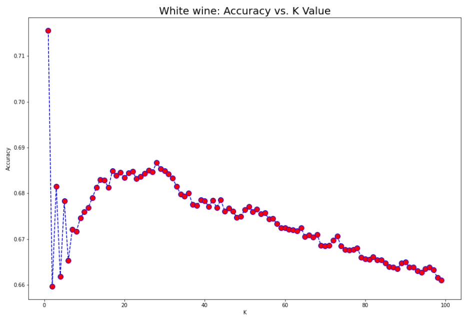
- We first splitted the dataset in two subsests: one for training and one for testing (without randomness in a deterministic way)
- We resplitted with a shuffle the training set to obtain again two subsets one for training and one for testing k-values and avoid testing it on real labels we wish to predict after
- We did the previous 10 times and averaged the error rate and accuracy and plotted the averaged values
- Based on this graphs we selected the k-value

2.1.1 Choice of the value of k

In [88]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw.iloc[:,:-1], columns=wqw.columns[:-1])
y = wqw['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)

error_rate = [0]*99
acc = [0]*99

for j in range(1,10):
    # Re-split training set to define k-value based on results on subset of training set
    X_train2, X_test2, y_train2, y_test2 = train_test_split(X_train, y_train, test_size=0.20
, shuffle=True)
    # Scaling
    scaler = StandardScaler()
    scaler.fit(X_train2)
    X_train2 = scaler.transform(X_train2)
    X_test2 = scaler.transform(X_test2)
    for i in range(1,100):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train2, y_train2)
        pred = knn.predict(X_test2)
        error_rate[i-1]+=(np.mean(pred != y_test2))
        acc[i-1]+=metrics.accuracy_score(y_test2, pred)

err = [error / 10 for error in error_rate]
accur = [accuracy / 10 for accuracy in acc]
# Error rate plot
plt.figure(figsize=(15,10))
plt.title("White wine: Error rate vs. K Value", fontsize=20)
plt.xlabel("Value of k")
plt.ylabel("Error rate")
plt.plot(range(1,100),err, color='blue', linestyle='dashed', marker='o', markerfacecolor='re
d', markersize=10)
plt.show()
# Accuracy plot
plt.figure(figsize=(15,10))
plt.plot(range(1,100),accur,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('White wine: Accuracy vs. K Value', fontsize=20)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```

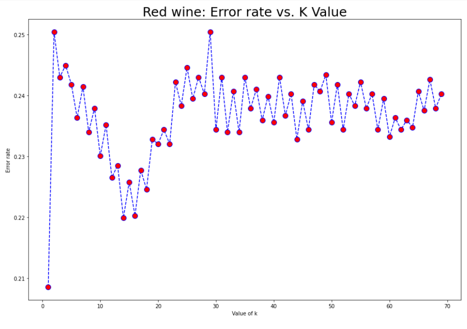


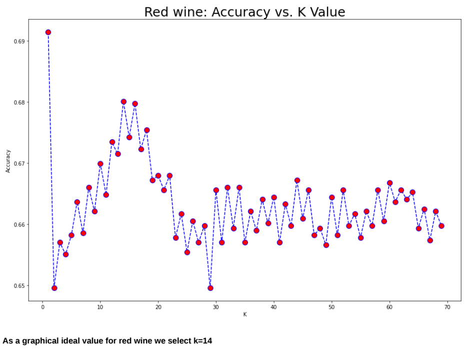


As a graphical ideal value for white wine we select k=28

In [89]:

```
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqr.iloc[:,:-1], columns=wqr.columns[:-1])
y = wqr['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)

error_rate = [0]*69
acc = [0]*69

for j in range(1,10):
    # Re-split training set to define k-value based on results on subset of training set
    X_train2, X_test2, y_train2, y_test2 = train_test_split(X_train, y_train, test_size=0.20
, shuffle=True)
    # Scaling
    scaler = StandardScaler()
    scaler.fit(X_train2)
    X_train2 = scaler.transform(X_train2)
    X_test2 = scaler.transform(X_test2)
    for i in range(1,70):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train2, y_train2)
        pred = knn.predict(X_test2)
        error_rate[i-1]+=(np.mean(pred != y_test2))
        acc[i-1]+=metrics.accuracy_score(y_test2, pred)

err = [error / 10 for error in error_rate]
accur = [accuracy / 10 for accuracy in acc]

# Error rate plot
plt.figure(figsize=(15,10))
plt.title("Red wine: Error rate vs. K Value", fontsize=25)
plt.xlabel("Value of k")
plt.ylabel("Error rate")
plt.plot(range(1,70),err, color='blue', linestyle='dashed', marker='o', markerfacecolor='re
d', markersize=10)
plt.show()
# Accuracy plot
plt.figure(figsize=(15,10))
plt.plot(range(1,70),accur,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Red wine: Accuracy vs. K Value', fontsize=25)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```





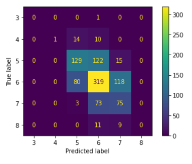
As a graphical ideal value for red wine we select k=14

2.1.2 Prediction on test set

In [92]:

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw.iloc[:,:-1], columns=wqw.columns[:-1])
y = wqw['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# kNN with k=61
classifier = KNeighborsClassifier(n_neighbors=28)
classifier.fit(X_train, y_train)
# Prediction
y_pred = classifier.predict(X_test)
# Confusion matrix
ConfM = confusion_matrix(y_test, y_pred)

display(HTML("<h3 style='text-align: center;'>White wine confusion matrix: </h3><br>"))
cm_display = ConfusionMatrixDisplay(ConfM,display_labels=[3,4,5,6,7,8]).plot()
plt.show()
display(HTML("<h3 style='text-align: center;'>White wine classification report: </h3><br>"))
print(classification_report(y_test, y_pred, zero_division=0))
```

White wine confusion matrix:



True label	3	4	5	6	7	8
3	0	0	0	1	0	0
4	0	1	14	10	0	0
5	0	0	129	122	15	0
6	0	0	80	319	118	0
7	0	0	3	73	75	0
8	0	0	0	11	9	0

White wine classification report:

```
              precision    recall  f1-score   support
```

```
           3       0.00      0.00      0.00         1
           4       1.00      0.04      0.08        25
           5       0.57      0.48      0.52       266
           6       0.60      0.62      0.61       517
           7       0.35      0.50      0.41       151
           8       0.00      0.00      0.00        20

    accuracy                           0.53       980
   macro avg       0.42      0.27      0.27       980
weighted avg       0.55      0.53      0.53       980
```

Here we can see that the prediction has **a 53% accuracy score which is a pretty bad score.**

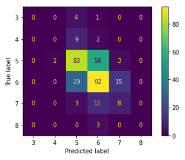
To assess how good the model performed we are going to focus on f-1 score which is the harmonic mean of precision and recall. It gives thus, at the same time, information on both parameters.

The f-1 score allows us to see that in the class of grade 6 we reach a f-1 score of 61% (but the support of this grade represents 53% of the total support). So our prediction is just above what would have been done with only the most basic model using previous repartition in the different classes. As we go on adjacent classes (7 and 5) the f-1 score decreases to 41% and 52% respectively. This could be explained by the relatively smallest amount of observations in these classes in the training set. Moreover, for the other classes none of the new observations have been guessed correctly by the model.

In [93]:

```
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqr.iloc[:,:-1], columns=wqr.columns[:-1])
y = wqr['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# kNN with k=61
classifier = KNeighborsClassifier(n_neighbors=14)
classifier.fit(X_train, y_train)
# Prediction
y_pred = classifier.predict(X_test)
# Confusion matrix
ConfM = confusion_matrix(y_test, y_pred)

display(HTML("<h3 style='text-align: center;'>Red wine confusion matrix: </h3><br>"))
cm_display = ConfusionMatrixDisplay(ConfM,display_labels=[3,4,5,6,7,8]).plot()
plt.show()
display(HTML("<h3 style='text-align: center;'>Red wine classification report: </h3><br>"))
print(classification_report(y_test, y_pred, zero_division=0))
```

Red wine confusion matrix:



Red wine classification report:

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00        11
           5       0.65      0.58      0.61       143
           6       0.56      0.68      0.61       136
           7       0.31      0.36      0.33        22
           8       0.00      0.00      0.00         3

    accuracy                           0.57       320
   macro avg       0.25      0.27      0.26       320
weighted avg       0.55      0.57      0.56       320
```

Here we can see that the prediction has **a 57% accuracy score which is still a pretty bad score.**

To assess how good the model performed we are going to focus on f-1 score which is the harmonic mean of precision and recall. It gives thus, at the same time, information on both parameters.

The f-1 score allows us to see that in the class of grade 6 we reach a f-1 score of 61% which is better than previoulsy but might be explained by the fact the the observations are more spread over class 5 and 6. Interestingly, the model performed quite good on the class 5 too with a f-1 score of 61%. As we go to the class 7 the f-1 score decreases to 33%. Moreover, for the other classes none of the new observations have been guessed correctly by the model.

To conclude on these results, the look pretty bad on both datasets, which could be explained by the relative concetration of observations within the the central classes. There are few obesrvations in the extreme classes which exlpains why the macro average f-1 score are so bad in both datasets (27% for white wine and 26% for red wine). Three possible thing came to our mind to overcome this problem:

- Rebalance observations to have a better distribution over classes
- Try another classification and prediction method
- Reduce the number of classes by grouping some classes

As a start we will try another classification and prediction method.

2.2 QDA

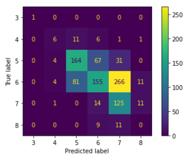
In this section we will try to reproduce the classification and prediction with the Quadratic Discriminant Analysis (QDA)

2.2.1 Prediction with QDA

In [94]:

```
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=(FutureWarning, UserWarning))
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import ConfusionMatrixDisplay

# White wine
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw.iloc[:,:-1], columns=wqw.columns[:-1])
y = wqw['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = QuadraticDiscriminantAnalysis()
classifier.fit(X_train, y_train)
y_pred = classifier.predict( X_test )

ConfM = confusion_matrix(y_test,y_pred)

display(HTML("<h3 style='text-align: center;'>White wine confusion matrix: </h3><br>"))
cm_display = ConfusionMatrixDisplay(ConfM, display_labels=[3,4,5,6,7,8]).plot()
plt.show()
display(HTML("<h3 style='text-align: center;'>White wine classification report: </h3><br>"))
print(classification_report(y_test, y_pred, zero_division=0))
```

White wine confusion matrix:



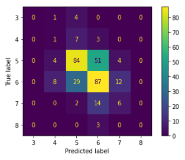
White wine classification report:

```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00         1
           4       0.40      0.24      0.30        25
           5       0.64      0.62      0.63       266
           6       0.62      0.30      0.40       517
           7       0.29      0.83      0.43       151
           8       0.00      0.00      0.00        20

    accuracy                           0.46       980
   macro avg       0.49      0.50      0.46       980
weighted avg       0.56      0.46      0.46       980
```

In [95]:

```
# ignore all future warnings
simplefilter(action='ignore', category=(FutureWarning, UserWarning))

# Red wine
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqr.iloc[:,:-1], columns=wqr.columns[:-1])
y = wqr['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = QuadraticDiscriminantAnalysis()
classifier.fit(X_train, y_train)
y_pred = classifier.predict( X_test )

ConfM = confusion_matrix(y_test,y_pred)

display(HTML("<h3 style='text-align: center;'>Red wine confusion matrix: </h3><br>"))
cm_display = ConfusionMatrixDisplay(ConfM, display_labels=[3,4,5,6,7,8]).plot()
plt.show()
display(HTML("<h3 style='text-align: center;'>Red wine classification report: </h3><br>"))
print(classification_report(y_test, y_pred, zero_division=0))
```

Red wine confusion matrix:



Red wine classification report:

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         5
           4       0.07      0.09      0.08        11
           5       0.67      0.59      0.62       143
           6       0.55      0.64      0.59       136
           7       0.27      0.27      0.27        22
           8       0.00      0.00      0.00         3

    accuracy                           0.56       320
   macro avg       0.26      0.27      0.26       320
weighted avg       0.55      0.56      0.55       320
```

As previously the results are quite bad, so we will not comment further and we decided to go for the last proposition listed above: **try to reduce the number of classes to see if we can better predict**.

3. Classification with less classes with kNN

As we are a mass distribution company, our goal is to provide our clients with good wines - but with a large enough variety of wines. **So we decided that we do not want to keep 'bad' wines with a quality score below 6**. We wish to discriminate only the wines that have a quality score below 6. This choice of keeping all wines with quality score equal or above 6 is based on the fact that usually the better the wine, the more expensive it is. And as a distribution company we have to propose affordable wines: so we decided to segregate datasets above and below 6.

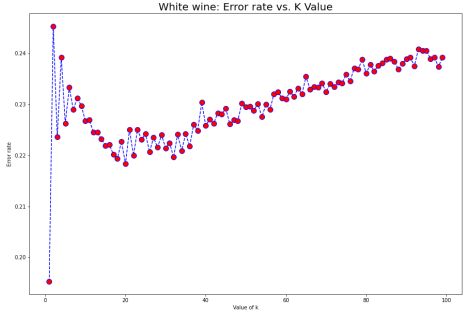
In [99]:

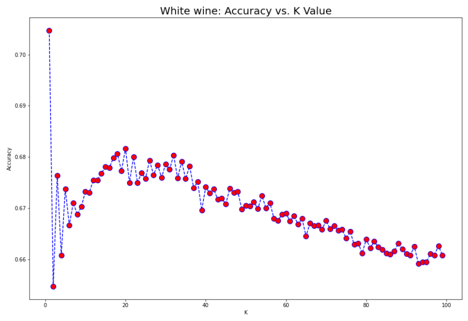
```
def under6(x):
    if x<6:
        return 0
    else:
        return 1

wqw2 = wqw.copy()
wqw2['quality']=wqw2['quality'].apply(under6)
wqr2 = wqr.copy()
wqr2['quality']=wqr2['quality'].apply(under6)
```

3.1 Choice of the value of k

In [100]:

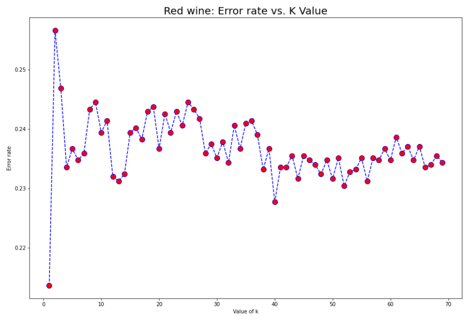
```
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw2.iloc[:,:-1], columns=wqw2.columns[:-1])
y = wqw2['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)

error_rate = [0]*99
acc = [0]*99

for j in range(1,10):
    # Re-split training set to define k-value based on results on subset of training set
    X_train2, X_test2, y_train2, y_test2 = train_test_split(X_train, y_train, test_size=0.20
, shuffle=True)
    # Scaling
    scaler = StandardScaler()
    scaler.fit(X_train2)
    X_train2 = scaler.transform(X_train2)
    X_test2 = scaler.transform(X_test2)
    for i in range(1,100):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train2, y_train2)
        pred = knn.predict(X_test2)
        error_rate[i-1]+=(np.mean(pred != y_test2))
        acc[i-1]+=metrics.accuracy_score(y_test2, pred)

err = [error / 10 for error in error_rate]
accur = [accuracy / 10 for accuracy in acc]

# Error rate plot
plt.figure(figsize=(15,10))
plt.title("White wine: Error rate vs. K Value", fontsize=20)
plt.xlabel("Value of k")
plt.ylabel("Error rate")
plt.plot(range(1,100),err, color='blue', linestyle='dashed', marker='o', markerfacecolor='re
d', markersize=10)
plt.show()
# Accuracy plot
plt.figure(figsize=(15,10))
plt.plot(range(1,100),accur,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('White wine: Accuracy vs. K Value', fontsize=20)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```

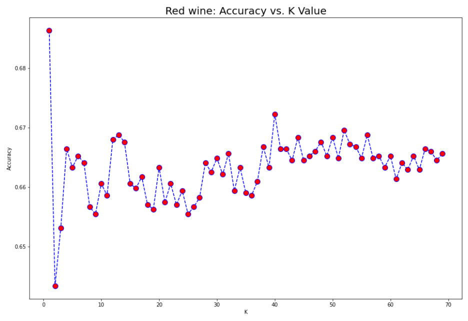




As a graphical ideal value for white wine we select k=20

In [101]:
```
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqr2.iloc[:,:-1], columns=wqr2.columns[:-1])
y = wqr2['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)

error_rate = [0]*69
acc = [0]*69

for j in range(1,10):
    # Re-split training set to define k-value based on results on subset of training set
    X_train2, X_test2, y_train2, y_test2 = train_test_split(X_train, y_train, test_size=0.20
, shuffle=True)
    # Scaling
    scaler = StandardScaler()
    scaler.fit(X_train2)
    X_train2 = scaler.transform(X_train2)
    X_test2 = scaler.transform(X_test2)
    for i in range(1,70):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train2, y_train2)
        pred = knn.predict(X_test2)
        error_rate[i-1]+=(np.mean(pred != y_test2))
        acc[i-1]+=metrics.accuracy_score(y_test2, pred)

err = [error / 10 for error in error_rate]
accur = [accuracy / 10 for accuracy in acc]

# Error rate plot
plt.figure(figsize=(15,10))
plt.title("Red wine: Error rate vs. K Value", fontsize=20)
plt.xlabel("Value of k")
plt.ylabel("Error rate")
plt.plot(range(1,70),err, color='blue', linestyle='dashed', marker='o', markerfacecolor='re
d', markersize=10)
plt.show()
# Accuracy plot
plt.figure(figsize=(15,10))
plt.plot(range(1,70),accur,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Red wine: Accuracy vs. K Value', fontsize=20)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```



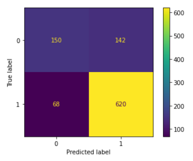


As a graphical ideal value for red wine we select k=40

3.2 Prediction results

In [102]:
```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw2.iloc[:,:-1], columns=wqw2.columns[:-1])
y = wqw2['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# kNN with k=20
classifier = KNeighborsClassifier(n_neighbors=20)
classifier.fit(X_train, y_train)
# Prediction
y_pred = classifier.predict(X_test)
# Confusion matrix
ConfM = confusion_matrix(y_test, y_pred)

display(HTML("<h3 style='text-align: center;'>White wine confusion matrix: </h3><br>"))
cm_display = ConfusionMatrixDisplay(ConfM,display_labels=[0,1]).plot()
plt.show()
display(HTML("<h3 style='text-align: center;'>White wine classification report: </h3><br>"))
print(classification_report(y_test, y_pred, zero_division=0))
```

White wine confusion matrix:



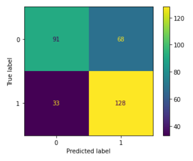
White wine classification report:

```
              precision    recall  f1-score   support

           0       0.69      0.51      0.59       292
           1       0.81      0.90      0.86       688

    accuracy                           0.79       980
   macro avg       0.75      0.71      0.72       980
weighted avg       0.78      0.79      0.78       980
```

We can directly see that performances are better than before in terms of accuracy with a **global accuracy score of 79%.** We can see that our **prediction recall is around 90% on good wines (1) and around 51% on bad ones (0)**. This means that we better classify good wines. This could be explained by the unbalanced repartition in both classes (in the training set). We can see a huge difference in terms of recall and f-1 scores between classes. This issue could - potentially - be solved by adding new observations to the datasets (missing extreme values). Given that good wines support represents 70% of the testing dataset we did quite good on that class (with a f-1 score at 86%). The overall macro f-1 score is quite convincing. It is due to a better repartition between the classes - there are no classes with few observations and bad prediction that reduce the score. To summarize, reducing the number of classes by creating two new classes seems to have improved drastically our performance. We might finally be able to predict wine quality !

In [103]:
```
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqr2.iloc[:,:-1], columns=wqr2.columns[:-1])
y = wqr2['quality']
# Dataset split between training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# kNN with k=40
classifier = KNeighborsClassifier(n_neighbors=40)
classifier.fit(X_train, y_train)
# Prediction
y_pred = classifier.predict(X_test)
# Confusion matrix
ConfM = confusion_matrix(y_test, y_pred)

display(HTML("<h3 style='text-align: center;'>Red wine confusion matrix: </h3><br>"))
cm_display = ConfusionMatrixDisplay(ConfM,display_labels=[0,1]).plot()
plt.show()
display(HTML("<h3 style='text-align: center;'>Red wine classification report: </h3><br>"))
print(classification_report(y_test, y_pred, zero_division=0))
```

Red wine confusion matrix:



Red wine classification report:

```
              precision    recall  f1-score   support

           0       0.73      0.57      0.64       159
           1       0.65      0.80      0.72       161

    accuracy                           0.68       320
   macro avg       0.69      0.68      0.68       320
weighted avg       0.69      0.68      0.68       320
```

We can directly see that performances are better than before in terms of accuracy with a global accuracy score of 68%. We can see that our **prediction recall is around 80% on good wines (1) and around 57% on bad ones (0)**. This means that we better classify good wines. This could be explained by the unbalanced repartition in both classes (in the training set). We can see a huge difference in terms of recall and f-1 scores between classes. This issue could - potentially - be solved by adding new observations to the datasets (missing extreme values). Especially for red wine where the number of observations is really small. Given that good wines support represents 50% of the testing dataset we did quite good on that class (with a f-1 score at 72%). **The overall macro f-1 score is not really convincing**. Thus on red wine the results are not really convincing. Maybe some additional work could be done. We will discuss that on the last section of the document.

4. Qualitative insights using regression methods

In our study we achieved to get decent results on quality prediction. But it comes at a price - interpretability. Indeed despite being able to predict wine quality based on its composition we have no insights on what makes a wine good. To do so we decided to focus on two features: density for white wine and volatile acidity for red wine.

As seen above, for white wine, we observed a negative correlaiton between the quality and the density. For red wine, we observed a negative correlation between the quality and the volatile acidity. Therefore we decided to find the main determinants of those two variables to be able "to reduce them" and improve quality of wines.

The goal for a distributor like us is to provide our suppliers with insights on how to make their wines better.

We decided to use the following methods to obtain coefficients impacting each parameters:

- **Mulitple Linear Regression**
- **Ridge Regression**

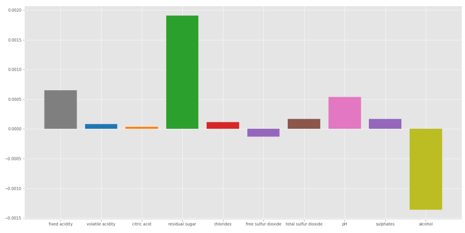
4.1 White wine - Density

Let's first start with a Mulitple Linear Regression:

In [120]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw.loc[:,(wqw.columns != 'density') & (wqw.columns != 'quality') ])
col = X.columns
y = wqw['density']
# Dataset split between training and testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(x_train)
x_train = pd.DataFrame(scaler.transform(x_train), columns = col)
x_test = pd.DataFrame(scaler.transform(x_test), columns = col)
# MULTIPLE LINEAR REGRESSION - WW/Density
lreg = LinearRegression()
lreg.fit(x_train, y_train)

# Generate Prediction on test set
lreg_y_pred = lreg.predict(x_test)

# calculating Mean Squared Error (mse)
mean_squared_error = np.mean((lreg_y_pred - y_test)**2)
print("Mean squared Error on test set : ", mean_squared_error)

# Putting together the coefficient and their corresponding variable names
lreg_coefficient = pd.DataFrame()
lreg_coefficient["Columns"] = x_train.columns
lreg_coefficient['Coefficient Estimate'] = pd.Series(lreg.coef_)
print("Regression coefficient are: ")
print(lreg_coefficient)

# plotting the coefficient score
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:purple', 'tab:olive']

ax.bar(lreg_coefficient["Columns"],
lreg_coefficient['Coefficient Estimate'],
color = color)

plt.style.use('ggplot')
plt.show()
```

```
Mean squared Error on test set :  1.7489050338739502e-07
Regression coefficient are: 
                Columns  Coefficient Estimate
0         fixed acidity              0.000655
1      volatile acidity              0.000081
2           citric acid              0.000035
3        residual sugar              0.001910
4             chlorides              0.000114
5   free sulfur dioxide             -0.000130
6  total sulfur dioxide              0.000172
7                    pH              0.000537
8             sulphates              0.000168
9               alcohol             -0.001361
```



We will also test results for a Ridge regression.
Note on alpha choice for ridge regression: we chose alpha value by cross validation on alpha value testing for 10 value (0.1 to 1 by 0.1 step). Hereafter, the cross-validation process.

In [137]:

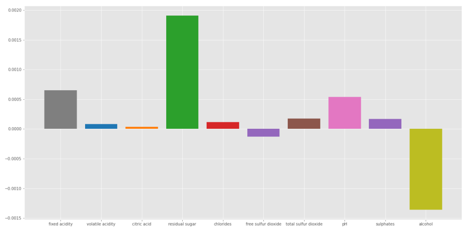
```
# Choice of best alpha by cross validation
from sklearn.linear_model import RidgeCV
# X is the features and y the labels we want to predict
X = pd.DataFrame(wqw.loc[:,(wqw.columns != 'density') & (wqw.columns != 'quality') ])
col = X.columns
y = wqw['density']
# Dataset split between training and testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(x_train)
x_train = pd.DataFrame(scaler.transform(x_train), columns = col)
x_test = pd.DataFrame(scaler.transform(x_test), columns = col)

# Cross validation on alpha values
regr_cv = RidgeCV(alphas=np.linspace(0.1,1,10))
# Fitting
model_cv = regr_cv.fit(X_test, y_test)
display(HTML(f"The ideal value for alpha is thus: {model_cv.alpha_}"))
```

The ideal value for alpha is thus: 1.0

In [138]:

```
# RIDGE REGRESSION - WW/Density
from sklearn.linear_model import Ridge


# Train the model
ridgeR = Ridge(alpha = 1)
ridgeR.fit(x_train, y_train)
y_pred = ridgeR.predict(x_test)

# calculate mean square error
mean_squared_error_ridge = np.mean((y_pred - y_test)**2)
print("The mean squared error on test set is: ", mean_squared_error_ridge)

# get ridge coefficient and print them
ridge_coefficient = pd.DataFrame()
ridge_coefficient["Columns"]= x_train.columns
ridge_coefficient['Coefficient Estimate'] = pd.Series(ridgeR.coef_)
print("Regression coefficient are: ")
print(ridge_coefficient)

# plotting the coefficient score
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:purple', 'tab:olive']

ax.bar(ridge_coefficient["Columns"],
ridge_coefficient['Coefficient Estimate'],
color = color)

plt.style.use('ggplot')
plt.show()
```

```
The mean squared error on test set is:  1.7482473585194396e-07
Regression coefficient are: 
                Columns  Coefficient Estimate
0         fixed acidity              0.000654
1      volatile acidity              0.000081
2           citric acid              0.000035
3        residual sugar              0.001910
4             chlorides              0.000114
5   free sulfur dioxide             -0.000130
6  total sulfur dioxide              0.000172
7                    pH              0.000537
8             sulphates              0.000168
9               alcohol             -0.001361
```



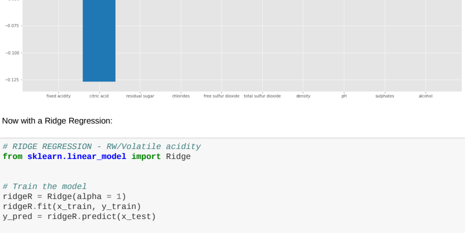
4.2 Red wine - Volatile acidity

Let's start with a Multiple Linear Regression:

In [159]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# X is the features and y the labels we want to predict
X = pd.DataFrame(wqr.loc[:,(wqr.columns != 'volatile acidity') & (wqr.columns != 'quality')
])
col = X.columns
y = wqr['volatile acidity']
# Dataset split between training and testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=None,
shuffle=False)
# Scaling
scaler = StandardScaler()
scaler.fit(x_train)
x_train = pd.DataFrame(scaler.transform(x_train), columns = col)
x_test = pd.DataFrame(scaler.transform(x_test), columns = col)
# MULTIPLE LINEAR REGRESSION - RW/Volatile acidity
lreg = LinearRegression()
lreg.fit(x_train, y_train)

# Generate Prediction on test set
lreg_y_pred = lreg.predict(x_test)

# calculating Mean Squared Error (mse)
mean_squared_error = np.mean((lreg_y_pred - y_test)**2)
print("Mean squared Error on test set : ", mean_squared_error)

# Putting together the coefficient and their corresponding variable names
lreg_coefficient = pd.DataFrame()
lreg_coefficient["Columns"] = x_train.columns
lreg_coefficient['Coefficient Estimate'] = pd.Series(lreg.coef_)
print("Regression coefficient are: ")
print(lreg_coefficient)

# plotting the coefficient score
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:purple', 'tab:olive']

ax.bar(lreg_coefficient["Columns"],
lreg_coefficient['Coefficient Estimate'],
color = color)

plt.style.use('ggplot')
plt.show()
```

```
Mean squared Error on test set :  0.019303121595725666
Regression coefficient are: 
                Columns  Coefficient Estimate
0         fixed acidity              0.012215
1           citric acid             -0.126862
2        residual sugar             -0.010255
3             chlorides              0.036644
4   free sulfur dioxide             -0.030863
5  total sulfur dioxide              0.044138
6               density              0.053949
7                    pH             -0.000571
8             sulphates             -0.031975
9               alcohol              0.025329
```

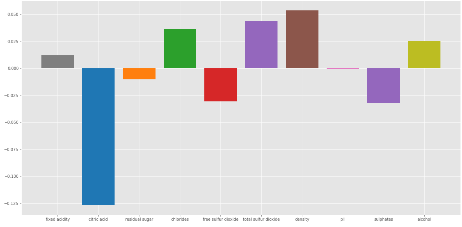


Now with a Ridge Regression:

In [160]:

```
# RIDGE REGRESSION - RW/Volatile acidity
from sklearn.linear_model import Ridge


# Train the model
ridgeR = Ridge(alpha = 1)
ridgeR.fit(x_train, y_train)
y_pred = ridgeR.predict(x_test)

# calculate mean square error
mean_squared_error_ridge = np.mean((y_pred - y_test)**2)
print("The mean squared error on test set is: ", mean_squared_error_ridge)

# get ridge coefficient and print them
ridge_coefficient = pd.DataFrame()
ridge_coefficient["Columns"]= x_train.columns
ridge_coefficient['Coefficient Estimate'] = pd.Series(ridgeR.coef_)
print("Regression coefficient are: ")
print(ridge_coefficient)

# plotting the coefficient score
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:purple', 'tab:olive']

ax.bar(ridge_coefficient["Columns"],
ridge_coefficient['Coefficient Estimate'],
color = color)

plt.style.use('ggplot')
plt.show()
```

```
The mean squared error on test set is:  0.019299706978271592
Regression coefficient are:
                Columns  Coefficient Estimate
0         fixed acidity              0.012193
1           citric acid             -0.126606
2        residual sugar             -0.010165
3             chlorides              0.036566
4   free sulfur dioxide             -0.030763
5  total sulfur dioxide              0.043980
6               density              0.053743
7                    pH             -0.000488
8             sulphates             -0.031925
9               alcohol              0.025105
```



We can observe that both method gave the same coefficients more or less, so both methods seem accurate. Because both parameters have a negative correlation with wine quality, we wish to minimize them. Nevertheless regarding red wine, due to a small amount of observations, cross validation for the choice of alpha in the ridge regression didn't work: **indicating that multiple linear regression is more robust in this case.**
For white wine, we obtained significant positive coefficients for residual sugar and negative significant coefficients for alcohol. This suggests that in order to minimize density, **one should reduce residual sugar in the wine - which seems coherent. On the other hand increasing alcohol concentration would decrease wine density and therefore positively impact wine quality**. Those two observations would be our recommandations for our suppliers concerning white wine.
For red wine, we obtained really significant negative coefficients for citric acid - which seems logic once again. **But the mean square error is relatively high on this dataset indicating that the regression didn't work so well. Thus we cannot make any recommandation to our suppliers concerning red wine.**

5. Further possible developments

In order to improve our work, we came up with the following ideas:

- Increasing the total number of observations - especially in the red wine dataset
- Rebalancing data amongst classes to obtain a more equal repartition by oversampling or generating synthetic data
- Enriching our confusion matrix analysis with ROC curves to see graphically model performance for each class
- Enrich our datasets with missing features such as the grape type that could be a determinant feature in terms of taste
- etc.

Conclusion

To conclude, during this study we uncovered the fact that wine quality prediction was a difficult task based on the datasets we were provided. The only way for us to obtain some acceptable prediction scores has been to reduce drastically the number of classes in the dataset to discriminate values below 6 and equal or above. As far as classification and prediction techniques are concerned kNN classification has performed significantly better than QDA. Nonetheless we couldn't overcome the fact that the data was unbalanced between classes, and failed to predict correctly red wine quality.
This study allowed us to see how important good data is a must for possible interpretation. Thus we proposed below some ways to make the dataset, maybe, more meaningful.
In addition to prediction, we tried to give some hints on how to produce better wine based on regression coefficients. This ended up not being that meaningful because nothing striking appeared.