Red wine analysis ¶

Python · wine quality selection

```
This kernel is based on a tutorial in EliteDataScience. In that tutorial, a Random Forest
Regressor was used and involved standardizing the data and hyperparameter tuning. Here, I
am using a Random Forest Classifier instead, to do a binary classification of good wine
and not-so-good wine.
```

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from bokeh.plotting import figure, output_file, show
from bokeh.layouts import row
from bokeh.io import output_notebook
import statsmodels.api as sm
import statsmodels.formula.api as smf
from patsy import dmatrices
import sklearn
import sklearn.metrics
from sklearn import ensemble
from sklearn import linear_model

import warnings
warnings.filterwarnings('ignore')
output_notebook()
%matplotlib inline
```

(https://bokeh.org)BokehJS 2.4.1 successfully loaded.

In [2]:

```
pip install patsy
```

```
Requirement already satisfied: patsy in c:\programdata\anaconda3\lib\site-pa
ckages (0.5.2)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-pack
ages (from patsy) (1.16.0)
Requirement already satisfied: numpy>=1.4 in c:\programdata\anaconda3\lib\si
te-packages (from patsy) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

Load data

In [3]:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-r
wine = pd.read_csv(url)
```

In [4]:

```
wine.head(n=5)
```

Out[4]:

	fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";"quality"
0	7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5
1	7.8;0.88;0;2.6;0.098;25;67;0.9968;3.2;0.68;9.8;5
2	7.8;0.76;0.04;2.3;0.092;15;54;0.997;3.26;0.65;...
3	11.2;0.28;0.56;1.9;0.075;17;60;0.998;3.16;0.58...
4	7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5

In [5]:

```
wine = pd.read_csv(url, sep=";")
wine.head(n=5)
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

Exploring the Red Wine dataset:

print("Shape of Red Wine dataset: {s}").format(s = wine.shape) print("Column headers/names: {s}").format(s = list(wine))

Shape of Red Wine dataset: (1599, 12) Column headers/names: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality']

In [6]:

```
# Now, let's check the information about different variables/column from the dataset:
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Exploring the Red Wine dataset:

```
All columns has the same number of data points, so it looks like there are no missing
data.
Are there duplicated rows in the data?
```

In [9]:

```
wine.isnull().sum()
```

Out[9]:

```
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
dtype: int64
```

In [11]:

```
wine.rename(columns={'fixed acidity': 'fixed_acidity','citric acid':'citric_acid','volatile
wine.head(n=5)
```

Out[11]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_
0	7.4	0.70	0.00	1.9	0.076	11.0	
1	7.8	0.88	0.00	2.6	0.098	25.0	
2	7.8	0.76	0.04	2.3	0.092	15.0	
3	11.2	0.28	0.56	1.9	0.075	17.0	
4	7.4	0.70	0.00	1.9	0.076	11.0	

In [12]:

```
# Now, let's check the information about different variables/column from the dataset:
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         1599 non-null   float64
 1   volatile_acidity      1599 non-null   float64
 2   citric_acid           1599 non-null   float64
 3   residual_sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free_sulfur_dioxide   1599 non-null   float64
 6   total_sulfur_dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [13]:

```
wine['quality'].unique()
```

Out[13]:

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [14]:

```
wine.quality.value_counts().sort_index()
```

Out[14]:

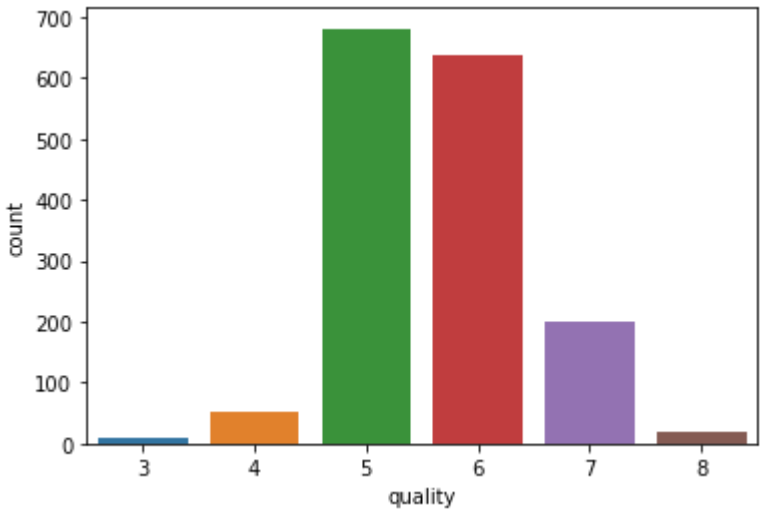
```
3     10
4     53
5    681
6    638
7    199
8     18
Name: quality, dtype: int64
```

In [15]:

```
sns.countplot(x='quality', data=wine)
```

Out[15]:

```
<AxesSubplot:xlabel='quality', ylabel='count'>
```




```
The above distribution shows the range for response variable (quality) is between 3 to 8.
Let's create a new discreet, categorical response variable/feature ('rating') from
existing 'quality' variable.
i.e. bad: 1-4
      average: 5-6
      good: 7-10
```

In [16]:

```
conditions = [
    (wine['quality'] >= 7),
    (wine['quality'] <= 4)
]
rating = ['good', 'bad']
wine['rating'] = np.select(conditions, rating, default='average')
wine.rating.value_counts()
```

Out[16]:

```
average    1319
good        217
bad          63
Name: rating, dtype: int64
```

In [17]:

```
wine.groupby('rating').mean()
```

Out[17]:

rating	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide
average	8.254284	0.538560	0.258264	2.503867	0.088973	16.368461
bad	7.871429	0.724206	0.173651	2.684921	0.095730	12.063492
good	8.847005	0.405530	0.376498	2.708756	0.075912	13.981567

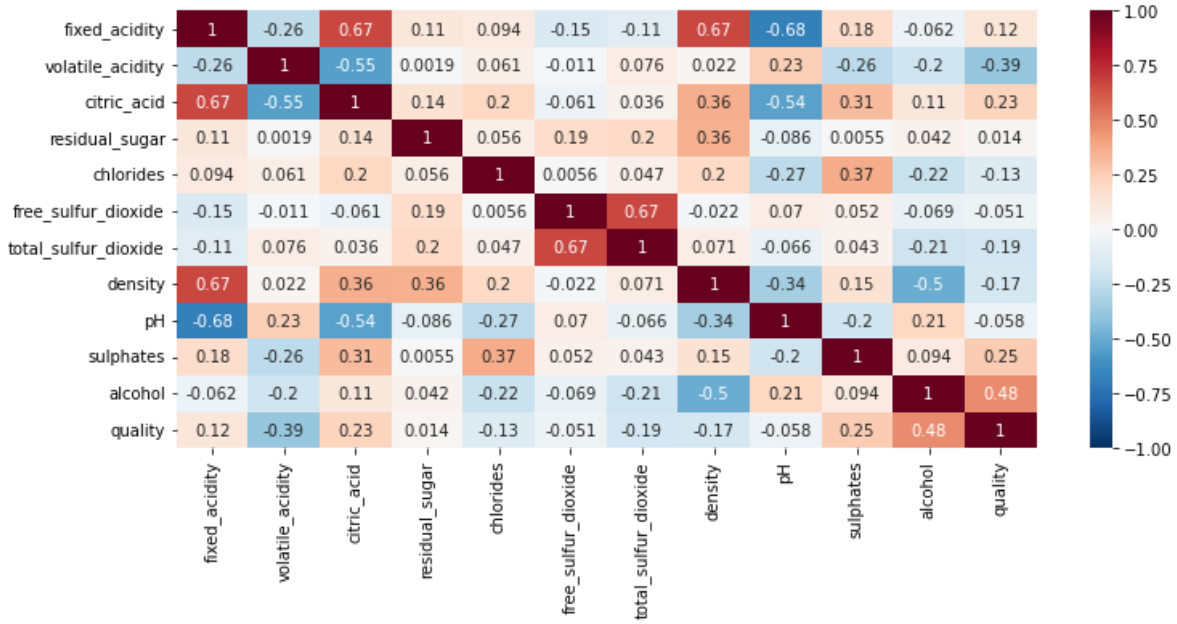
Corelation between features/variables:
Let's check the corelation between the target variable and predictor variables,

In [18]:

```
correlation = wine.corr()
plt.figure(figsize=(12, 5))
sns.heatmap(correlation, annot=True, linewidths=0, vmin=-1, cmap="RdBu_r")
```

Out[18]:

<AxesSubplot:>



In [19]:

```
correlation['quality'].sort_values(ascending=False)
```

Out[19]:

```
quality                1.000000
alcohol                0.476166
sulphates              0.251397
citric_acid            0.226373
fixed_acidity          0.124052
residual_sugar         0.013732
free_sulfur_dioxide   -0.050656
pH                    -0.057731
chlorides             -0.128907
density               -0.174919
total_sulfur_dioxide  -0.185100
volatile_acidity      -0.390558
Name: quality, dtype: float64
```

```
We can observe that, the 'alcohol, sulphates, citric_acid & fixed_acidity' have maximum
corelation with response variable 'quality'.
This means that, they need to be further analysed for detailed pattern and corelation
exploration. Hence, we will use only these 4 variables in our future analysis.
```

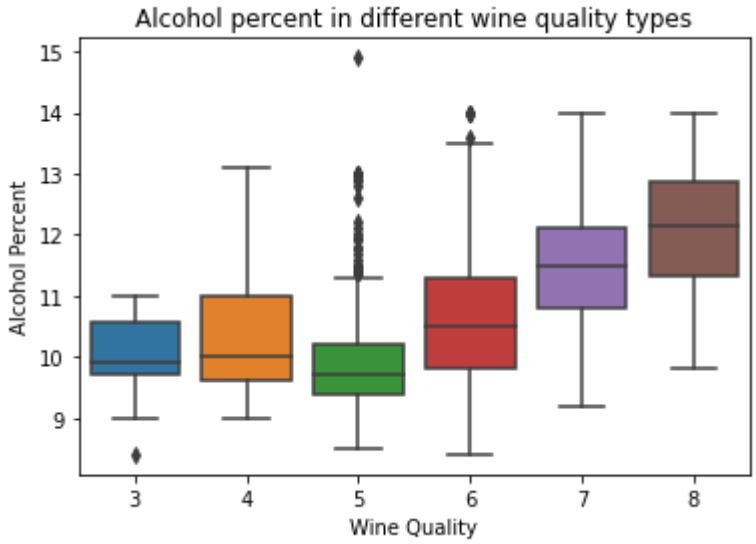
Analysis of alcohol percentage with wine quality:

In [20]:

```
bx = sns.boxplot(x="quality", y='alcohol', data = wine)
bx.set(xlabel='Wine Quality', ylabel='Alcohol Percent', title='Alcohol percent in different
```

Out[20]:

```
[Text(0.5, 0, 'Wine Quality'),
 Text(0, 0.5, 'Alcohol Percent'),
 Text(0.5, 1.0, 'Alcohol percent in different wine quality types')]
```



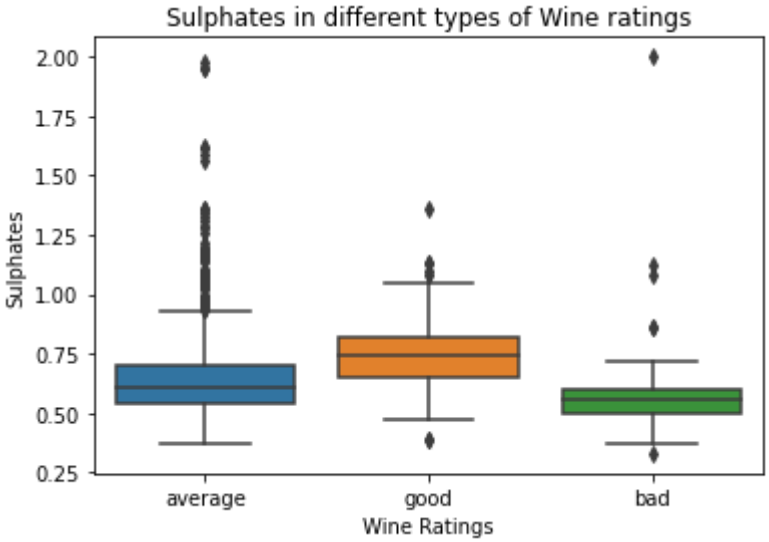
Analysis of sulphates & wine ratings:

In [21]:

```
bx = sns.boxplot(x="rating", y='sulphates', data = wine)
bx.set(xlabel='Wine Ratings', ylabel='Sulphates', title='Sulphates in different types of Wi
```

Out[21]:

```
[Text(0.5, 0, 'Wine Ratings'),
 Text(0, 0.5, 'Sulphates'),
 Text(0.5, 1.0, 'Sulphates in different types of Wine ratings')]
```



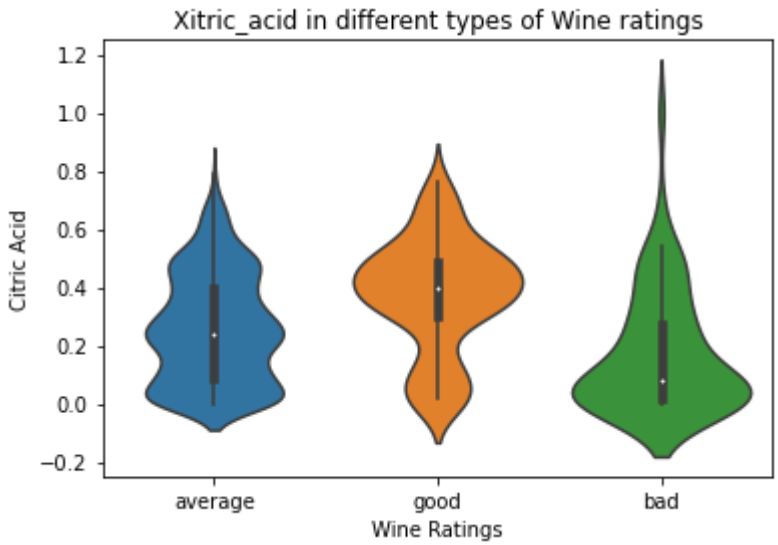
Analysis of Citric Acid & wine ratings:

In [22]:

```
bx = sns.violinplot(x="rating", y='citric_acid', data = wine)
bx.set(xlabel='Wine Ratings', ylabel='Citric Acid', title='Xitric_acid in different types o
```

Out[22]:

```
[Text(0.5, 0, 'Wine Ratings'),
 Text(0, 0.5, 'Citric Acid'),
 Text(0.5, 1.0, 'Xitric_acid in different types of Wine ratings')]
```



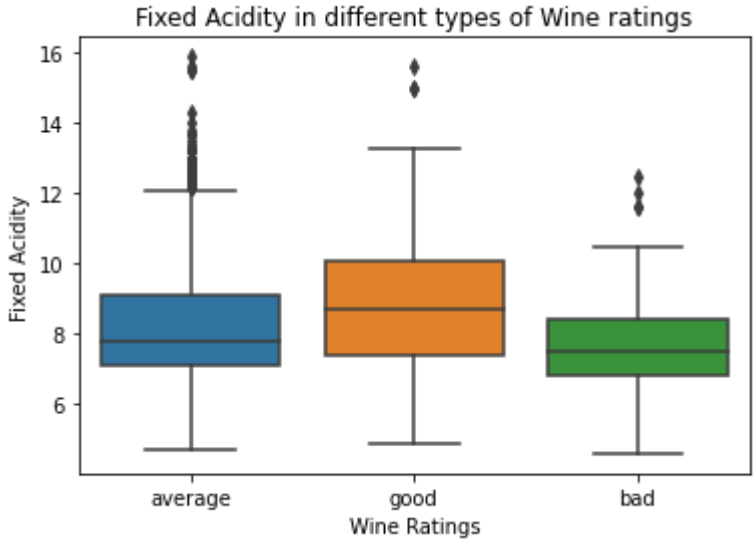
Analysis of fixed acidity & wine ratings:

In [23]:

```
bx = sns.boxplot(x="rating", y='fixed_acidity', data = wine)
bx.set(xlabel='Wine Ratings', ylabel='Fixed Acidity', title='Fixed Acidity in different typ
```

Out[23]:

```
[Text(0.5, 0, 'Wine Ratings'),
 Text(0, 0.5, 'Fixed Acidity'),
 Text(0.5, 1.0, 'Fixed Acidity in different types of Wine ratings')]
```



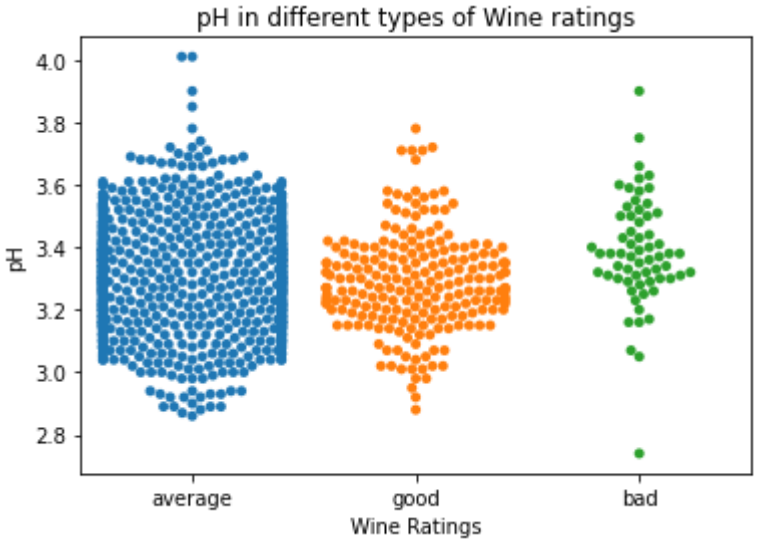
Analysis of pH & wine ratings:

In [24]:

```
bx = sns.swarmplot(x="rating", y="pH", data = wine);
bx.set(xlabel='Wine Ratings', ylabel='pH', title='pH in different types of Wine ratings')
```

Out[24]:

```
[Text(0.5, 0, 'Wine Ratings'),
 Text(0, 0.5, 'pH'),
 Text(0.5, 1.0, 'pH in different types of Wine ratings')]
```



Linear Regression:

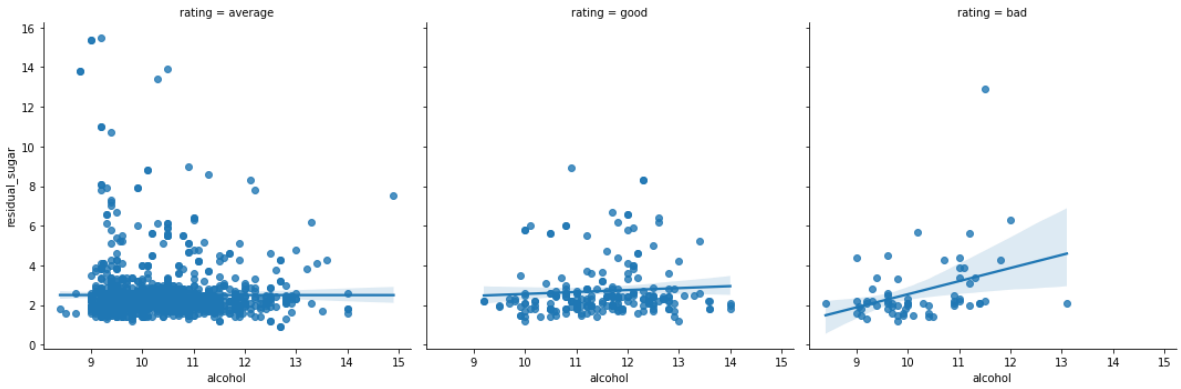
Below graphs for different quality ratings shows a linear regression between residual_sugar & alcohol in red wine,

In [25]:

```
sns.lmplot(x = "alcohol", y = "residual_sugar", col = "rating", data = wine)
```

Out[25]:

```
<seaborn.axisgrid.FacetGrid at 0x13cea6fa130>
```



The linear regression plots above for different wine quality ratings (bad, average & good) shows the regression between alcohol and residual sugar content of the red wine.

We can observe from the trendline that, for good and average wine types the residual sugar content remains almost constant irrespective of alcohol content value. Whereas for bad quality wine, the residual sugar content increases gradually with the increase in alcohol content.
This analysis can help in manufacturing the good quality wine with continuous monitoring and contrilling the alcohol and residual sugar content of the red wine.

In [27]:

```
y,X = dmatrices('quality ~ alcohol', data=wine, return_type='dataframe')
print("X:", type(X))
print(X.columns)
model=smf.OLS(y, X)
result=model.fit()
result.summary()
```

```
X: <class 'pandas.core.frame.DataFrame'>
Index(['Intercept', 'alcohol'], dtype='object')

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17988/511796430.py in <module>
      2 print("X:", type(X))
      3 print(X.columns)
----> 4 model=smf.OLS(y, X)
      5 result=model.fit()
      6 result.summary()

AttributeError: module 'statsmodels.formula.api' has no attribute 'OLS'
```

In [28]:

```
import statsmodels.api as sm
import statsmodels.regression.linear_model as sm
import statsmodels.api as sm
```

In [29]:

```
model = smf.OLS.from_formula('quality ~ alcohol', data = wine)
results = model.fit()
print(results.params)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17988/65294927.py in <module>
----> 1 model = smf.OLS.from_formula('quality ~ alcohol', data = wine)
      2 results = model.fit()
      3 print(results.params)

AttributeError: module 'statsmodels.formula.api' has no attribute 'OLS'
```

In [30]:

```
pip install patsy
```

```
Requirement already satisfied: patsy in c:\programdata\anaconda3\lib\site-pa
ckages (0.5.2)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-pack
ages (from patsy) (1.16.0)
Requirement already satisfied: numpy>=1.4 in c:\programdata\anaconda3\lib\si
te-packages (from patsy) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

Classification

Classification using Statsmodel:

We will use statsmodel for this logistic regression analysis of predicting good wine quality (>4). Let's create a new categorical variable/column (rate_code) with two possible values (good = 1 & bad = 0).

In [31]:

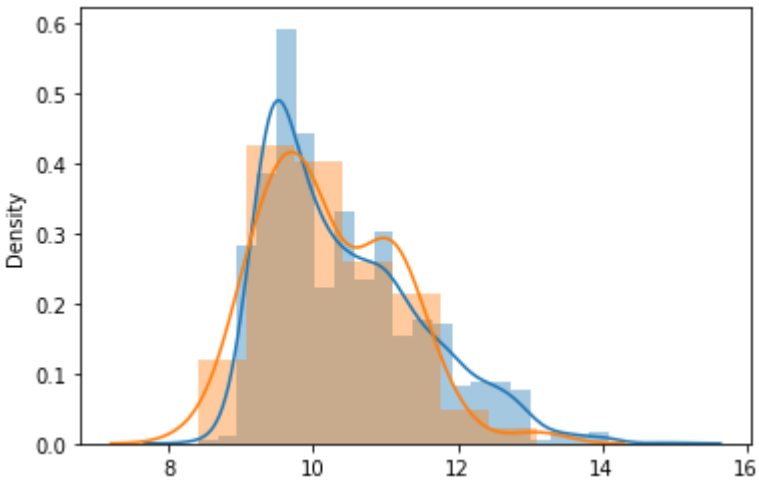
```
wine['rate_code'] = (wine['quality'] > 4).astype(np.float32)
```

In [32]:

```
y, X = dmatrices('rate_code ~ alcohol', data = wine)
sns.distplot(X[y[:,0] > 0, 1])
sns.distplot(X[y[:,0] == 0, 1])
```

Out[32]:

```
<AxesSubplot:ylabel='Density'>
```



In [33]:

```
model = smf.Logit(y, X)
result = model.fit()
result.summary2()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17988/3913109600.py in <module>
----> 1 model = smf.Logit(y, X)
      2 result = model.fit()
      3 result.summary2()

AttributeError: module 'statsmodels.formula.api' has no attribute 'Logit'
```

In [34]:

```
yhat = result.predict(X)
sns.distplot(yhat[y[:,0] > 0])
sns.distplot(yhat[y[:,0] == 0])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17988/2415631086.py in <module>
----> 1 yhat = result.predict(X)
      2 sns.distplot(yhat[y[:,0] > 0])
      3 sns.distplot(yhat[y[:,0] == 0])

NameError: name 'result' is not defined
```

In [37]:

```
yhat = result.predict(X) > 0.955
print(sklearn.metrics.classification_report(y, yhat))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17988/1615879700.py in <module>
----> 1 yhat = result.predict(X) > 0.955
      2 print(sklearn.metrics.classification_report(y, yhat))

NameError: name 'result' is not defined
```

Classification using Sklearn's LogisticRegression:

In [36]:

```
model = sklearn.linear_model.LogisticRegression()
y,X = dmatrices('rate_code ~ alcohol + sulphates + citric_acid + fixed_acidity', data = win
model.fit(X, y)
yhat = model.predict(X)
print(sklearn.metrics.classification_report(y, yhat))
```

```
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        63
         1.0       0.96      1.00      0.98      1536

    accuracy                           0.96      1599
   macro avg       0.48      0.50      0.49      1599
weighted avg       0.92      0.96      0.94      1599
```

```
The accuracy matrix for sklearn's linear regression model for red wine quality prediction
shows the overall 92% precision which is similar to previous statsmodel's average
precision.
Also the precision for good wine (1) prediction is almost 96%.
But the precision is almost 0% for the bad type of wine (0) with sklearn's linear
regression model. Which is not a good sign for the analysis.
```

Classification using Sklearn's RandomForestClassifier:

In [38]:

```
y, X = dmatrices('rate_code ~ alcohol', data = wine)
model = sklearn.ensemble.RandomForestClassifier()
model.fit(X, y)
yhat = model.predict(X)
print(sklearn.metrics.classification_report(y, yhat))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.03      0.06        63
         1.0       0.96      1.00      0.98      1536

    accuracy                           0.96      1599
   macro avg       0.98      0.52      0.52      1599
weighted avg       0.96      0.96      0.94      1599
```

```
Here, with the accuracy matrix for sklearn's random forest classifier model for the
prediction of red wine quality, we can observe that the values have been improved
significantly.
The precision for the prediction of bad quality wine (0) is almost 100% where as the
precision for prediction of good quality wine (1) is approximately 96%.
This sklearn's random forest classifier model also has the overall precision around 96%,
which is far better than the previous two models (i.e. statsmodel and sklearn's linear
regression model)
```

2 . Abalone case study

In [47]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from bokeh.plotting import figure, output_file, show
from bokeh.layouts import row
from bokeh.io import output_notebook
import statsmodels.api as sm
import statsmodels.formula.api as smf
from patsy import dmatrices
import sklearn
import sklearn.metrics
from sklearn import ensemble
from sklearn import linear_model

import warnings
warnings.filterwarnings('ignore')
output_notebook()
%matplotlib inline
```

(https://bokeh.org)BokehJS 2.4.1 successfully loaded.

In [48]:

```
his Python 3 environment comes with many helpful analytics libraries installed
t is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
or example, here's several helpful packages to load

ort numpy as np # linear algebra
ort pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

nput data files are available in the read-only "../input/" directory
or example, running this (by clicking run or pressing Shift+Enter) will list all files under

ort os
 dirname, _, filenames in os.walk('/kaggle/input'):
 for filename in filenames:
     print(os.path.join(dirname, filename))

ou can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as o
ou can also write temporary files to /kaggle/temp/, but they won't be saved outside of the c
```

In [86]:

```
# Pandas : librairie de manipulation de données
# NumPy : librairie de calcul scientifique
# MatPlotLib : librairie de visualisation et graphiques
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score
```

In [55]:

```
od.read_csv("https://raw.githubusercontent.com/nishitpatel01/predicting-age-of-abalone-using
```

Out[55]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_we
0	M	70	53	18	45.1	19.9	9.7	
1	F	106	84	27	135.4	51.3	28.3	
2	M	88	73	25	103.2	43.1	22.8	
3	I	66	51	16	41.0	17.9	7.9	
4	I	85	60	19	70.3	28.2	15.5	
...	...	...	...	...	...	...	...	
4171	F	113	90	33	177.4	74.0	47.8	
4172	M	118	88	27	193.2	87.8	42.9	
4173	M	120	95	41	235.2	105.1	57.5	
4174	F	125	97	30	218.9	106.2	52.2	
4175	M	142	111	39	389.7	189.1	75.3	

4176 rows × 9 columns

In [56]:

```
df.head(10).T
```

Out[56]:

	0	1	2	3	4	5	6	7	8	9
Sex	M	F	M	I	I	F	F	M	F	F
Length	70	106	88	66	85	106	109	95	110	105
Diameter	53	84	73	51	60	83	85	74	88	76
Height	18	27	25	16	19	30	25	25	30	28
Whole_weight	45.1	135.4	103.2	41.0	70.3	155.5	153.6	101.9	178.9	121.3
Shucked_weight	19.9	51.3	43.1	17.9	28.2	47.4	58.8	43.3	62.9	38.8
Viscera_weight	9.7	28.3	22.8	7.9	15.5	28.3	29.9	22.5	30.2	29.5
Shell_weight	14.0	42.0	31.0	11.0	24.0	66.0	52.0	33.0	64.0	42.0
Rings	7	9	10	7	8	20	16	9	19	14

In [57]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4176 entries, 0 to 4175
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sex             4176 non-null   object 
 1   Length          4176 non-null   int64  
 2   Diameter        4176 non-null   int64  
 3   Height          4176 non-null   int64  
 4   Whole_weight    4176 non-null   float64
 5   Shucked_weight  4176 non-null   float64
 6   Viscera_weight  4176 non-null   float64
 7   Shell_weight    4176 non-null   float64
 8   Rings           4176 non-null   int64  
dtypes: float64(4), int64(4), object(1)
memory usage: 293.8+ KB
```

In [58]:

```
df["Sex"] = df["Sex"].map({"F":0, "M":1, "I":2})
```

In [59]:

```
df.head(10).T
```

Out[59]:

	0	1	2	3	4	5	6	7	8	9
Sex	1.0	0.0	1.0	2.0	2.0	0.0	0.0	1.0	0.0	0.0
Length	70.0	106.0	88.0	66.0	85.0	106.0	109.0	95.0	110.0	105.0
Diameter	53.0	84.0	73.0	51.0	60.0	83.0	85.0	74.0	88.0	76.0
Height	18.0	27.0	25.0	16.0	19.0	30.0	25.0	25.0	30.0	28.0
Whole_weight	45.1	135.4	103.2	41.0	70.3	155.5	153.6	101.9	178.9	121.3
Shucked_weight	19.9	51.3	43.1	17.9	28.2	47.4	58.8	43.3	62.9	38.8
Viscera_weight	9.7	28.3	22.8	7.9	15.5	28.3	29.9	22.5	30.2	29.5
Shell_weight	14.0	42.0	31.0	11.0	24.0	66.0	52.0	33.0	64.0	42.0
Rings	7.0	9.0	10.0	7.0	8.0	20.0	16.0	9.0	19.0	14.0

In [60]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4176 entries, 0 to 4175
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sex             4176 non-null   int64  
 1   Length          4176 non-null   int64  
 2   Diameter        4176 non-null   int64  
 3   Height          4176 non-null   int64  
 4   Whole_weight    4176 non-null   float64
 5   Shucked_weight  4176 non-null   float64
 6   Viscera_weight  4176 non-null   float64
 7   Shell_weight    4176 non-null   float64
 8   Rings           4176 non-null   int64  
dtypes: float64(4), int64(5)
memory usage: 293.8 KB
```

In [61]:

```
tabcorr = df.corr()
```

In [62]:

```
tabcorr
```

Out[62]:

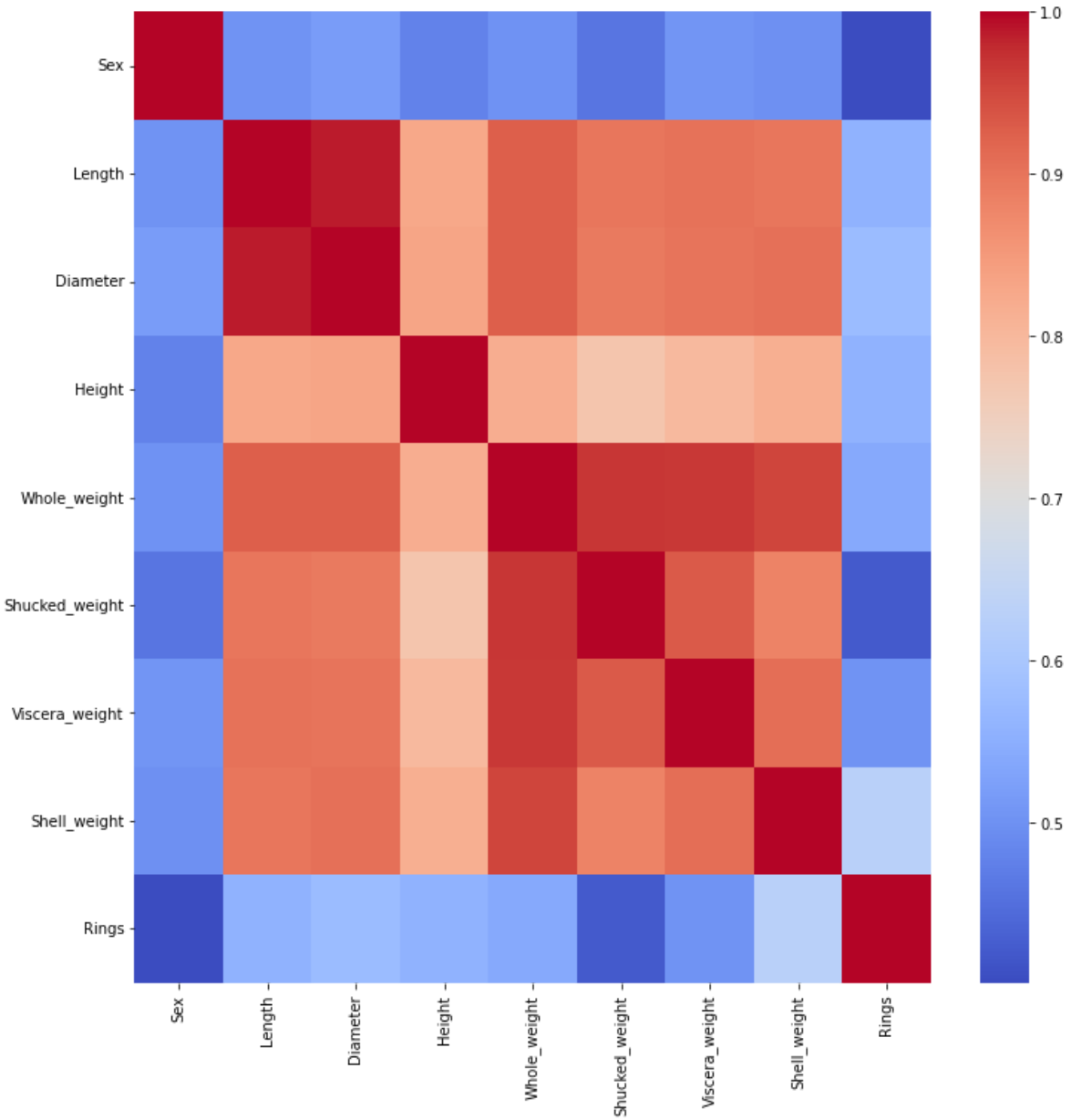
	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Vis
Sex	1.000000	-0.503719	-0.516463	-0.477917	-0.501538	-0.459753	
Length	-0.503719	1.000000	0.986813	0.827552	0.925255	0.897905	
Diameter	-0.516463	0.986813	1.000000	0.833705	0.925452	0.893159	
Height	-0.477917	0.827552	0.833705	1.000000	0.819209	0.774957	
Whole_weight	-0.501538	0.925255	0.925452	0.819209	1.000000	0.969403	
Shucked_weight	-0.459753	0.897905	0.893159	0.774957	0.969403	1.000000	
Viscera_weight	-0.505726	0.903010	0.899726	0.798293	0.966372	0.931956	
Shell_weight	-0.499129	0.897697	0.905328	0.817326	0.955351	0.882606	
Rings	-0.401560	0.557123	0.575005	0.558109	0.540818	0.421256	

In [63]:

```
plt.figure(figsize=(12,12))
sns.heatmap(abs(tabcorr), cmap="coolwarm")
```

Out[63]:

<AxesSubplot:>

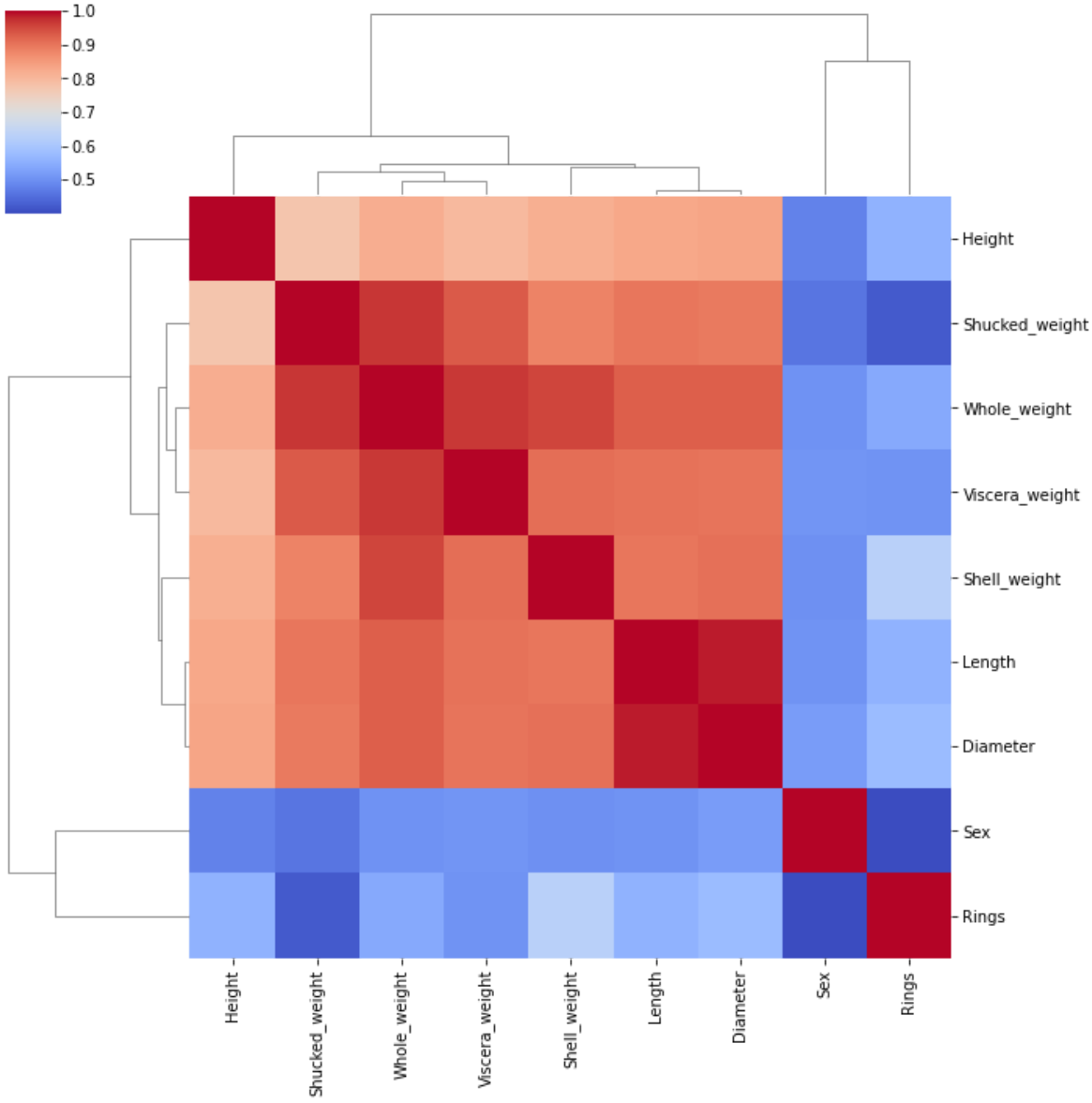


In [64]:

```
sns.clustermap(abs(tabcorr), cmap="coolwarm")
```

Out[64]:

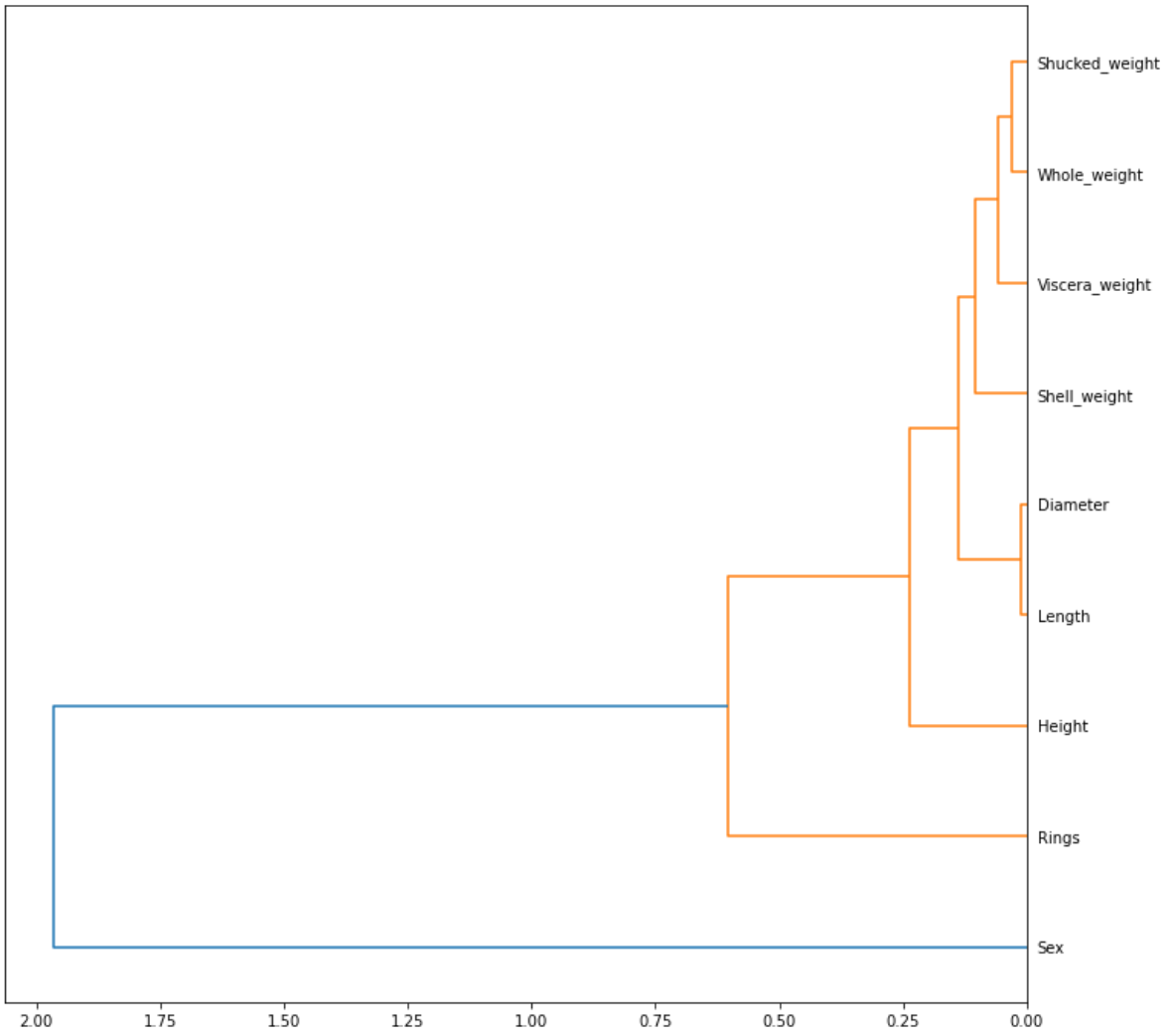
```
<seaborn.matrix.ClusterGrid at 0x13cead730d0>
```



In [65]:

```
from scipy.cluster import hierarchy as hc

corr = 1 - df.corr()
corr_condensed = hc.distance.squareform(corr)
link = hc.linkage(corr_condensed, method='ward')
plt.figure(figsize=(12,12))
den = hc.dendrogram(link, labels=df.columns, orientation='left', leaf_font_size=10)
```



In [66]:

```
correlations = tabcorr.Rings
print(correlations)
```

```
Sex              -0.401560
Length            0.557123
Diameter          0.575005
Height            0.558109
Whole_weight      0.540818
Shucked_weight    0.421256
Viscera_weight    0.504274
Shell_weight      0.628031
Rings             1.000000
Name: Rings, dtype: float64
```

In [67]:

```
correlations = correlations.drop(['Rings'],axis=0)
```

In [68]:

```
print(abs(correlations).sort_values(ascending=False))
```

```
Shell_weight      0.628031
Diameter          0.575005
Height            0.558109
Length            0.557123
Whole_weight      0.540818
Viscera_weight    0.504274
Shucked_weight    0.421256
Sex               0.401560
Name: Rings, dtype: float64
```

In [69]:

```
df1 = df.drop(['Sex'], axis=1)
```

In [70]:

```
X = df1.drop(['Rings'], axis=1)
y = df1.Rings
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)
```

In [71]:

```
from sklearn.linear_model import LinearRegression
```

In [72]:

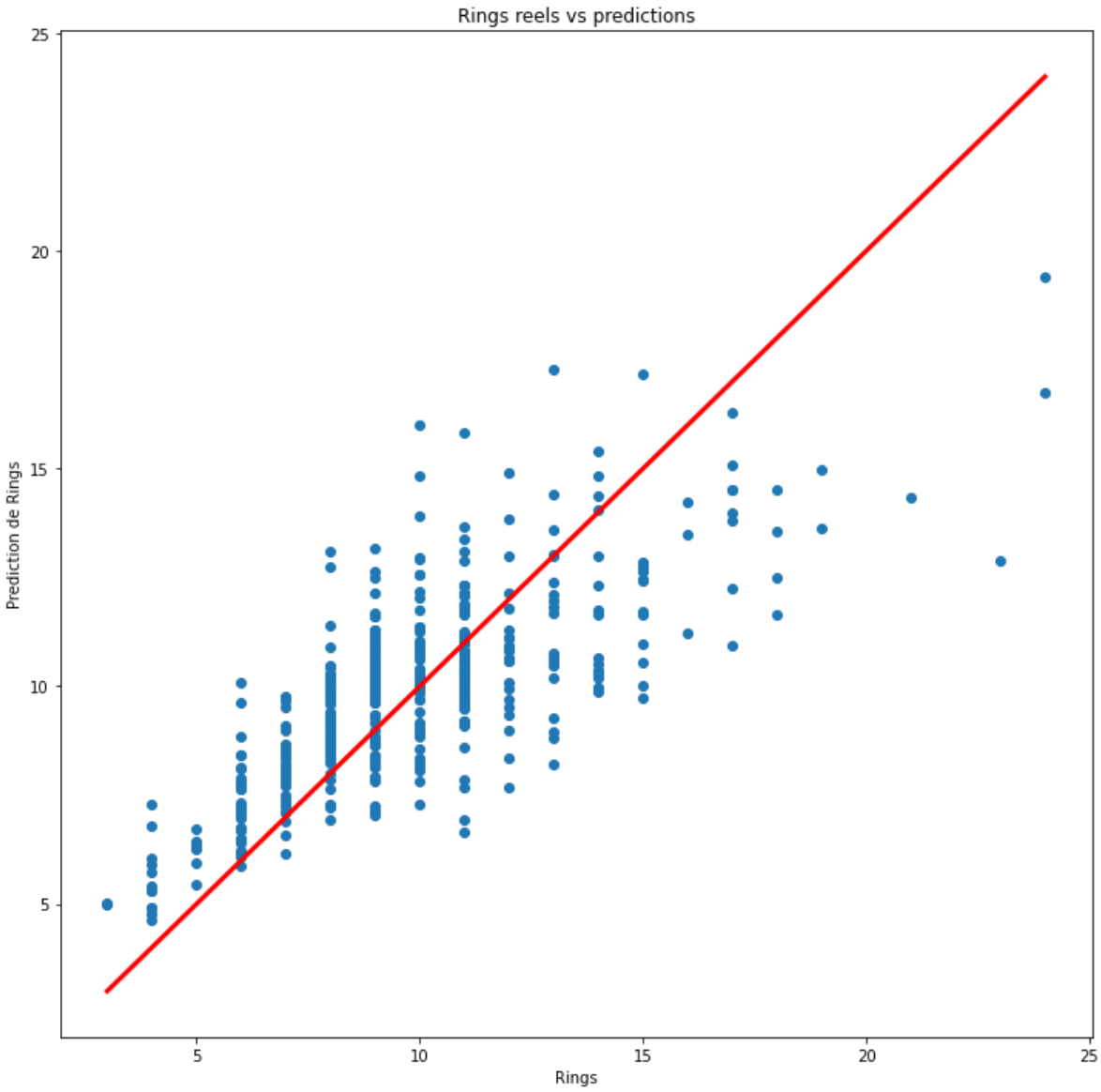
```
lm = LinearRegression()
lm.fit(X_train, y_train)           # apprentissage
y_pred = lm.predict(X_test)        # prédiction sur l'ensemble de test
```

In [73]:

```
plt.figure(figsize=(12,12))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], color='red', linewidth=3)
plt.xlabel("Rings")
plt.ylabel("Prediction de Rings")
plt.title("Rings reels vs predictions")
```

Out[73]:

```
Text(0.5, 1.0, 'Rings reels vs predictions')
```

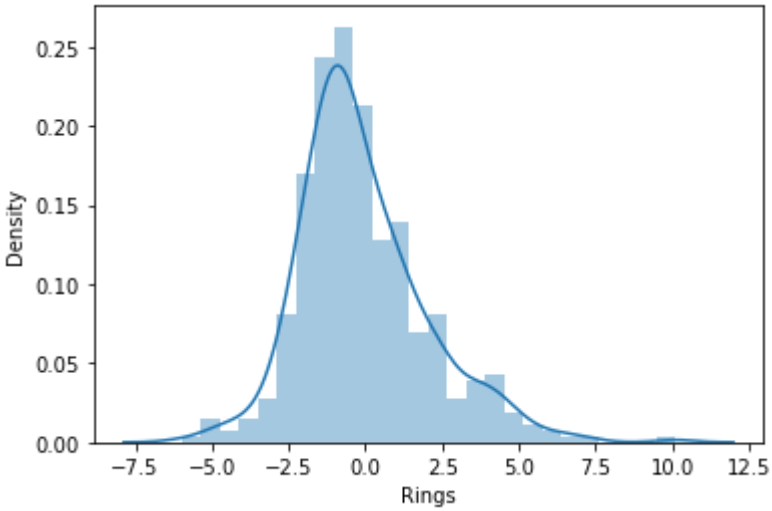


In [74]:

```
sns.distplot(y_test-y_pred)
```

Out[74]:

```
<AxesSubplot:xlabel='Rings', ylabel='Density'>
```



In [75]:

```
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
2.101169820213064
```

In [76]:

```
scoreR2 = r2_score(y_test, y_pred)
print(scoreR2)
```

```
0.5723398649661379
```

In [77]:

```
lm.score(X_test,y_test)
```

Out[77]:

```
0.5723398649661379
```

In [78]:

```
X = df.drop(['Rings'], axis=1)
y = df.Rings
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)
```

In [79]:

```
from sklearn import ensemble
rf = ensemble.RandomForestRegressor()
rf.fit(X_train, y_train)
y_rf = rf.predict(X_test)
print(rf.score(X_test,y_test))
```

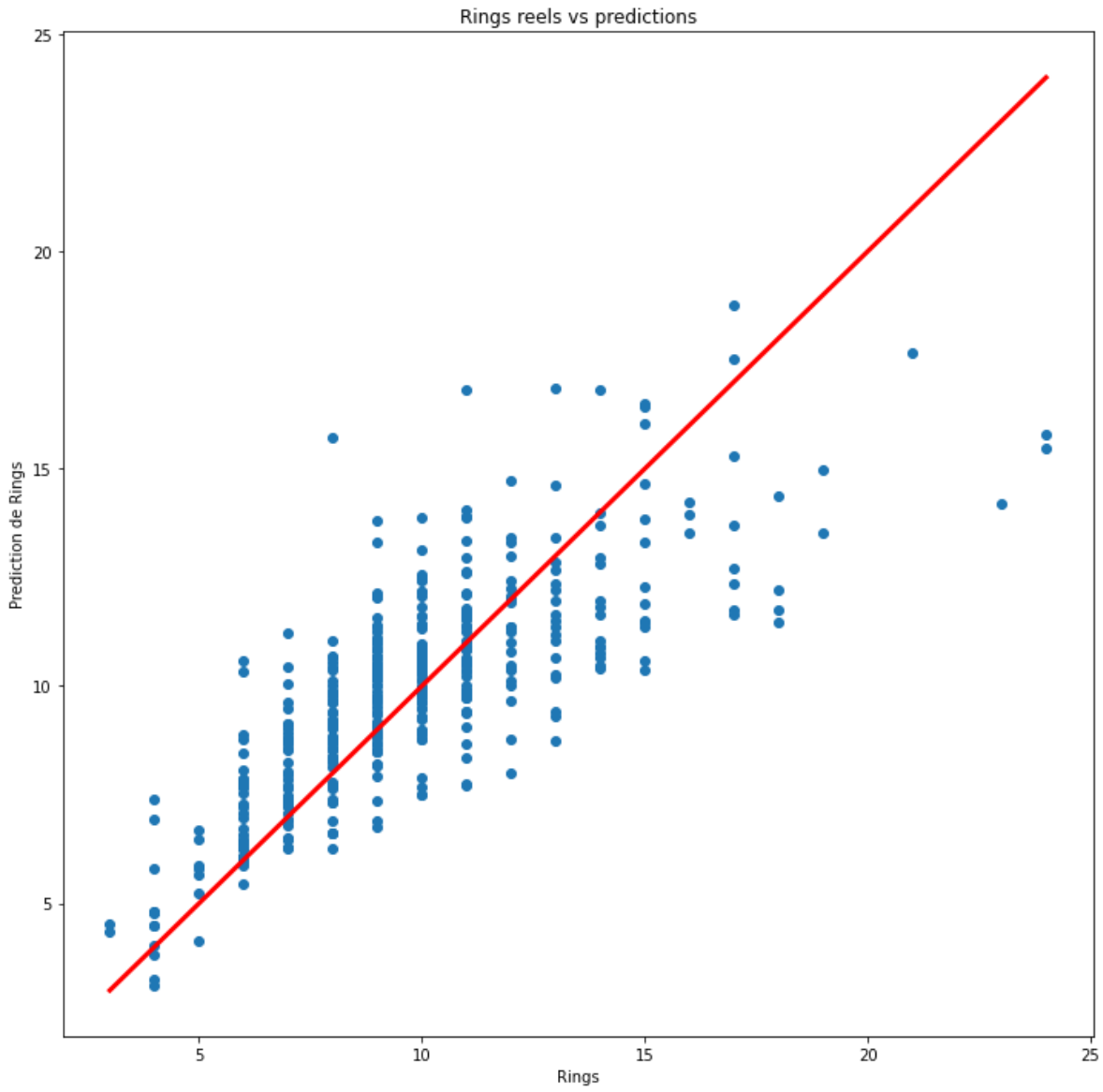
```
0.5958075991726122
```

In [80]:

```
plt.figure(figsize=(12,12))
plt.scatter(y_test, y_rf)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], color='red', linewidth=3)
plt.xlabel("Rings")
plt.ylabel("Prediction de Rings")
plt.title("Rings reels vs predictions")
```

Out[80]:

```
Text(0.5, 1.0, 'Rings reels vs predictions')
```

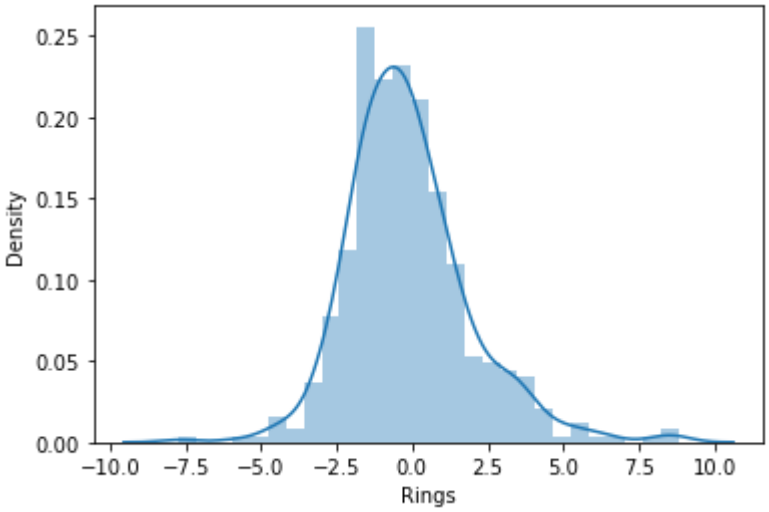


In [81]:

```
sns.distplot(y_test-y_rf)
```

Out[81]:

```
<AxesSubplot:xlabel='Rings', ylabel='Density'>
```



In [82]:

```
print(np.sqrt(mean_squared_error(y_test, y_rf)))
```

```
2.0427058937647025
```

In [83]:

```
rf.score(X_test,y_test)
```

Out[83]:

```
0.5958075991726122
```

In [92]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\programdata\anaconda3\lib\site-
packages (1.6.1)Note: you may need to restart the kernel to use updated pack
ages.
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-pa
ckages (from xgboost) (1.20.3)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-pa
ckages (from xgboost) (1.7.1)
```

In [93]:

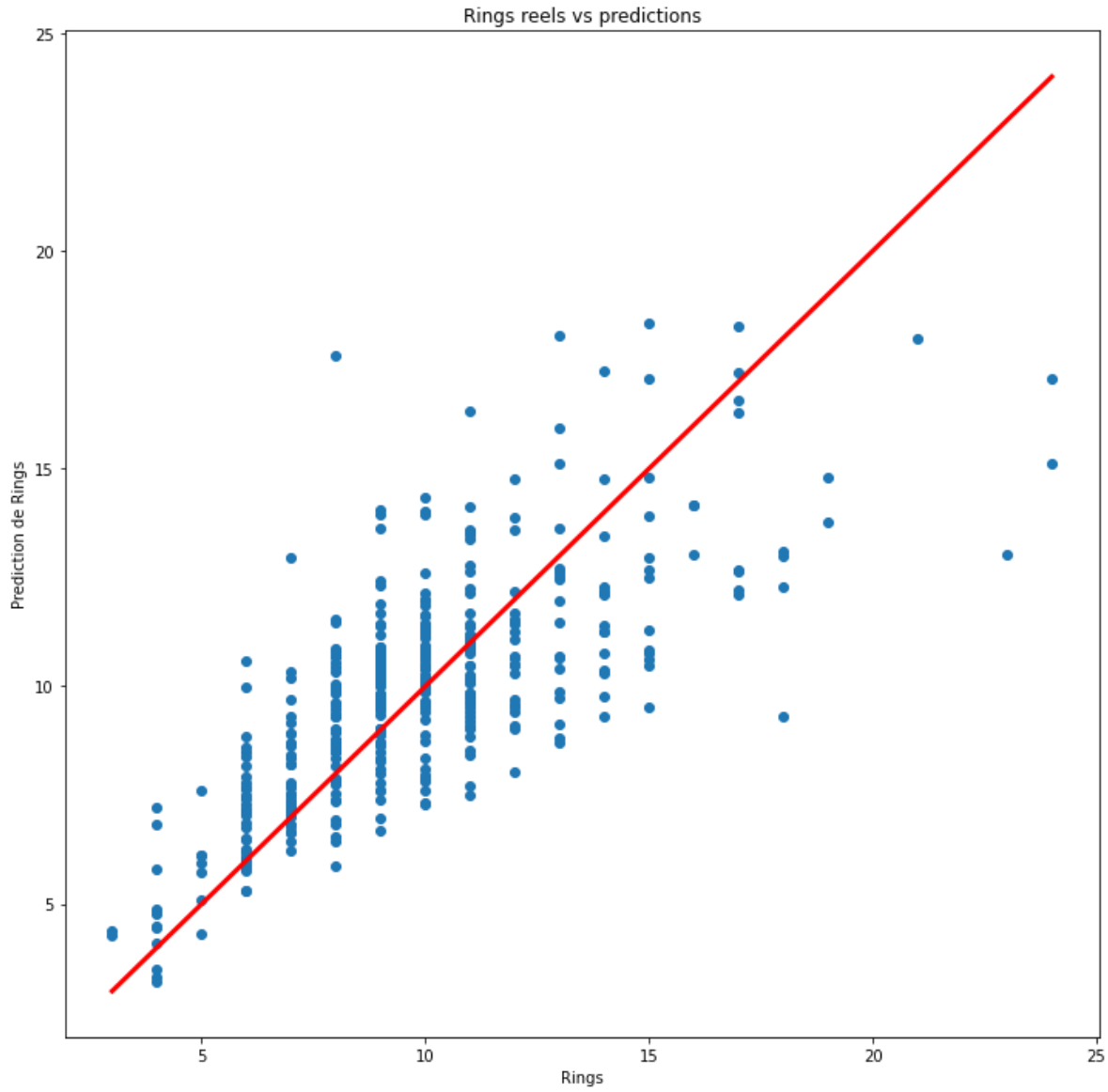
```
import xgboost as XGB
xgb  = XGB.XGBRegressor()
xgb.fit(X_train, y_train)
y_xgb = xgb.predict(X_test)
print(xgb.score(X_test,y_test))

plt.figure(figsize=(12,12))
plt.scatter(y_test, y_xgb)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], color='red', linewidth=3)
plt.xlabel("Rings")
plt.ylabel("Prediction de Rings")
plt.title("Rings reels vs predictions")
```

```
0.5335170867554013
```

Out[93]:

```
Text(0.5, 1.0, 'Rings reels vs predictions')
```



In [ ]: