In [69]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

In [3]:
```
df = pd.read_csv("wine.csv")
```

In [4]:
```
df.head()
```

Out[4]:

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

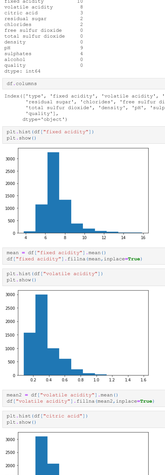
PreProcess

In [5]:
```
df.shape
```

Out[5]:
```
(6497, 13)
```

In [13]:
```
sns.heatmap(df.corr())
```

Out[13]:
```
<AxesSubplot:>
```



In [6]:
```
df.isna().sum()
```

Out[6]:
```
type                    0
fixed acidity          10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
```

In [33]:
```
df.columns
```

Out[33]:
```
Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

In [17]:
```
plt.hist(df["fixed acidity"])
plt.show()
```

In [16]:
```
mean = df["fixed acidity"].mean()
df["fixed acidity"].fillna(mean,inplace=True)
```

In [11]:
```
plt.hist(df["volatile acidity"])
plt.show()
```

In [19]:
```
mean2 = df["volatile acidity"].mean()
df["volatile acidity"].fillna(mean2,inplace=True)
```

In [23]:
```
plt.hist(df["citric acid"])
plt.show()
```

In [22]:
```
mean3 = df["citric acid"].mean()
df["citric acid"].fillna(mean3,inplace=True)
```

In [24]:
```
mean4 = df["residual sugar"].mean()
df["residual sugar"].fillna(mean4,inplace=True)
```

In [25]:
```
mean4 = df["chlorides"].mean()
df["chlorides"].fillna(mean4,inplace=True)
```

In [26]:
```
mean5 = df["pH"].mean()
df["pH"].fillna(mean5,inplace=True)
```

In [27]:
```
mean6 = df["sulphates"].mean()
df["sulphates"].fillna(mean6,inplace=True)
```

In [28]:
```
df.isna().sum()
```

Out[28]:
```
type                    0
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 [31]:
```
sns.set()
plt.figure(figsize=(20,10))
sns.boxplot(data=df,palette="Set1")
plt.show()
```


Outlier Removal

In [30]:
```
lower_limit = df["free sulfur dioxide"].mean() -3*df["free sulfur dioxide"].std()
upper_limit = df["free sulfur dioxide"].mean() +3*df["free sulfur dioxide"].std()
```

In [34]:
```
df2 = df[(df["free sulfur dioxide"] > lower_limit) & (df["free sulfur dioxide"] < upper_limit)]
```

In [37]:
```
df2.shape
```

Out[37]:
```
(6461, 13)
```

In [39]:
```
lower_limit = df2["residual sugar"].mean() -3*df2["residual sugar"].std()
upper_limit = df2["residual sugar"].mean() +3*df2["residual sugar"].std()
```

In [42]:
```
df3 = df2[(df2["residual sugar"] > lower_limit) & (df2["residual sugar"] < upper_limit)]
df3.shape
```

Out[42]:
```
(6435, 13)
```

In [43]:
```
df3.drop("type",axis=1,inplace=True)
```

```
C:\Anaconda\lib\site-packages\pandas\core\frame.py:3990: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

In [44]:
```
df3.head()
```

Out[44]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
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
5	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6

In [51]:
```
df3["quality"].unique()
```

Out[51]:
```
array([6, 5, 7, 8, 4, 3, 9], dtype=int64)
```

Encoding

In [52]:
```
quality_mapping = { 3 : "Low",4 : "Low",5: "Medium",6 : "Medium",7: "Medium",8 : "High",9 : "High"}
df3["quality"] =  df3["quality"].map(quality_mapping)
```

```
<ipython-input-52-cc0467bb3b33>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df3["quality"] =  df3["quality"].map(quality_mapping)
```

In [55]:
```
df3.quality.value_counts()
```

Out[55]:
```
Medium    6011
Low        242
High       182
Name: quality, dtype: int64
```

In [56]:
```
mapping_quality = {"Low" : 0,"Medium": 1,"High" : 2}
df3["quality"] =  df3["quality"].map(mapping_quality)
```

```
<ipython-input-56-762dc8f37c33>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df3["quality"] =  df3["quality"].map(mapping_quality)
```

Feature Importance

In [58]:
```
x = df3.drop("quality",axis=True)
y = df3["quality"]
```

In [60]:
```
model = ExtraTreesClassifier()
model.fit(x,y)
```

Out[60]:
```
ExtraTreesClassifier()
```

In [61]:
```
f_i = pd.Series(model.feature_importances_,index =x.columns)
f_i.nlargest(15).plot(kind="barh")
plt.show()
```


Train test Split

In [64]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42)
```

Model

In [72]:
```
model_params = {
    "svm" : {
        "model":SVC(gamma="auto"),
        "params":{
            'C' : [1,10,20],
            'kernel':['rbf']
        }
    },

    "decision_tree":{
        "model": DecisionTreeClassifier(),
        "params":{
            "criterion":['entropy',"gini"],
            "max_depth":[5,8,9]
        }
    },

    "random_forest":{
        "model": RandomForestClassifier(),
        "params":{
            "n_estimators":[1,5,10],
            "max_depth":[5,8,9]
        }
    },
    "naive_bayes":{
        "model": GaussianNB(),
        "params":{}
    },

    "logistic_regression" : {
        'model' : LogisticRegression(solver='liblinear',multi_class = 'auto'),
        'params' : {
            'C' : [1,5,10]
        }
    }
}
```

In [78]:
```
scores=[]
for model_name,mp in model_params.items():
    clf = GridSearchCV(mp["model"],mp["params"],cv=5,return_train_scores=False)
    clf.fit(x,y)
    scores.append({
        "Model" : model_name,
        "Score": clf.best_score_,
    })
```

In [79]:
```
df5 = pd.DataFrame(scores,columns=["Model","Best_Score"])
```

In [77]:
```
df5
```

Out[77]:

	Model	Best_Score
0	svm	0.931469
1	decision_tree	0.924785
2	random_forest	0.932090
3	naive_bayes	0.780532
4	logistic_regression	0.932246

In [ ]:
```

```