Predicting Red Wine Quality

Data from http://archive.ics.uci.edu/ml/datasets/Wine+Quality (http://archive.ics.uci.edu/ml/datasets/Wine+Quality)

Citations


```
Dua, D. and Karra Taniskidou, E. (2017).
UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/index.php].
Irvine, CA: University of California, School of Information and Computer Science.
```

```
P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties.
In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.
```


Available at:

- @Elsevier (http://dx.doi.org/10.1016/j.dss.2009.05.016)
- Pre-press (pdf) (http://www3.dsi.uminho.pt/pcortez/winequality09.pdf)
- bib (http://www3.dsi.uminho.pt/pcortez/dss09.bib)

Setup

In [2]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

EDA

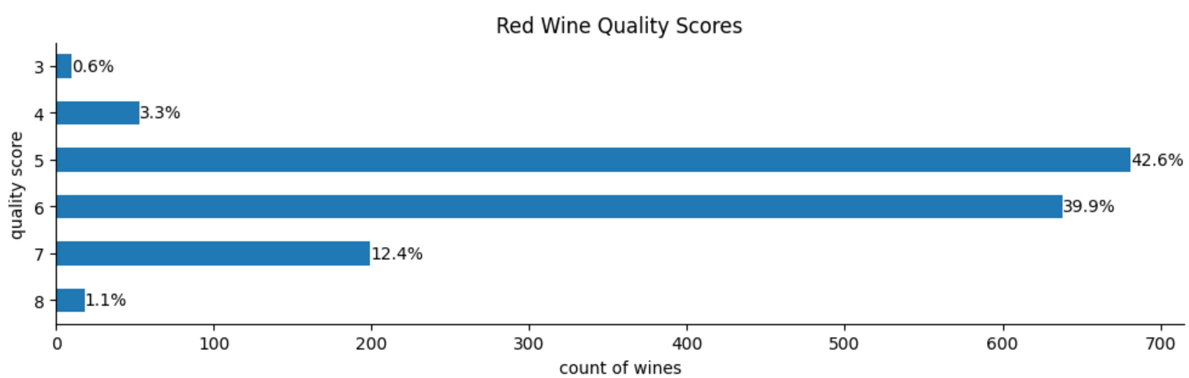
In [3]:
```
red_wine = pd.read_csv('data/winequality-red.csv')
red_wine.head()
```

Out[3]:

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

In [4]:
```
def plot_quality_scores(df, kind):
    ax = df.quality.value_counts().sort_index().plot.barh(
        title=f'{kind.title()} Wine Quality Scores', figsize=(12, 3)
    )
    ax.axes.invert_yaxis()
    for bar in ax.patches:
        ax.text(
            bar.get_width(),
            bar.get_y() + bar.get_height()/2,
            f'{bar.get_width()/df.shape[0]:.1%}',
            verticalalignment='center'
        )
    plt.xlabel('count of wines')
    plt.ylabel('quality score')

    for spine in ['top', 'right']:
        ax.spines[spine].set_visible(False)

    return ax

plot_quality_scores(red_wine, 'red')

# The information on the dataset says that quality varies from 0 (terrible) to 10
# (excellent); however, we only have values in the middle of that range. An interesting task
# for this dataset could be to see if we can predict high-quality red wines (a quality score of
# 7 or higher):
```

Out[4]:
```
<AxesSubplot:title={'center':'Red Wine Quality Scores'}, xlabel='count of wines', ylabel='quality score'>
```



In [5]:
```
red_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 [6]:
```
red_wine.describe()
```

Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4677
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.89532
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

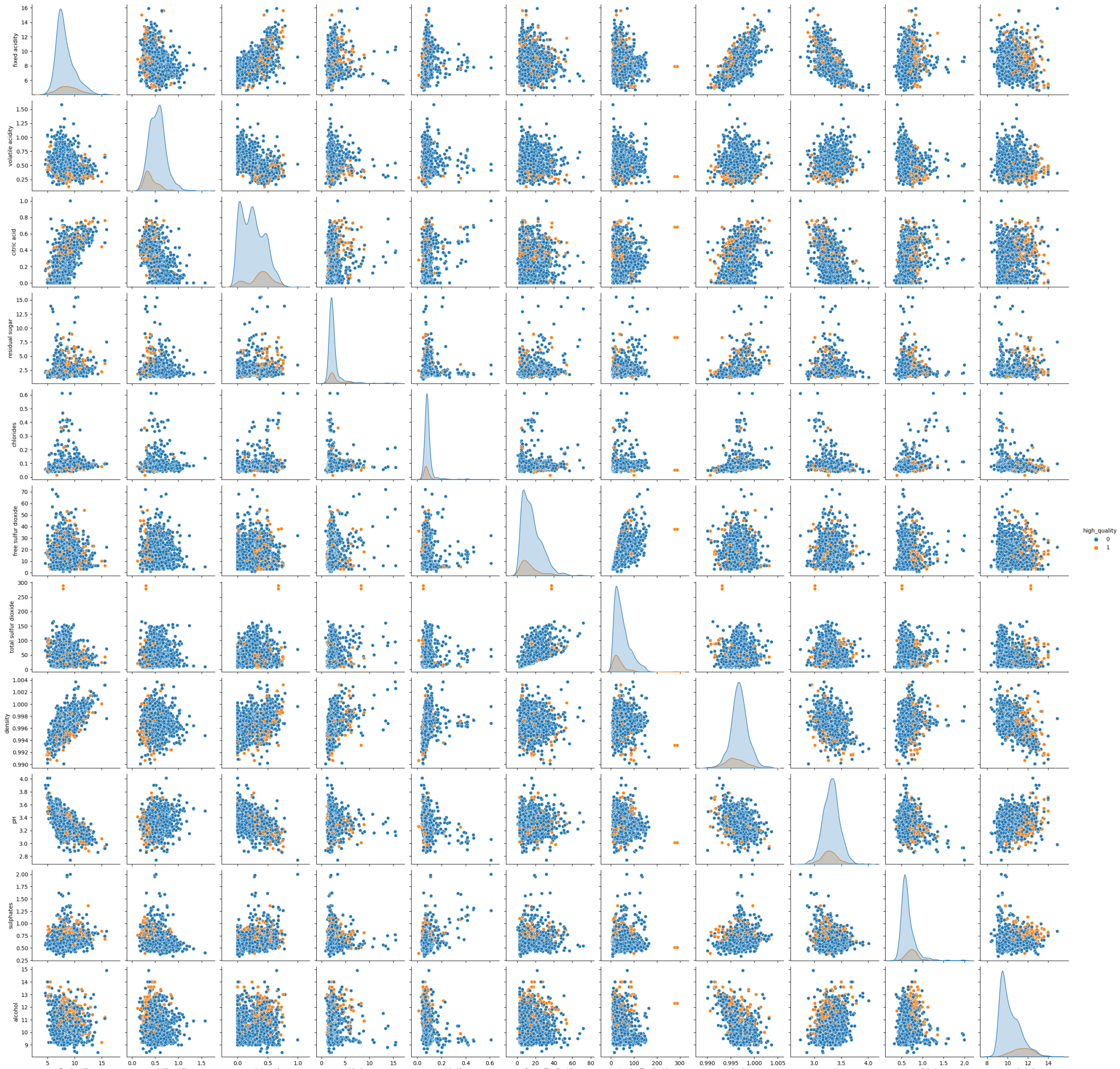
In [7]:
```
red_wine['high_quality'] = pd.cut(red_wine.quality, bins=[0, 6, 10], labels=
[0, 1])
red_wine.high_quality.value_counts(normalize=True)
# use pd.cut() to bin our high-quality red wines (roughly 14% of the data)
```

Out[7]:
```
0    0.86429
1    0.13571
Name: high_quality, dtype: float64
```

It's important to perform an in-depth exploration of the data before modeling. This includes consulting domain experts, looking for correlations between variables, examining distributions, etc. The visualizations covered in chapters 5 and 6 will prove indispensible for this process. One such visualization is the pairplot. In order to predict high quality red wines, we would try to see if there is a difference in the distribution of our variables for low versus high quality red wines. We would also look for correlations. Some other helpful plot types include box plots, heatmaps, and the scatter matrix.

In [8]:
```
sns.pairplot(red_wine.drop(columns='quality'), hue='high_quality')
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x20ed1d7bbe0>
```

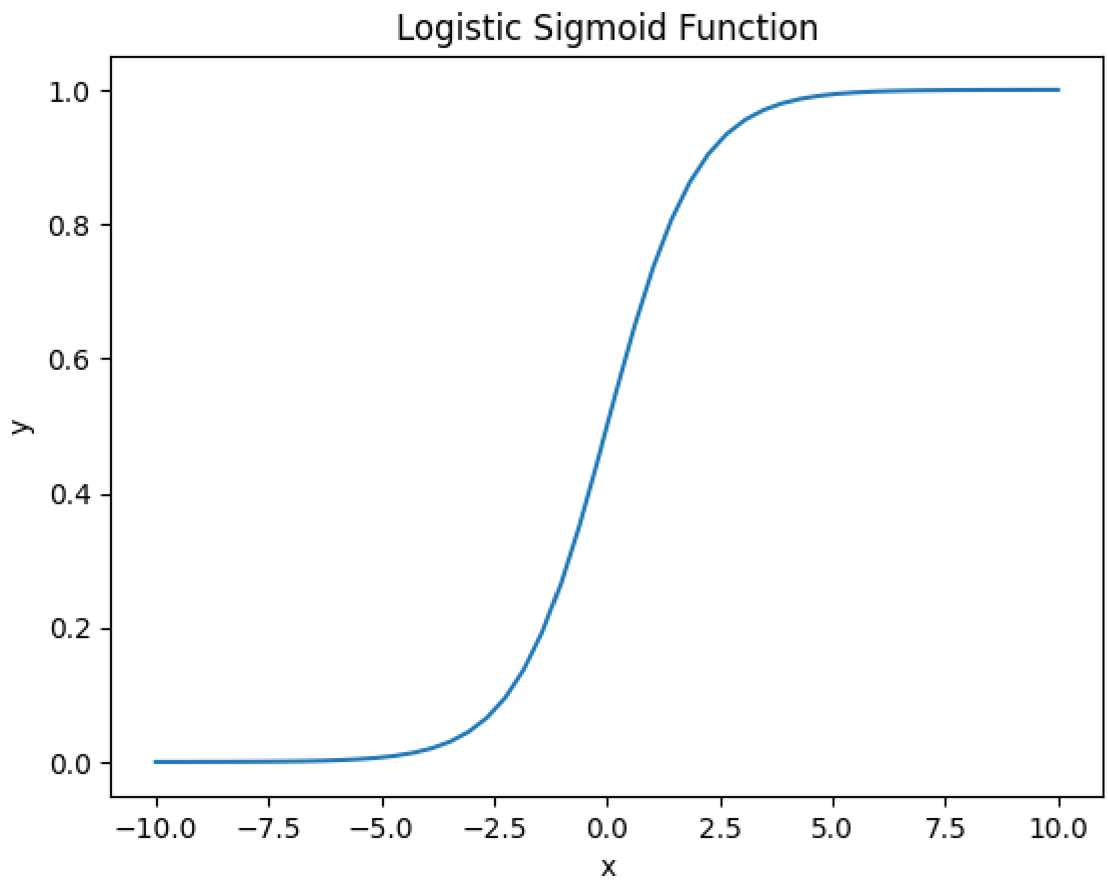


Logistic Regression

The logistic sigmoid function gives values in the range [0, 1], which can be used as probabilities for classification problems:

In [10]:
```
from visual_aids import ml_viz
ml_viz.logistic_sigmoid()
```

Out[10]:
```
[<AxesSubplot:title={'center':'Logistic Sigmoid Function'}, xlabel='x', ylabe
l='y'>]
```



Building a model

1. separate x and y data
2. get the training and testing sets
3. build a pipeline with preprocessing (standardizing here) ending in the model (logistic regression here)
4. fit the model
5. make predictions
6. evaluate predictions

Steps 1 and 2:

In [11]:
```
from sklearn.model_selection import train_test_split

# 1
red_y = red_wine.pop('high_quality')
red_X = red_wine.drop(columns='quality')

# 2
r_X_train, r_X_test, r_y_train, r_y_test = train_test_split(
    red_X, red_y, test_size=0.1, random_state=0, stratify=red_y
)
```

Since we stratified on the high quality versus not from the entire dataset (from `red_y`), we preserve the ratio of high quality to not in both our test and training sets:

In [12]:
```
red_y.value_counts(normalize=True)
```

Out[12]:
```
0    0.86429
1    0.13571
Name: high_quality, dtype: float64
```

Percentage of high- and low-quality red wines in the training set:

In [13]:
```
r_y_train.value_counts(normalize=True)
```

Out[13]:
```
0    0.864489
1    0.135511
Name: high_quality, dtype: float64
```

Percentage of high- and low-quality red wines in the test set:

In [14]:
```
r_y_test.value_counts(normalize=True)
```

Out[14]:
```
0    0.8625
1    0.1375
Name: high_quality, dtype: float64
```

Step 3:

In [15]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

red_quality_lr = Pipeline([
    ('scale', StandardScaler()),
    ('lr', LogisticRegression(
        class_weight='balanced', random_state=0
    ))
])
```

Step 4:

In [16]:
```
red_quality_lr.fit(r_X_train, r_y_train)
```

Out[16]:
```
Pipeline(steps=[('scale', StandardScaler()),
                ('lr',
                 LogisticRegression(class_weight='balanced', random_state=
0))])
```

Step 5:

In [17]:
```
quality_preds = red_quality_lr.predict(r_X_test)
```

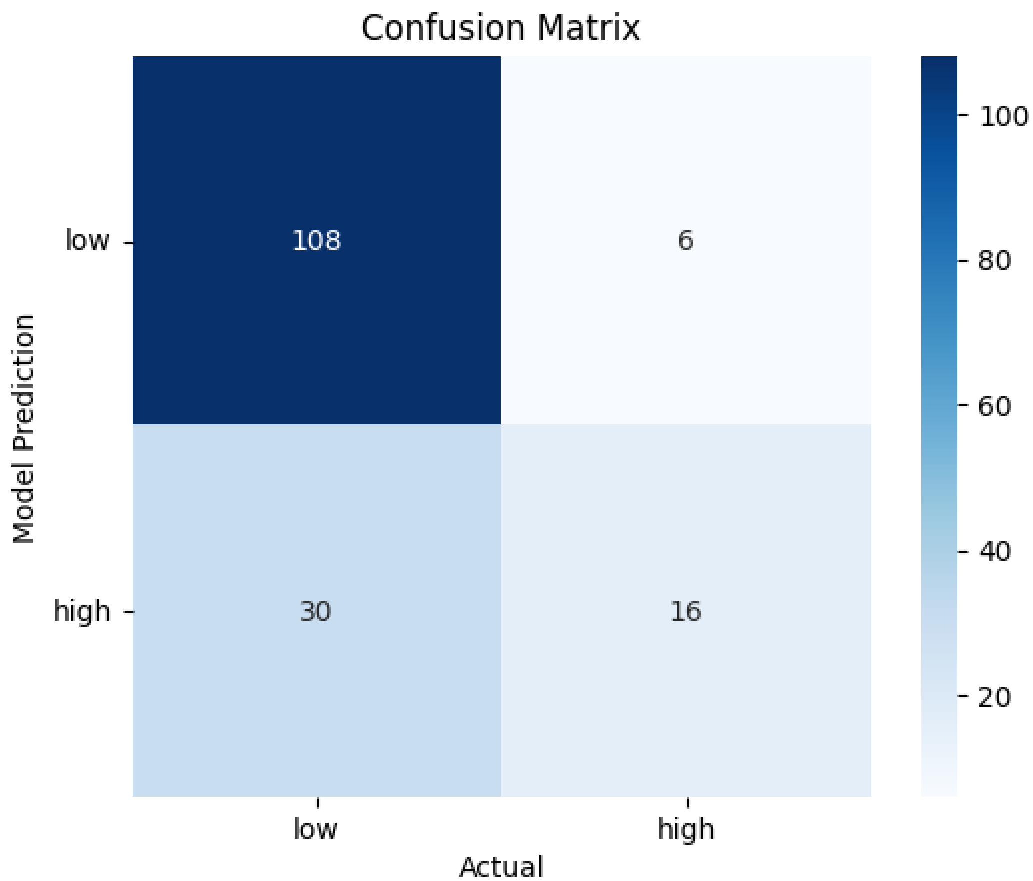
Evaluation

Step 6

We can use a confusion matrix to see how the model's predictions align with the actual class labels. This model gets 36 wrong. It seems to predict high quality too often:

In [18]:
```
from ml_utils.classification import confusion_matrix_visual

confusion_matrix_visual(r_y_test, quality_preds, ['low', 'high'])
```

Out[18]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Actual', ylabel='Mo
del Prediction'>
```



Accuracy tells us how many the model got right. However, it is often misleading in cases of class imbalance (like here):

In [19]:
```
# mean accuracy
red_quality_lr.score(r_X_test, r_y_test)
```

Out[19]:
```
0.775
```

Zero-one loss is our error rate:

In [20]:
```
from sklearn.metrics import zero_one_loss
zero_one_loss(r_y_test, quality_preds)
```

Out[20]:
```
0.22499999999999998
```

Another way to look at performance is with the classification report. Performance is better on the low-quality wines (0 in output below) that are the majority:

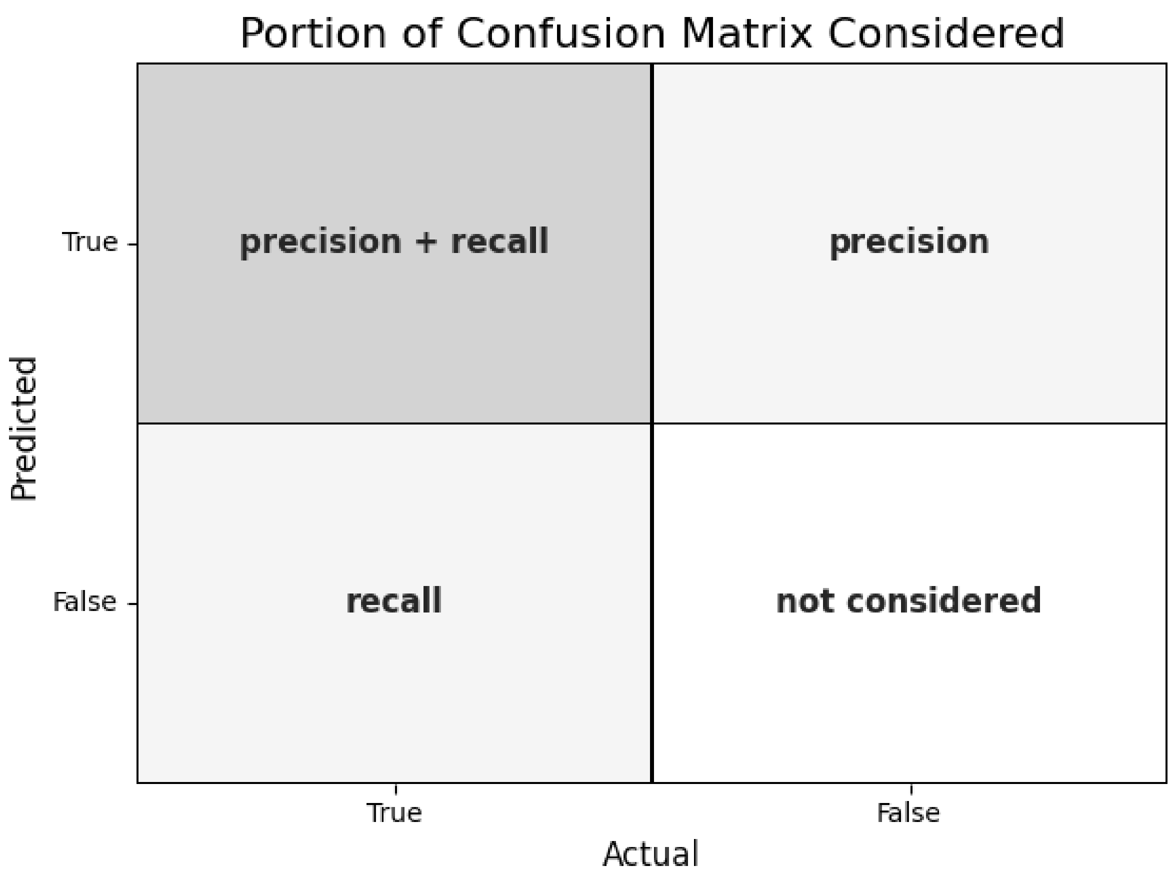
In [21]:
```
from sklearn.metrics import classification_report
print(classification_report(r_y_test, quality_preds))
```

```
              precision    recall  f1-score   support

           0       0.95      0.78      0.86       138
           1       0.35      0.73      0.47        22

    accuracy                           0.78       160
   macro avg       0.65      0.75      0.66       160
weighted avg       0.86      0.78      0.80       160
```

Precision, recall, and F_1 score are more informative when dealing with class imbalance. They ignore true negatives which are likely to be very high (when predicting the minority class):

In [22]:
```
ml_viz.portion_of_confusion_matrix_considered({'precision', 'recall'})
```

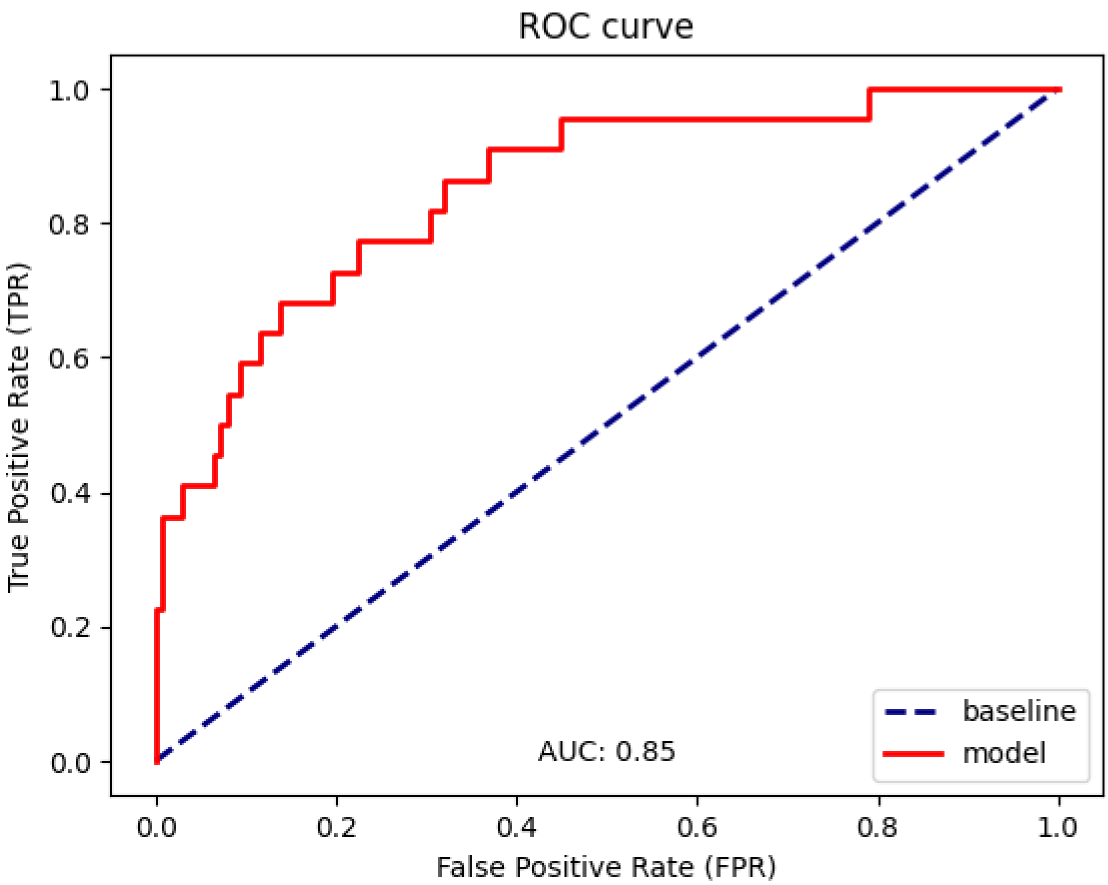
Out[22]:
```
<AxesSubplot:title={'center':'Portion of Confusion Matrix Considered'}, xlabe
l='Actual', ylabel='Predicted'>
```



The ROC curve indicates this is better than the random guessing baseline; however, the performance isn't great:

In [23]:
```
from ml_utils.classification import plot_roc

plot_roc(r_y_test, red_quality_lr.predict_proba(r_X_test)[:,1])
```

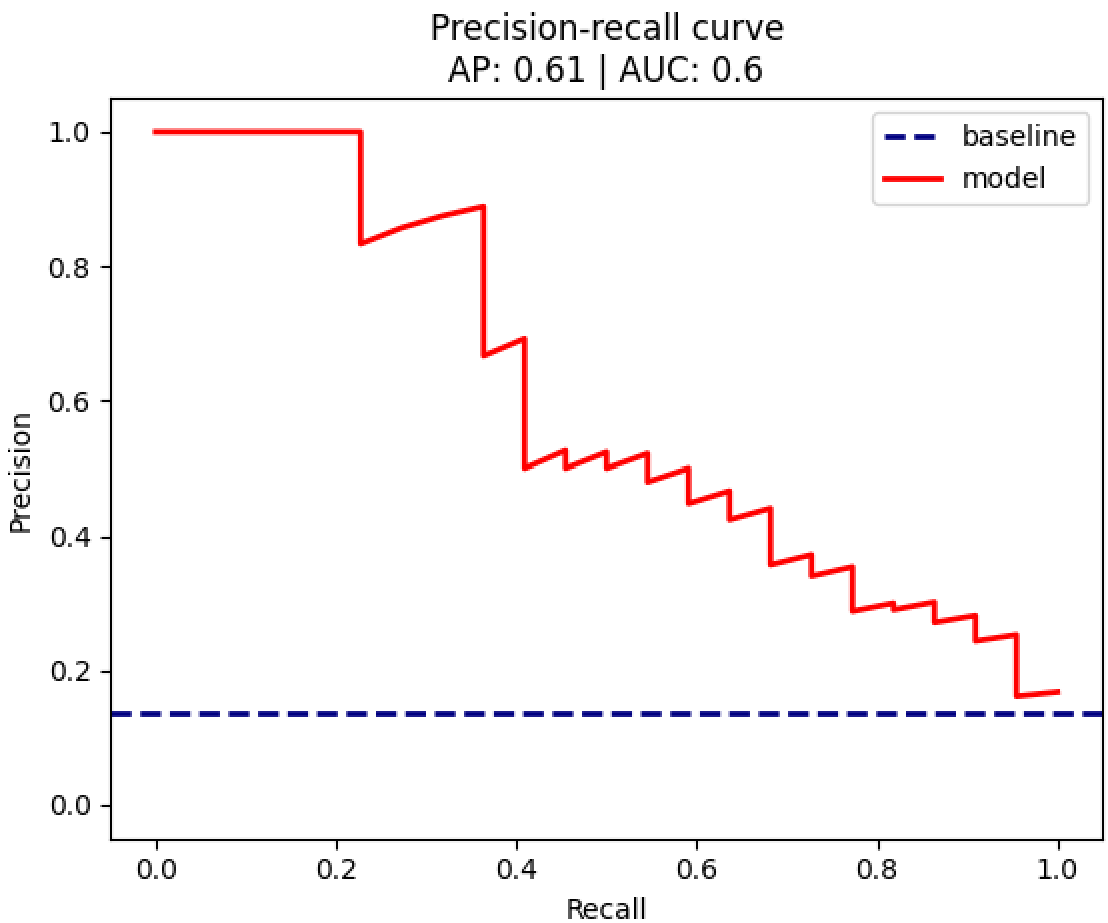
Out[23]:
```
<AxesSubplot:title={'center':'ROC curve'}, xlabel='False Positive Rate (FP
R)', ylabel='True Positive Rate (TPR)'>
```



Remember, the ROC curve includes true negatives so it is optimistic in cases of class imbalance. When faced with class imbalance, we use precision-recall curves since ROC curves will be optimistic of model performance. AP is the weighted average precision, and AUC is the area under the curve once again in the range [0, 1]. The baseline is now the percentage of observations belonging to the positive class. Values below this line are worse than random:

In [24]:
```
from ml_utils.classification import plot_pr_curve

plot_pr_curve(r_y_test, red_quality_lr.predict_proba(r_X_test)[:,1])
```

Out[24]:
```
<AxesSubplot:title={'center':'Precision-recall curve\nAP: 0.61 | AUC: 0.6'},
xlabel='Recall', ylabel='Precision'>
```



← Chapter 8 (../../ch_08/anomaly_detection.ipynb) Preprocessing (./preprocessing.ipynb) Planets (./planets_ml.ipynb) Red + White Wine (./wine.ipynb)
</div> Solutions (../../solutions/ch_09/exercise_1.ipynb) Chapter 10 → (../ch_10/red_wine.ipynb)

In [ ]: