Determining quality of wine based on 11 factors

this dataset is taken from the UCI repository https://archive.ics.uci.edu/ml/datasets/wine+quality (https://archive.ics.uci.edu/ml/datasets/wine+quality)

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew
```

In [2]:

```
wine=pd.read_csv('winequality-red.csv')
wine.head()
```

Out[2]:

	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

In [3]:

```
print(f"There are {wine.shape[0]} rows and {wine.shape[1]} columns in dataset."
)
```

```
There are 1599 rows and 12 columns in dataset.
```

In [4]:

```
wine.dtypes
```

Out[4]:

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [5]:
```
wine.isnull().sum()
```

Out[5]:
```
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 [6]:
```
target=wine['quality']
```

In [7]:
```
features=wine
features=features.drop('quality',axis=1)
```

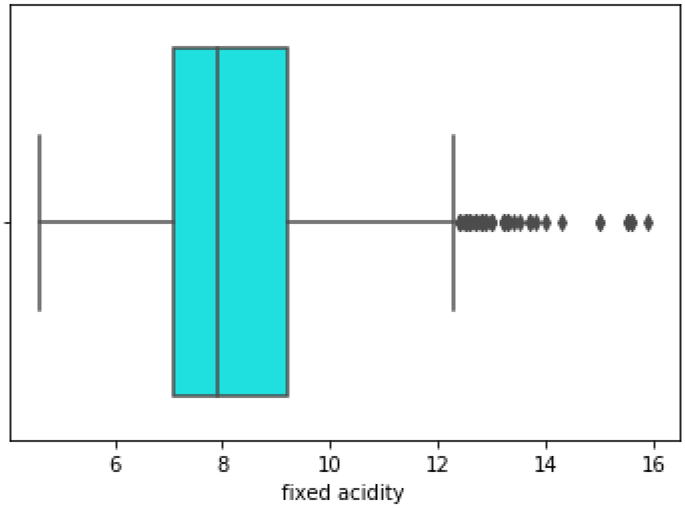
In [8]:
```
pd.set_option('display.float_format', lambda x: '%.3f' % x)
wine.describe().T
```

Out[8]:

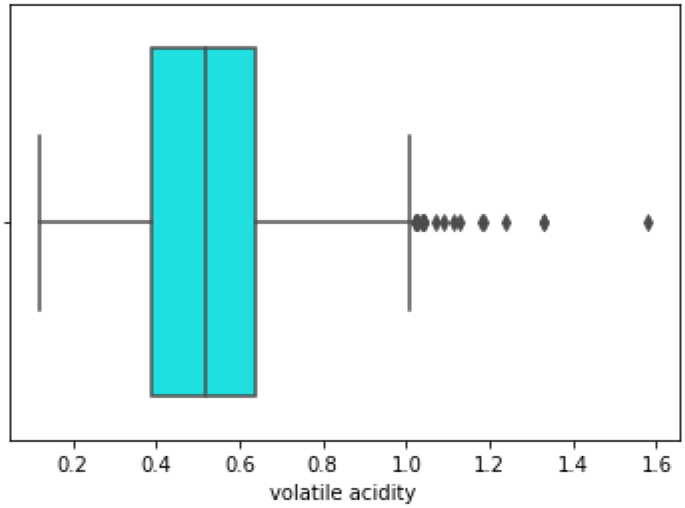
	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.000	8.320	1.741	4.600	7.100	7.900	9.200	15.900
volatile acidity	1599.000	0.528	0.179	0.120	0.390	0.520	0.640	1.580
citric acid	1599.000	0.271	0.195	0.000	0.090	0.260	0.420	1.000
residual sugar	1599.000	2.539	1.410	0.900	1.900	2.200	2.600	15.500
chlorides	1599.000	0.087	0.047	0.012	0.070	0.079	0.090	0.611
free sulfur dioxide	1599.000	15.875	10.460	1.000	7.000	14.000	21.000	72.000
total sulfur dioxide	1599.000	46.468	32.895	6.000	22.000	38.000	62.000	289.000
density	1599.000	0.997	0.002	0.990	0.996	0.997	0.998	1.004
pH	1599.000	3.311	0.154	2.740	3.210	3.310	3.400	4.010
sulphates	1599.000	0.658	0.170	0.330	0.550	0.620	0.730	2.000
alcohol	1599.000	10.423	1.066	8.400	9.500	10.200	11.100	14.900
quality	1599.000	5.636	0.808	3.000	5.000	6.000	6.000	8.000

In [9]:

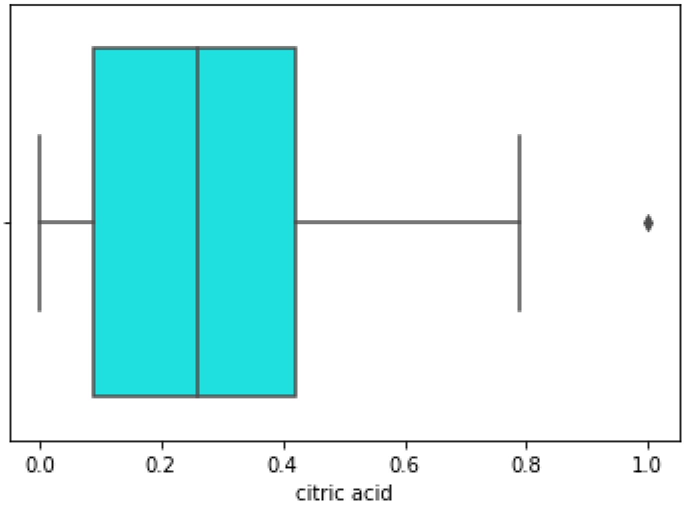
```
for column in wine:
    fig, ax = plt.subplots()
    sns.boxplot(x=wine[column], ax=ax, color="cyan");
    plt.title(f'Boxplot of {column.title().replace("_", " ")}')
```



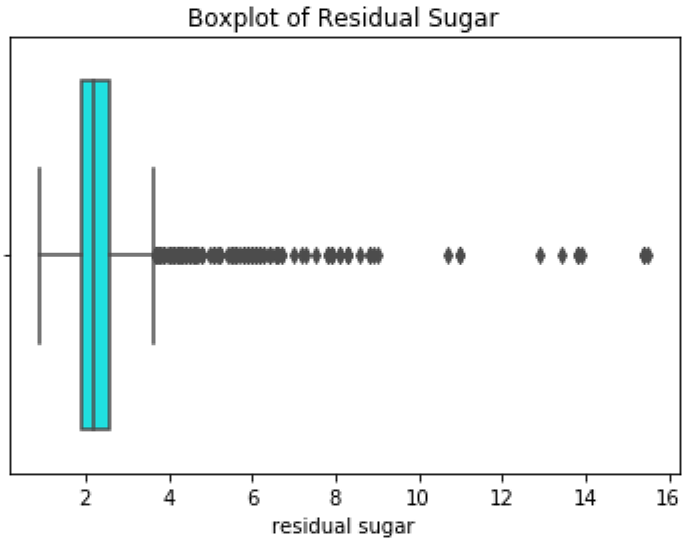
Boxplot of Fixed Acidity

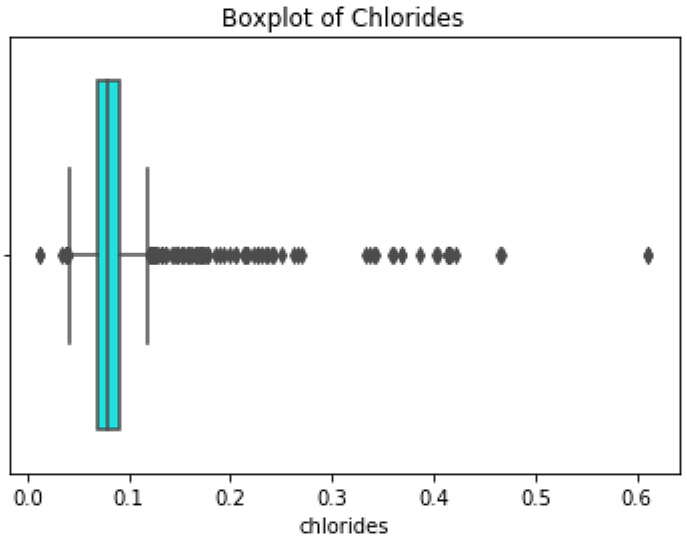


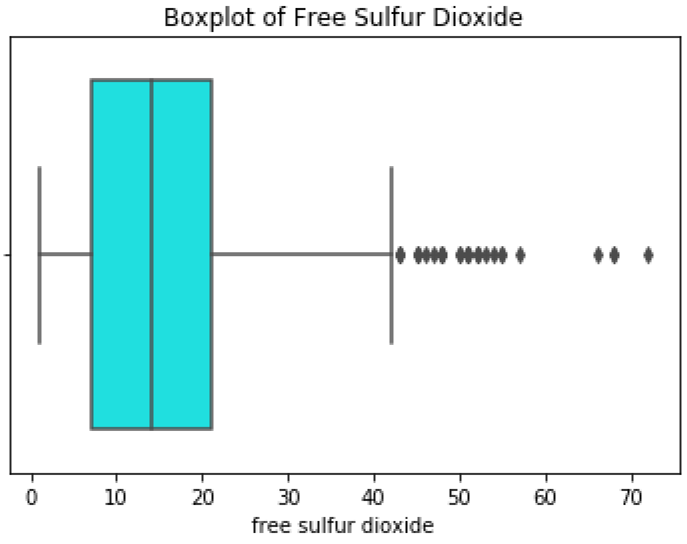
Boxplot of Volatile Acidity

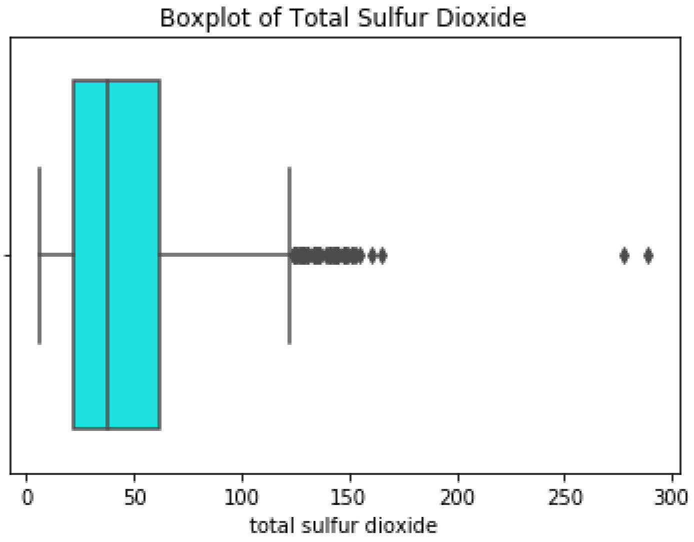


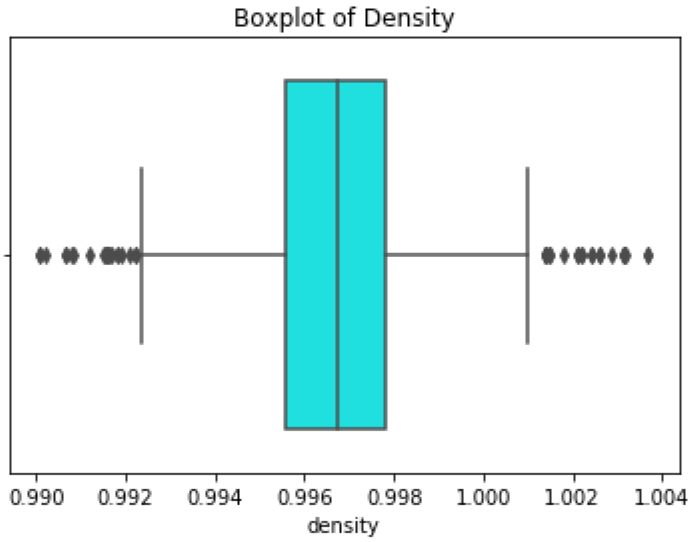
Boxplot of Citric Acid

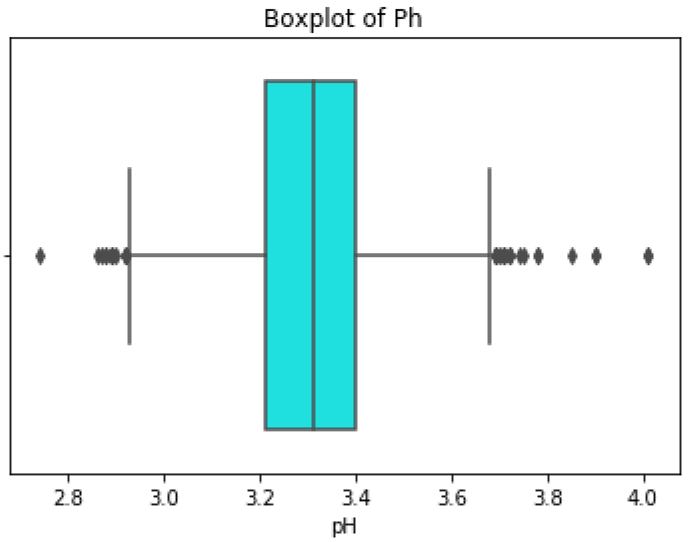

Boxplot of Residual Sugar
2
4
6
8
10
12
14
16
residual sugar

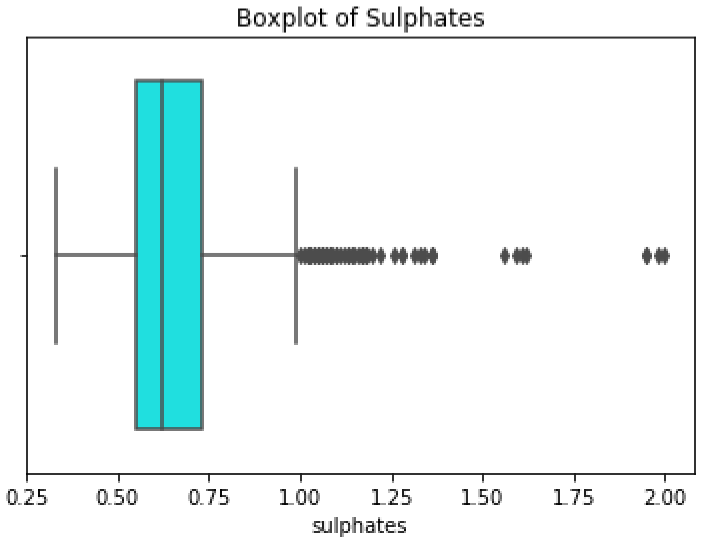

Boxplot of Chlorides
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

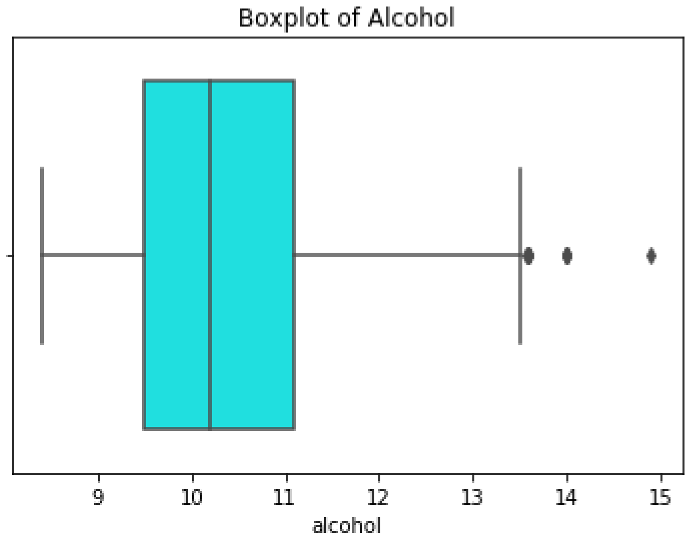

Boxplot of Free Sulfur Dioxide
0
10
20
30
40
50
60
70
free sulfur dioxide

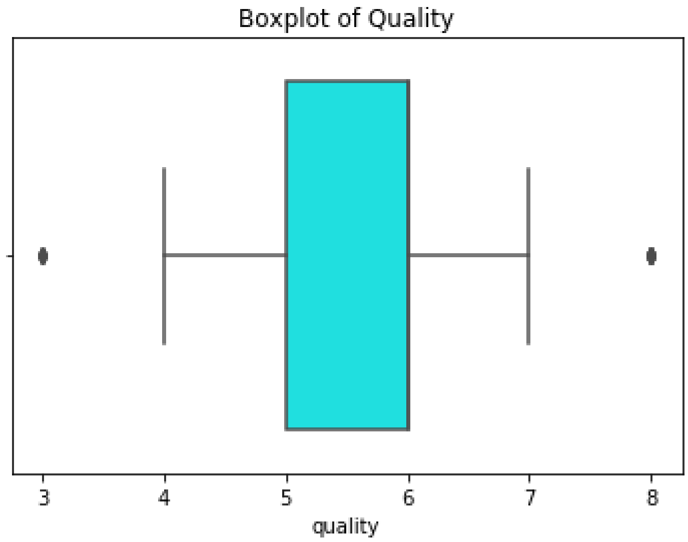

Boxplot of Total Sulfur Dioxide
0
50
100
150
200
250
300
total sulfur dioxide


Boxplot of Density
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
density


Boxplot of Ph
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH







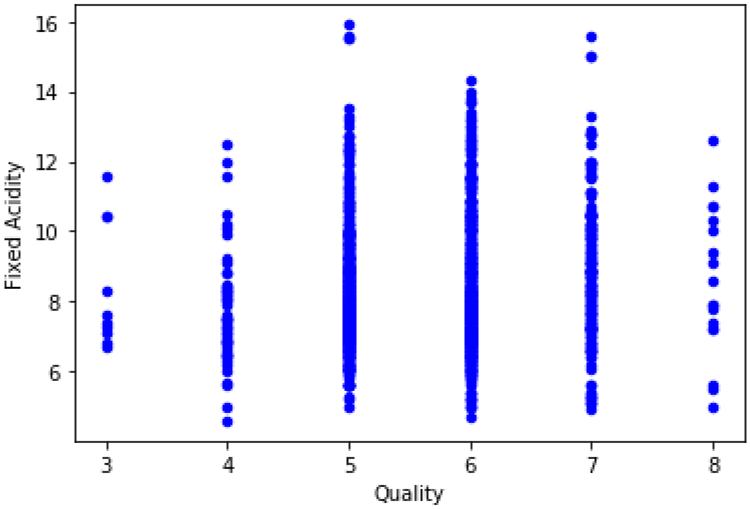
In [10]:

```
scale= wine['quality'].unique().tolist()
scale.sort()
print(scale)
```

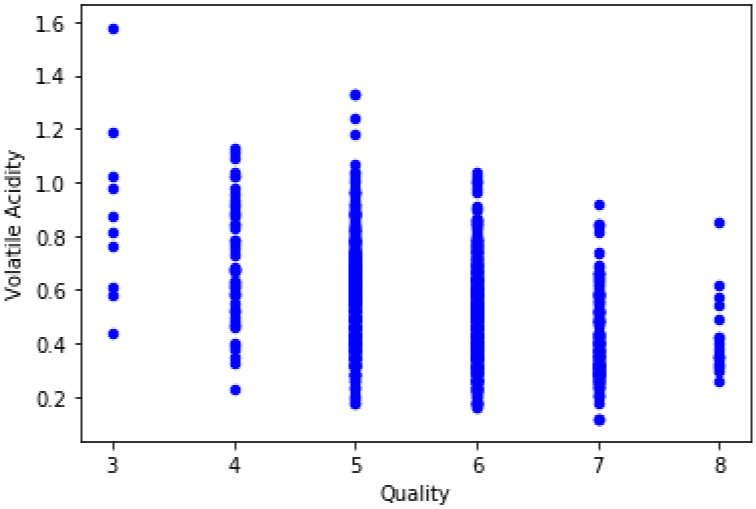
```
[3, 4, 5, 6, 7, 8]
```

In [11]:

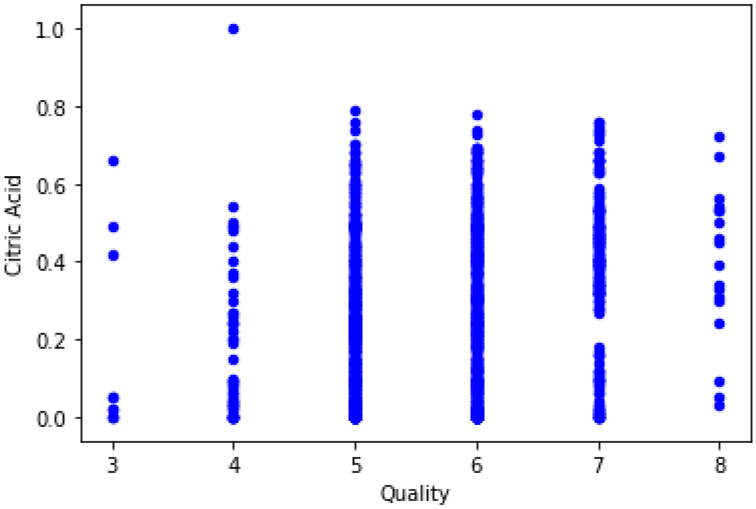
```
for column in features:
    ax=wine.plot(kind='scatter',x='quality',y=[column],color='blue')
    ax.set_xlabel("Quality")
    ax.set_ylabel(column.title().replace("_", " "))
```



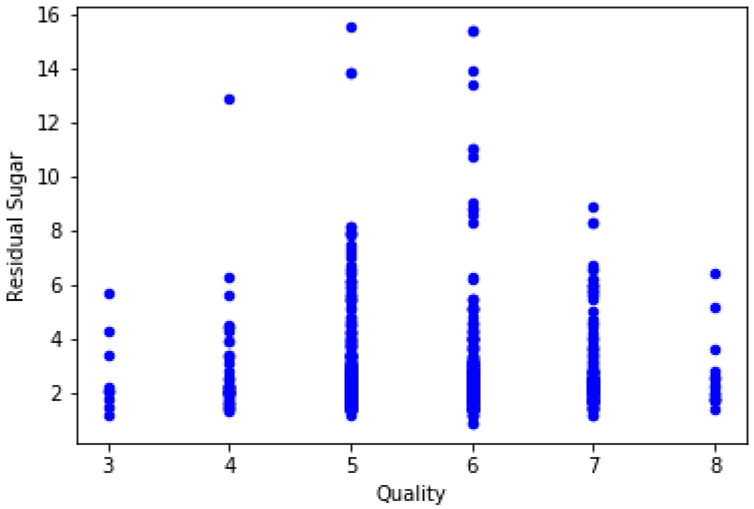
Fixed Acidity
16
14
12
10
8
6
3
4
5
6
7
8
Quality

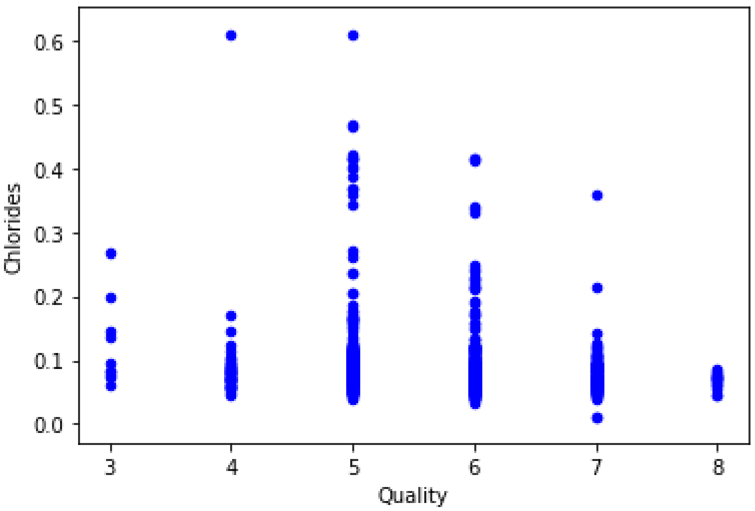


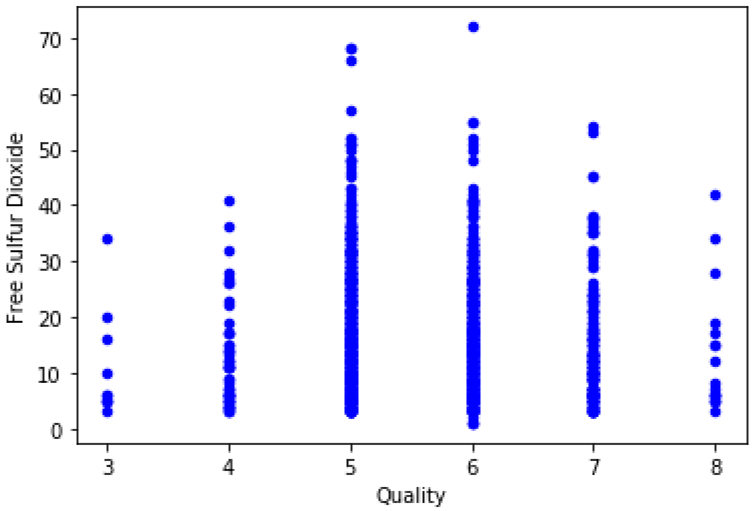
Volatile Acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
3
4
5
6
7
8
Quality

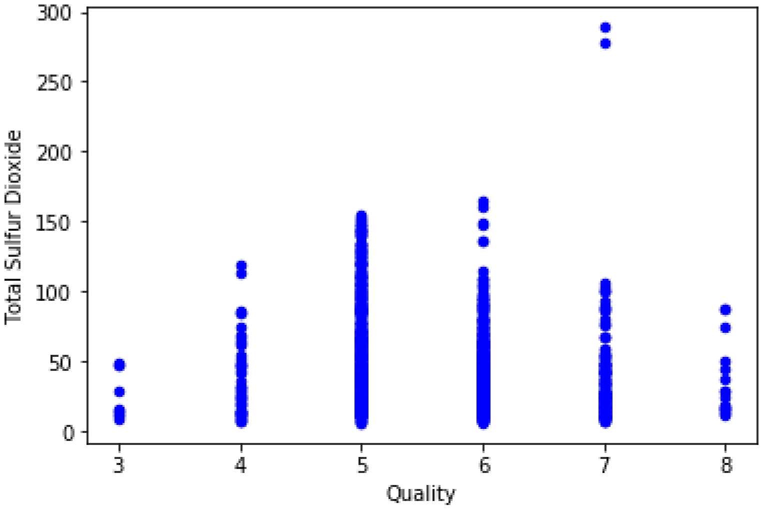


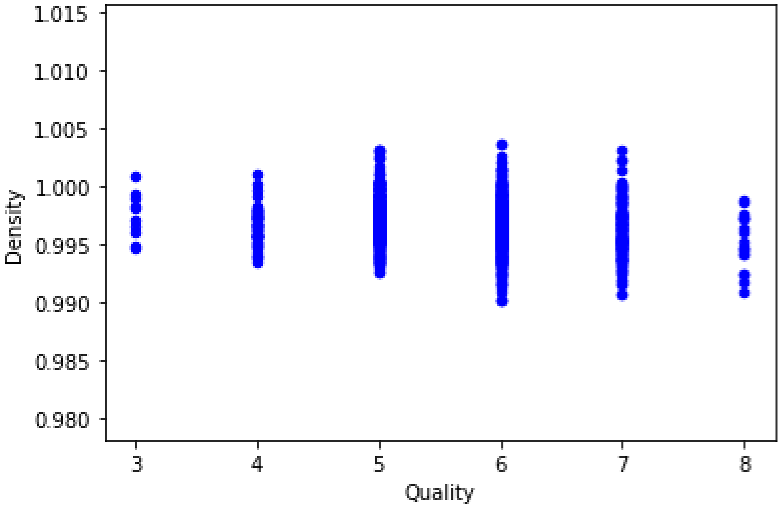
Citric Acid
1.0
0.8
0.6
0.4
0.2
0.0
3
4
5
6
7
8
Quality

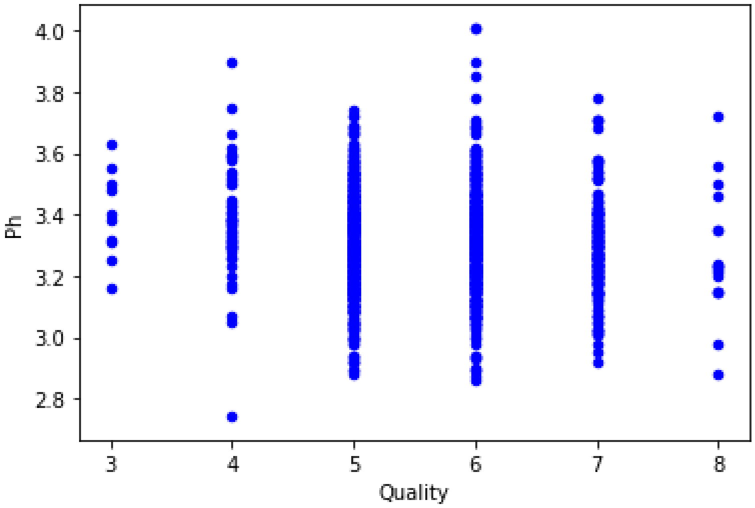

Residual Sugar
16
14
12
10
8
6
4
2
3
4
5
6
7
8
Quality

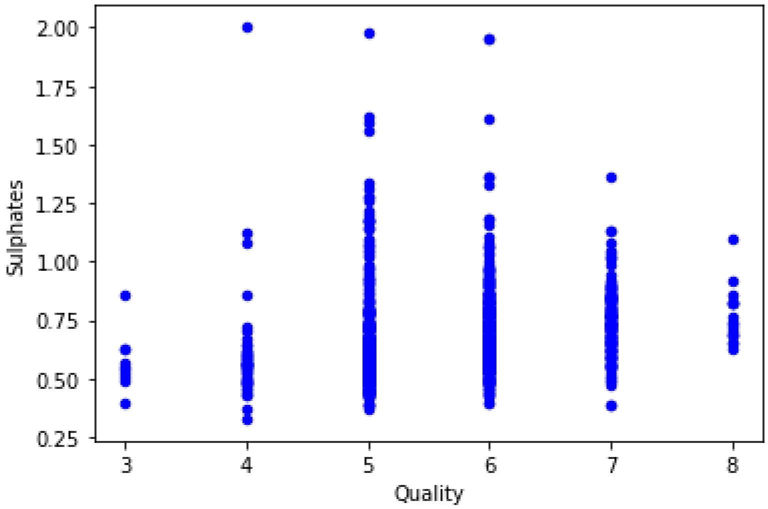

Chlorides
0.6
0.5
0.4
0.3
0.2
0.1
0.0
3
4
5
6
7
8
Quality

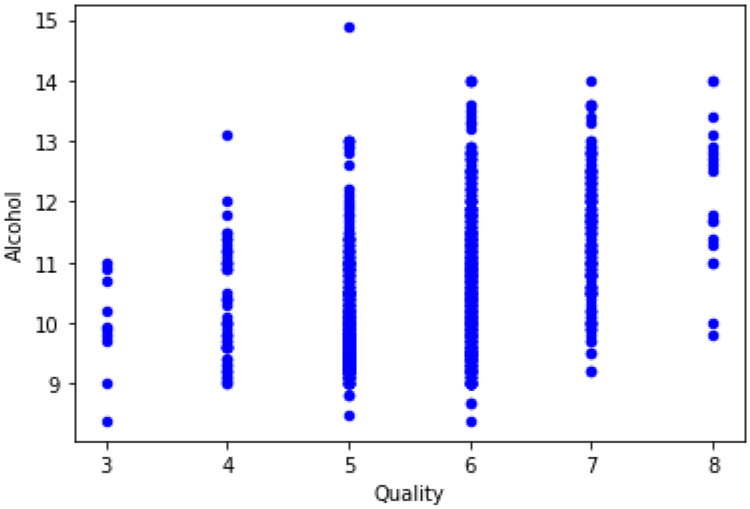

Free Sulfur Dioxide
70
60
50
40
30
20
10
0
3
4
5
6
7
8
Quality


Total Sulfur Dioxide
300
250
200
150
100
50
0
3
4
5
6
7
8
Quality


Density
1.015
1.010
1.005
1.000
0.995
0.990
0.985
0.980
3
4
5
6
7
8
Quality


Ph
4.0
3.8
3.6
3.4
3.2
3.0
2.8
3
4
5
6
7
8
Quality





In [12]:

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

Out[12]:

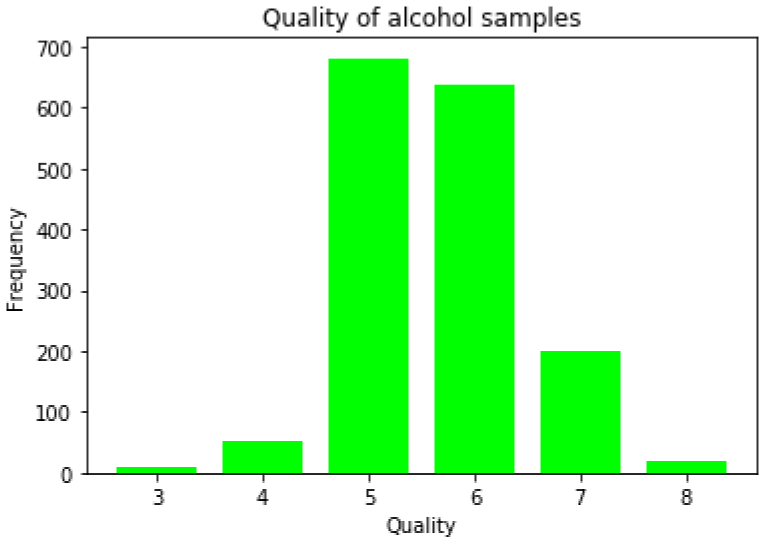
```
3     10
4     53
5    681
6    638
7    199
8     18
Name: quality, dtype: int64
```

In [13]:

```
plt.bar(scale,wine_grp, color='lime',width=0.75)
plt.xlabel('Quality')
plt.ylabel('Frequency')
plt.title("Quality of alcohol samples")
```

Out[13]:

```
Text(0.5, 1.0, 'Quality of alcohol samples')
```



In [14]:

```
wine[list].skew().sort_values()
```

Out[14]:

```
density                 0.071
pH                      0.194
quality                 0.218
citric acid             0.318
volatile acidity        0.672
alcohol                 0.861
fixed acidity           0.983
free sulfur dioxide     1.251
total sulfur dioxide    1.516
sulphates               2.429
residual sugar          4.541
chlorides               5.680
dtype: float64
```

vanilla Regression

In [15]:

```
from sklearn.model_selection import train_test_split
```

In [16]:

```
X_data = features
y_data = target
```

In [17]:

```
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.3, random_state
=42)
```

In [18]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [19]:

```
from sklearn.metrics import mean_squared_error
def rmse(ytrue, ypredicted):
    return np.sqrt(mean_squared_error(ytrue, ypredicted))
```

In [20]:

```
LR = LinearRegression()
error_df = list()

LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

error_df.append(pd.Series({'train': rmse(y_train, y_train_pred),
                           'test' : rmse(y_test,  y_test_pred)},
                           name='no enc'))
```

In [21]:

```
error_df
```

Out[21]:

```
[train    0.649
 test     0.641
 Name: no enc, dtype: float64]
```

Scaling vanilla

In [22]:

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler
scalers = {'standard': StandardScaler(),
           'minmax': MinMaxScaler(),
           'maxabs': MaxAbsScaler()}
```

In [23]:

```
mask = X_train.dtypes == np.float
errors = {}
lists=[]
float_columns = X_train.columns[mask]
for scaler_label, scaler in scalers.items():
        trainingset = X_train.copy()
        testset = X_test.copy()
        trainingset[float_columns] = scaler.fit_transform(trainingset[float_col
umns])
        testset[float_columns] = scaler.transform(testset[float_columns])
        regression=LR.fit(trainingset, y_train)
        predictions = LR.predict(testset)
        key = scaler_label + 'scaling'
        errors[key] = rmse(y_test, predictions)
        coeff=regression.coef_
        lists.append(coeff)

for key, error_val in errors.items():
    print(key, error_val)
```

```
standardscaling 0.6412759715991387
minmaxscaling 0.6412759715991387
maxabsscaling 0.6412759715991388
```

In [24]:

```
coeff_table=pd.DataFrame(np.row_stack(lists))
coeff_table.columns = list(features.columns)
```

In [25]:

```
coeff_table
```

Out[25]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.040	-0.200	-0.048	0.011	-0.079	0.047	-0.108	-0.026	-0.049	0.140	0.309
1	0.265	-1.605	-0.248	0.113	-1.002	0.305	-0.924	-0.194	-0.405	1.325	1.898
2	0.373	-1.737	-0.248	0.120	-1.023	0.309	-0.943	-14.292	-1.280	1.626	4.351

Polynomial regression

In [26]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.3, random_state
=42)

poly = PolynomialFeatures(degree=2)
X_ = poly.fit_transform(X_train)
predict_ = poly.fit_transform(X_test)

clf = linear_model.LinearRegression()
clf.fit(X_, y_train)
y_pred=clf.predict(predict_)
rmse(y_test,y_pred)
```

Out[26]:

```
1.2247685785449771
```

Regularization

In [27]:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(wine, test_size=0.3, random_state=42)
```

In [28]:

```
mask = wine.dtypes == np.float
float_cols = wine.columns[mask]
skew_limit = 0.75
skew_vals = train[float_cols].skew()

skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {0}'.format(skew_limit)))

skew_cols
```

Out[28]:

	Skew
chlorides	5.836
residual sugar	4.511
sulphates	2.554
total sulfur dioxide	1.382
free sulfur dioxide	1.190
fixed acidity	0.994
alcohol	0.915
volatile acidity	0.756

In [29]:

```
pd.options.mode.chained_assignment = None

for col in skew_cols.index.tolist():
    if col == "quality":
        continue
    train[col] = np.log1p(train[col])
    test[col]  = test[col].apply(np.log1p)
```

In [30]:

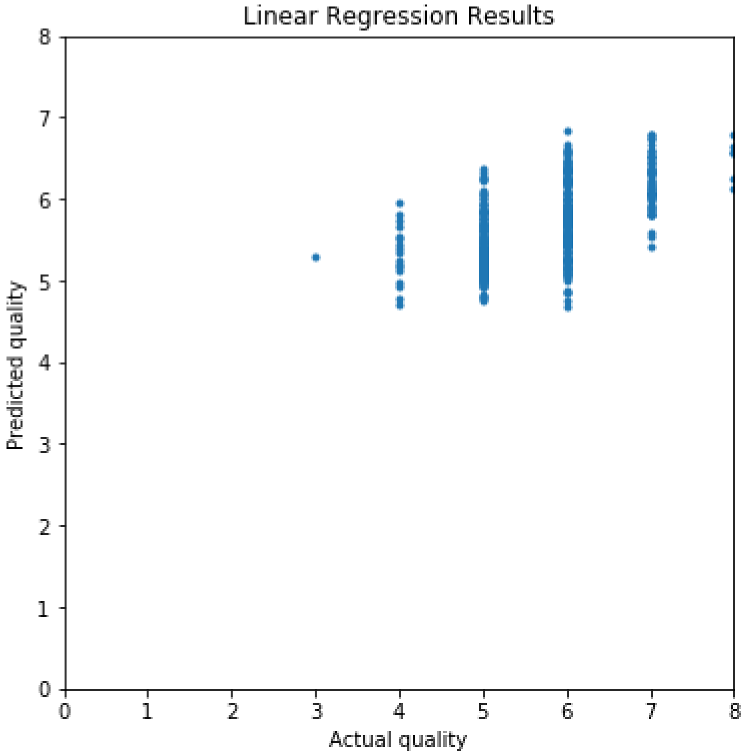
```
feature_cols = [x for x in train.columns if x != 'quality']
X_train = train[feature_cols]
y_train = train['quality']

X_test  = test[feature_cols]
y_test  = test['quality']
```

In [31]:

```
from sklearn.linear_model import LinearRegression

linearRegression = LinearRegression().fit(X_train, y_train)

linearRegression_rmse = rmse(y_test, linearRegression.predict(X_test))

print(linearRegression_rmse)
```

```
0.6414088340302385
```

In [32]:
```
f = plt.figure(figsize=(6,6))
ax = plt.axes()

ax.plot(y_test, linearRegression.predict(X_test),
         marker='o', ls='', ms=3.0)

lim = (0, y_test.max())

ax.set(xlabel='Actual quality',
       ylabel='Predicted quality',
       xlim=lim,
       ylim=lim,
       title='Linear Regression Results');
```



Ridge, Lasso and ElasticNet

In [33]:
```
from sklearn.linear_model import RidgeCV

alphas = [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 80]

ridgeCV = RidgeCV(alphas=alphas,
                  cv=4).fit(X_train, y_train)

ridgeCV_rmse = rmse(y_test, ridgeCV.predict(X_test))

print(ridgeCV.alpha_, ridgeCV_rmse)
```

```
0.3 0.6415615577068983
```

In [34]:
```
from sklearn.linear_model import LassoCV

alphas2 = np.array([1e-5, 5e-5, 0.0001, 0.0005,0.001,0.005,0.01,0.05,0.1,0.5,1
])

lassoCV = LassoCV(alphas=alphas2,
                  max_iter=5e4,
                  cv=3).fit(X_train, y_train)

lassoCV_rmse = rmse(y_test, lassoCV.predict(X_test))

print(lassoCV.alpha_, lassoCV_rmse)
```

```
0.001 0.6435974916416971
```

In [35]:
```
print('Of {} coefficients, {} are non-zero with Lasso.'.format(len(lassoCV.coef
_),
                                                                len(lassoCV.coef
_.nonzero()[0])))
```

```
Of 11 coefficients, 9 are non-zero with Lasso.
```

In [36]:
```
from sklearn.linear_model import ElasticNetCV

l1_ratios = np.linspace(0.1, 0.9, 9)

elasticNetCV = ElasticNetCV(alphas=alphas2,
                            l1_ratio=l1_ratios,
                            max_iter=1e4).fit(X_train, y_train)
elasticNetCV_rmse = rmse(y_test, elasticNetCV.predict(X_test))

print(elasticNetCV.alpha_, elasticNetCV.l1_ratio_, elasticNetCV_rmse)
```

```
0.0005 0.1 0.6419132509667584
```

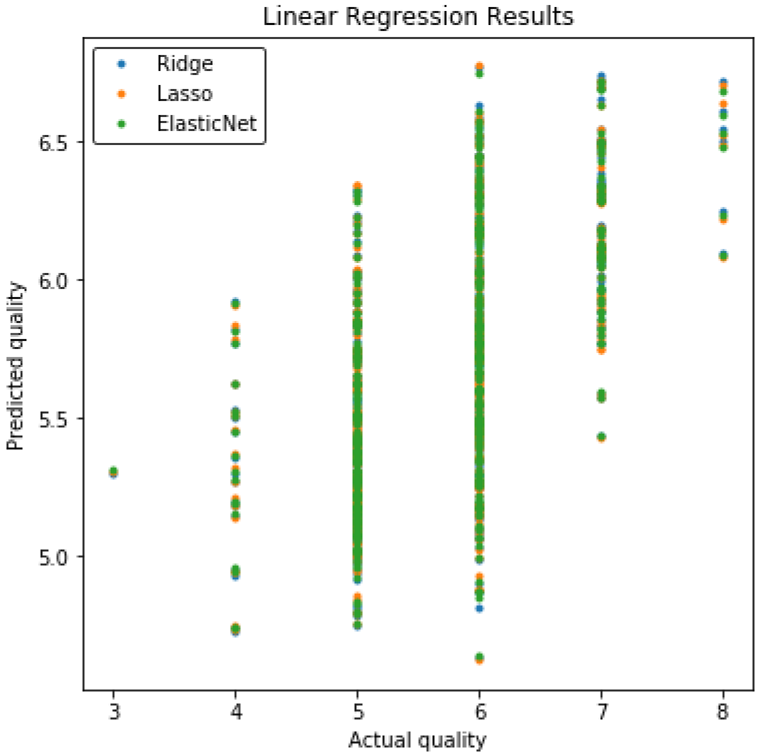
In [37]:
```
rmse_vals = [linearRegression_rmse, ridgeCV_rmse, lassoCV_rmse, elasticNetCV_rm
se]

labels = ['Linear', 'Ridge', 'Lasso', 'ElasticNet']

rmse_df = pd.Series(rmse_vals, index=labels).to_frame()
rmse_df.rename(columns={0: 'RMSE'}, inplace=1)
rmse_df
```

Out[37]:

	RMSE
Linear	0.641
Ridge	0.642
Lasso	0.644
ElasticNet	0.642

In [38]:

```
f = plt.figure(figsize=(6,6))
ax = plt.axes()

labels = ['Ridge', 'Lasso', 'ElasticNet']

models = [ridgeCV, lassoCV, elasticNetCV]

for mod, lab in zip(models, labels):
    ax.plot(y_test, mod.predict(X_test),
             marker='o', ls='', ms=3.0, label=lab)


leg = plt.legend(frameon=True)
leg.get_frame().set_edgecolor('black')
leg.get_frame().set_linewidth(1.0)

ax.set(xlabel='Actual quality',
       ylabel='Predicted quality',
       title='Linear Regression Results');
```



Scohastic gradient descent

In [39]:

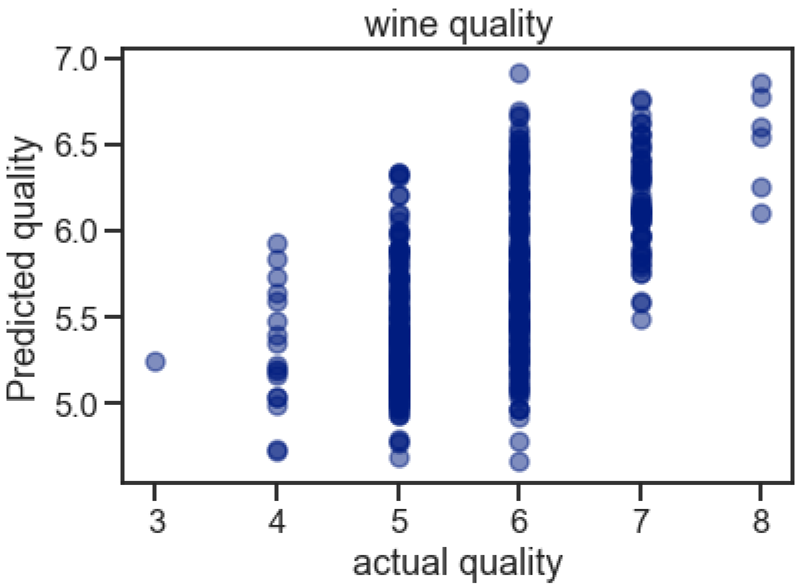
```
from sklearn.linear_model import SGDRegressor

model_parameters_dict = {
    'Linear': {'penalty': 'none'},
    'Lasso': {'penalty': 'l2',
           'alpha': lassoCV.alpha_},
    'Ridge': {'penalty': 'l1',
           'alpha': ridgeCV_rmse},
    'ElasticNet': {'penalty': 'elasticnet',
                   'alpha': elasticNetCV.alpha_,
                   'l1_ratio': elasticNetCV.l1_ratio_}
}

new_rmses = {}
for modellabel, parameters in model_parameters_dict.items():
    # following notation passes the dict items as arguments
    SGD = SGDRegressor(**parameters)
    SGD.fit(X_train, y_train)
    new_rmses[modellabel] = rmse(y_test, SGD.predict(X_test))

rmse_df['RMSE-SGD'] = pd.Series(new_rmses)
rmse_df
```

Out[39]:

	RMSE	RMSE-SGD
Linear	0.641	0.699
Ridge	0.642	0.798
Lasso	0.644	0.696
ElasticNet	0.642	0.701

In [40]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


sns.set_context('talk')
sns.set_style('ticks')
sns.set_palette('dark')

ax = plt.axes()
ax.scatter(y_test, y_test_pred, alpha=.5)

ax.set(xlabel='actual quality',
       ylabel='Predicted quality',
       title='wine quality');
```



In [ ]: