# Regression+Regularization Tutorial

April 22, 2021

## 0.1 Regression Tutorial: UCI Red Wine Quality Dataset

In this tutorial, simple linear regression will be modelled with the red wine quality dataset. In this dataset, various parameters describing wine such as the fixed acidity, pH, total sulfur dioxide, etc. will be used to rate wine quality from a scale of 0 to 10.

The dataset is availiable at UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Wine+Quality)

Some metrics such as MSE and R-Squared scores will also be shown for the following: * Linear Regression * Lasso, Ridge, and ElasticNet Regression * Decision Tree Regression * Random Forest Regression * Support Vector Regression

### 0.1.1 Data Manipulation: Creating Training and Testing Sets

[160]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

[161]:
```
df = pd.read_csv('winequality-red.csv', sep=';')
df.head()
```

[161]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
```

```
2      9.8        5
3      9.8        6
4      9.4        5
```

[162]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from seaborn import heatmap
from sklearn.preprocessing import StandardScaler
```

In this case, our target will be the quality of the wine. We will separate the target from the rest of the data, which will be scaled using the StandScaler.

[163]:
```python
# Creating a temporary dataframe for the quality
df_target = pd.DataFrame(df['quality'])
df_target
```

[163]:
```
      quality
0           5
1           5
2           5
3           6
4           5
...       ...
1594        5
1595        6
1596        6
1597        5
1598        6

[1599 rows x 1 columns]
```

[164]:
```python
# Dropping the quality column in the original dataframe
df.drop(['quality'], axis=1, inplace=True)
df.head()
```

[164]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol
```

```
0       9.4
1       9.8
2       9.8
3       9.8
4       9.4
```

The motivation of StandardScaler() is that it transforms the data such that its distribution will have a mean of 0 and standard deviation of 1. In other words, the transformed value of the dataframe is the (original value - mean) / standard deviation.

Additional information about feature scaling can be found in the following link. (http://sebastianraschka.com/Articles/2014_about_feature_scaling.html#standardization-and-min-max-scaling)

[165]:
```
# Creating a scaler object
scaler = StandardScaler()

# Fitting and transforming the original data to scaled data
df_s=scaler.fit_transform(df)
df_data=pd.DataFrame(df_s)
df_data.head()
```

[165]:
```
          0         1         2         3         4         5         6  \
0 -0.528360  0.961877 -1.391472 -0.453218 -0.243707 -0.466193 -0.379133
1 -0.298547  1.967442 -1.391472  0.043416  0.223875  0.872638  0.624363
2 -0.298547  1.297065 -1.186070 -0.169427  0.096353 -0.083669  0.229047
3  1.654856 -1.384443  1.484154 -0.453218 -0.264960  0.107592  0.411500
4 -0.528360  0.961877 -1.391472 -0.453218 -0.243707 -0.466193 -0.379133

          7         8         9        10
0  0.558274  1.288643 -0.579207 -0.960246
1  0.028261 -0.719933  0.128950 -0.584777
2  0.134264 -0.331177 -0.048089 -0.584777
3  0.664277 -0.979104 -0.461180 -0.584777
4  0.558274  1.288643 -0.579207 -0.960246
```

[166]:
```
# Piece together the newly transformed data and the target previously created␣
 ↪column-wise
scaled_df = pd.concat([df_target, df_data], axis=1)
scaled_df.head()
```

[166]:
```
   quality         0         1         2         3         4         5  \
0        5 -0.528360  0.961877 -1.391472 -0.453218 -0.243707 -0.466193
1        5 -0.298547  1.967442 -1.391472  0.043416  0.223875  0.872638
2        5 -0.298547  1.297065 -1.186070 -0.169427  0.096353 -0.083669
3        6  1.654856 -1.384443  1.484154 -0.453218 -0.264960  0.107592
4        5 -0.528360  0.961877 -1.391472 -0.453218 -0.243707 -0.466193

          6         7         8         9        10
0 -0.379133  0.558274  1.288643 -0.579207 -0.960246
1  0.624363  0.028261 -0.719933  0.128950 -0.584777
```

```
2  0.229047  0.134264 -0.331177 -0.048089 -0.584777
3  0.411500  0.664277 -0.979104 -0.461180 -0.584777
4 -0.379133  0.558274  1.288643 -0.579207 -0.960246
```

[167]:
```
X = scaled_df[[0,1,2,3,4,5,6,7,8,9,10]]
y = scaled_df['quality']
X.head()
```

[167]:
```
          0         1         2         3         4         5         6  \
0 -0.528360  0.961877 -1.391472 -0.453218 -0.243707 -0.466193 -0.379133
1 -0.298547  1.967442 -1.391472  0.043416  0.223875  0.872638  0.624363
2 -0.298547  1.297065 -1.186070 -0.169427  0.096353 -0.083669  0.229047
3  1.654856 -1.384443  1.484154 -0.453218 -0.264960  0.107592  0.411500
4 -0.528360  0.961877 -1.391472 -0.453218 -0.243707 -0.466193 -0.379133

          7         8         9        10
0  0.558274  1.288643 -0.579207 -0.960246
1  0.028261 -0.719933  0.128950 -0.584777
2  0.134264 -0.331177 -0.048089 -0.584777
3  0.664277 -0.979104 -0.461180 -0.584777
4  0.558274  1.288643 -0.579207 -0.960246
```

Next, the training and testing sets are created.

[168]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪3,random_state=8)
```

### 0.1.2 Linear Regression

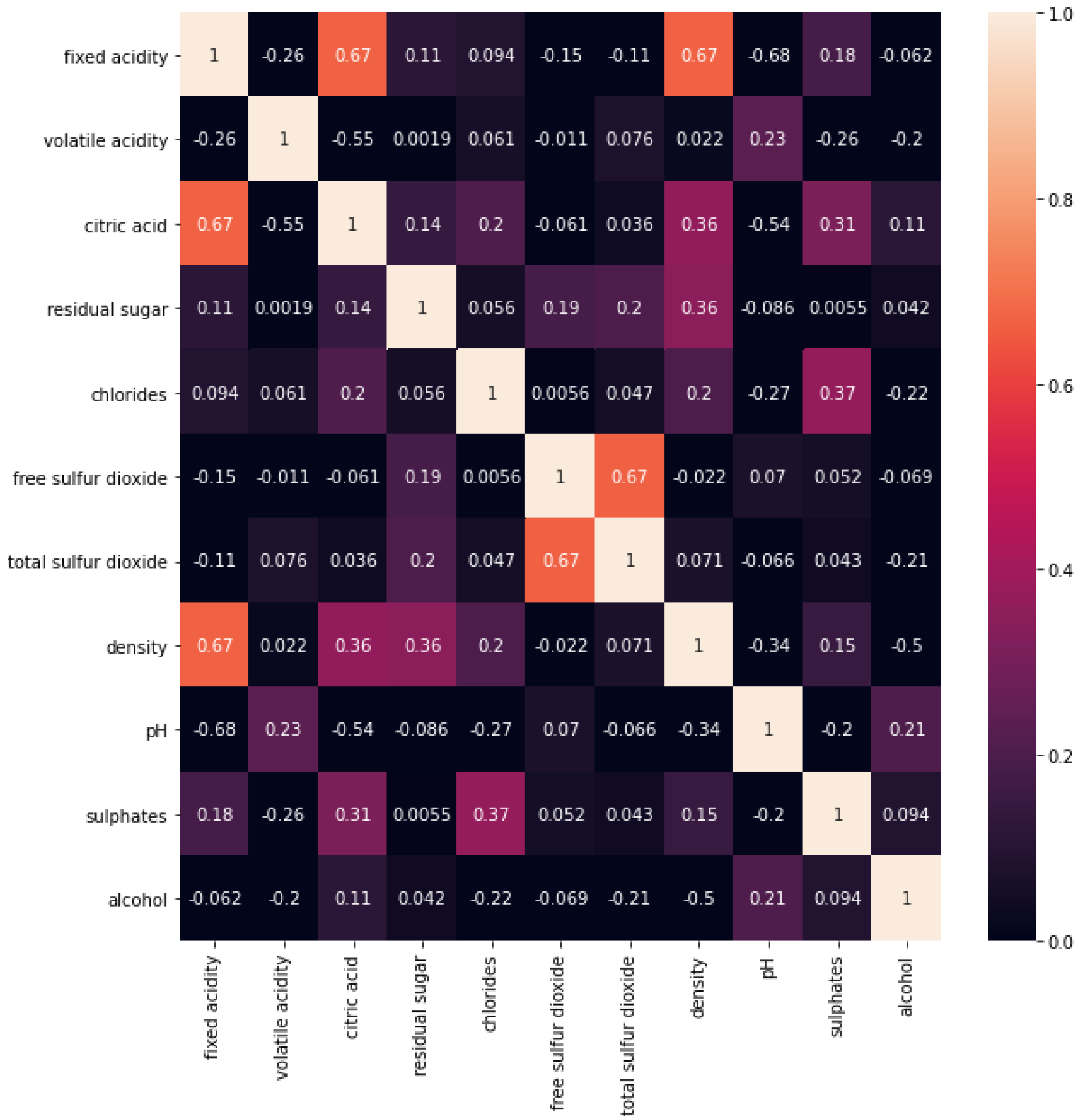
A heatmap may be helpful visual tool in determining which two attributes have the highest correlation to consider before building a model. A possible usage of the heatmap is to threshold out attributes that do not have high correlation pairs. For example, the pair (fixed acidity, density) have a correlation rho of 0.67 which is much higher than the (volatile acidity, residual sugar) pair.

This article (https://medium.com/datadriveninvestor/regression-from-scratch-wine-quality-prediction-d61195cb91c8) provides a step by step walk through on using feature selection based on a threshold for linear regression.

Here, we will show some metrics such as the MSE and MAE without thresholding features.

[169]:
```
plt.figure(figsize=(10,10))
heatmap(df.corr(), vmin=0, vmax=1, annot=True, color='Red')
```

[169]:
```
<AxesSubplot:>
```




[170]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

[171]:
```
lin_reg = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,␣
↪normalize=False)
lin_reg.fit(X_train,y_train)
```

[171]:
```
LinearRegression(n_jobs=1)
```

[172]:
```
lin_reg.coef_, lin_reg.intercept_
```

[172]:
```
(array([ 0.04059119, -0.17948958, -0.01560414,  0.01825026, -0.06667982,
         0.06023032, -0.11298877, -0.05606959, -0.06663801,  0.17100804,
         0.28801656]),
 5.643495301439839)
```

[173]:
```
y_hat = lin_reg.predict(X_test)
```

[174]:
```
# the score is the r-squred values. a value close to 1 is the best.
score=lin_reg.score(X_test, y_test)
score
```

[174]: 0.3365811445014787

Here is a snippet of predictions from the model.

[175]:
```
df2 = pd.DataFrame({'Actual Quality': y_test, 'Predicted Quality': y_hat[:
 ↪len(y_test)]})
df2.head()
```

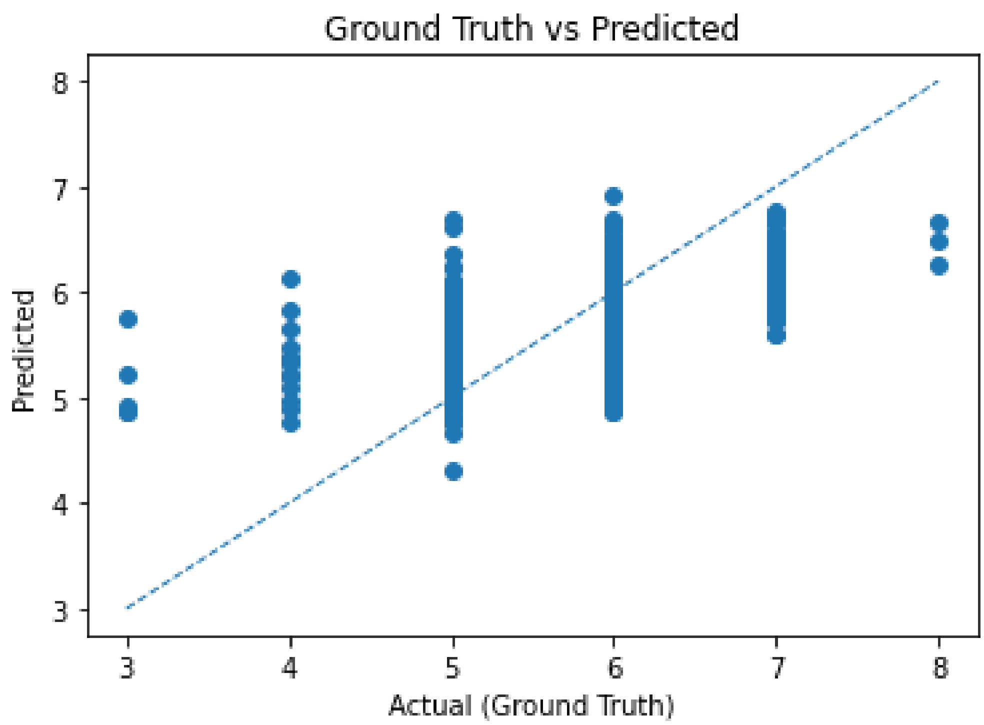
[175]:

| | Actual Quality | Predicted Quality |
|---|---|---|
| 145 | 5 | 4.934385 |
| 345 | 5 | 5.432563 |
| 603 | 6 | 5.334275 |
| 319 | 6 | 5.365508 |
| 1544 | 7 | 6.355804 |

Below is a visual of the predictions and the actual qualities for comparision.

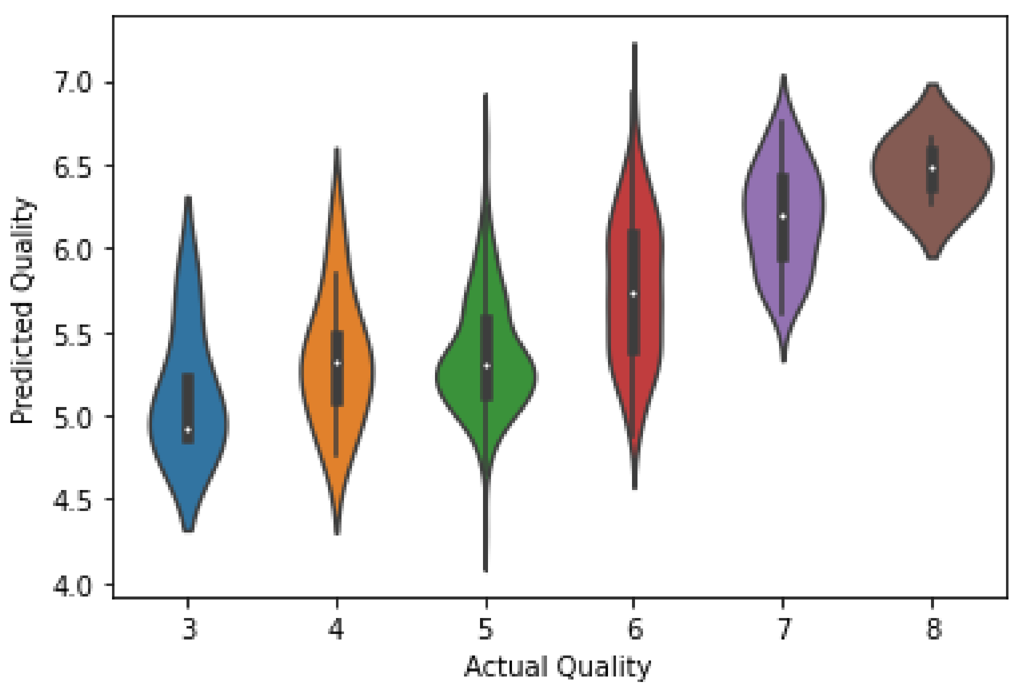
[176]:
```
fig, ax = plt.subplots()
ax.scatter(y_test, y_hat)
ax.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', lw=1)
ax.set_xlabel('Actual (Ground Truth)')
ax.set_ylabel('Predicted')
ax.set_title("Ground Truth vs Predicted")
```

[176]: Text(0.5, 1.0, 'Ground Truth vs Predicted')




[177]:
```python
import seaborn as sns

sns.violinplot(x="Actual Quality", y="Predicted Quality", data=df2)
```

[177]: <AxesSubplot:xlabel='Actual Quality', ylabel='Predicted Quality'>



[178]:
```
r2 = r2_score(y_test, y_hat)
mse = mean_squared_error(y_test, y_hat)
mae = mean_absolute_error(y_test, y_hat)
r2, mse, mae
```

[178]:
```
(0.3365811445014787, 0.4115702001331647, 0.4964410749148005)
```

### 0.1.3 Ridge Regression

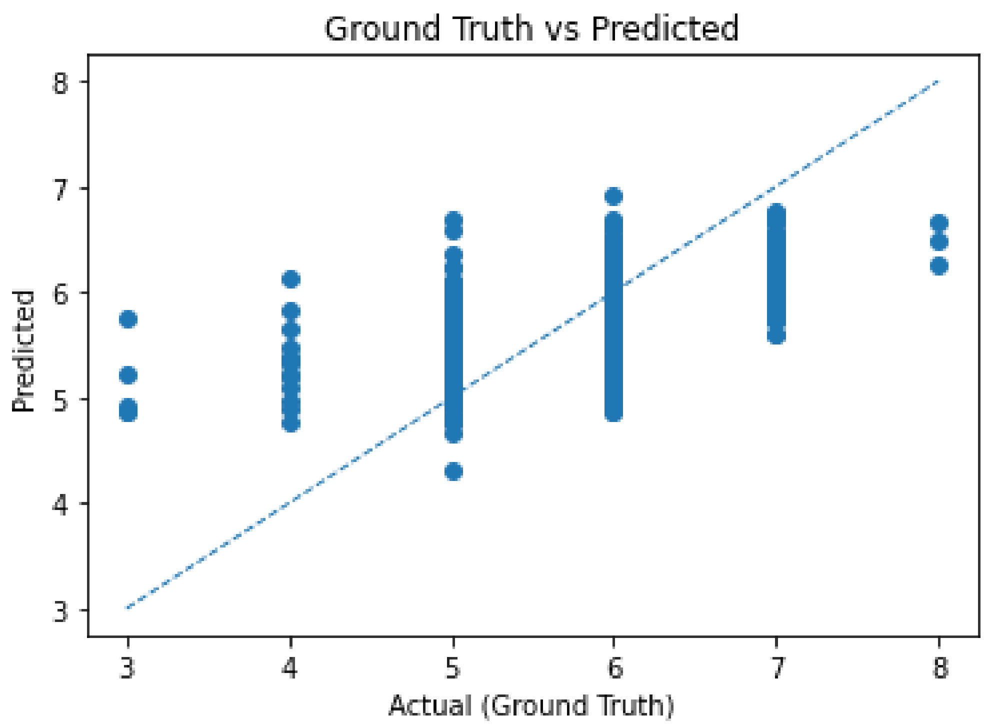
[179]:
```
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
```

[180]:
```
# The alpha parameter may be tuned with in the Ridge() object
model = Ridge(normalize=False, copy_X=True)

model.fit(X_train, y_train)
y_hat = model.predict(X_test)

R2_train = model.score(X_train, y_train)
R2_test = model.score(X_test, y_test)

mse = mean_squared_error(y_test, y_hat)

model, R2_train, R2_test, mse
```

[180]:
```
(Ridge(), 0.36559254798849194, 0.336569353190854, 0.4115775151982)
```

[181]:
```
fig, ax = plt.subplots()
ax.scatter(y_test, y_hat)
ax.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', lw=1)
ax.set_xlabel('Actual (Ground Truth)')
ax.set_ylabel('Predicted')
ax.set_title("Ground Truth vs Predicted")
```

[181]:
```
Text(0.5, 1.0, 'Ground Truth vs Predicted')
```



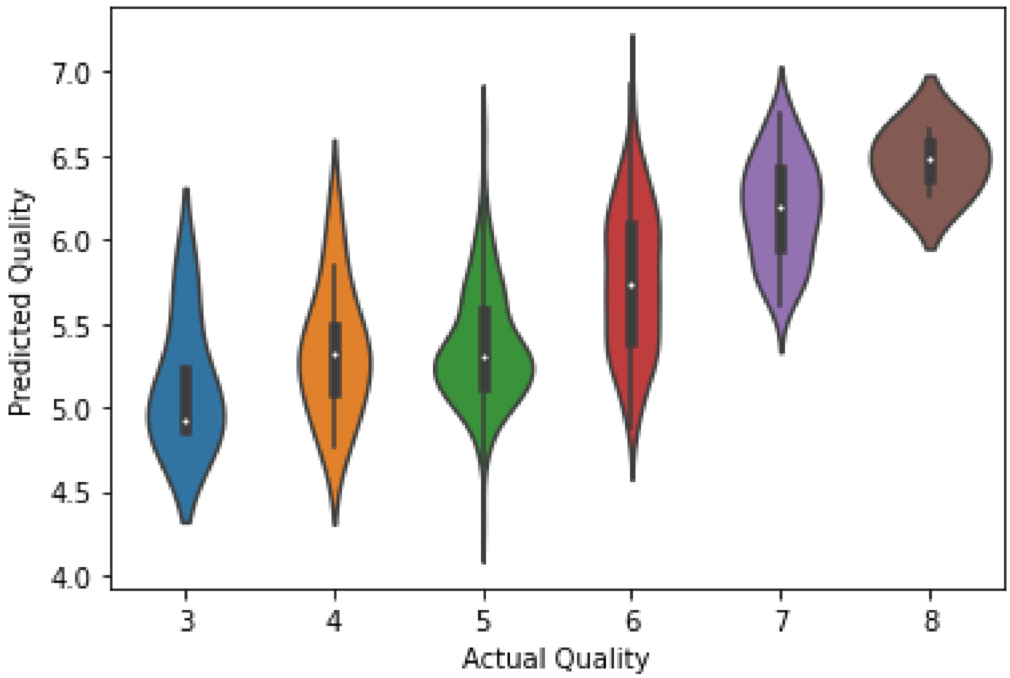

[182]:
```
df2 = pd.DataFrame({'Actual Quality': y_test, 'Predicted Quality': y_hat[:
 ↪len(y_test)]})
df2.head()
```

[182]:

| | Actual Quality | Predicted Quality |
|---|---|---|
| 145 | 5 | 4.935313 |
| 345 | 5 | 5.432258 |
| 603 | 6 | 5.334757 |
| 319 | 6 | 5.365232 |
| 1544 | 7 | 6.355306 |

[183]:
```
sns.violinplot(x="Actual Quality", y="Predicted Quality", data=df2)
```

[183]: <AxesSubplot:xlabel='Actual Quality', ylabel='Predicted Quality'>




### 0.1.4 Lasso Regression

[184]:
```
model = Lasso(alpha=.1,normalize=False, copy_X=True) # The alpha parameter may␣
↪be experimented with

model.fit(X_train, y_train)
y_hat = model.predict(X_test)

R2_train = model.score(X_train, y_train)
R2_test = model.score(X_test, y_test)

mse = mean_squared_error(y_test, y_hat)

model, R2_train, R2_test, mse
```

[184]:
```
(Lasso(alpha=0.1),
 0.3121283925416296,
 0.29084156343493683,
 0.43994601185081295)
```

[185]:
```
X_test
```

[185]:
```
            0         1         2         3         4         5         6  \
145 -0.126188  0.794282  1.432803 -0.524166  0.627696  1.542054  2.874627
345 -0.758172  0.878080 -1.391472 -0.453218 -0.434990  2.307100  0.502727
603  2.803917 -0.378878  1.278752 -0.240375 -0.349975 -0.370562 -0.348724
```

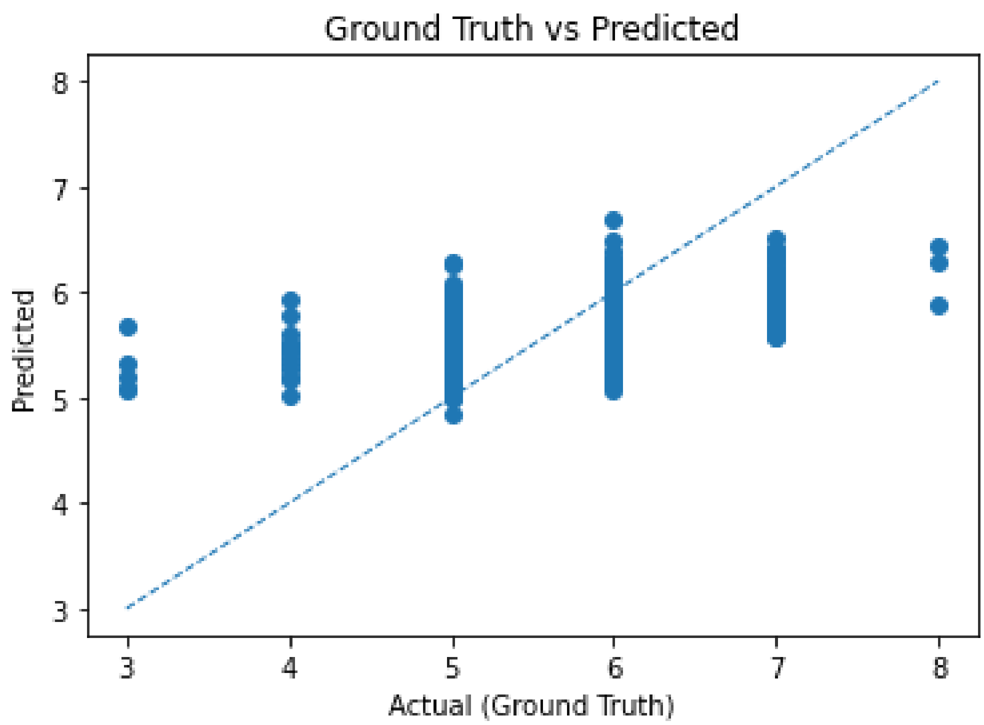
```
319   0.735607  1.352930 -0.775267  0.256260 -0.116184  1.350792  0.837226
1544  0.046171 -0.881661  0.816598 -0.169427 -0.520005 -0.370562 -0.835267
...        ...       ...       ...       ...       ...       ...       ...
666  -0.011282 -0.211283  0.457144 -0.524166  2.859337 -0.944346 -0.926494
1577 -1.217796  0.961877 -0.621215  1.817111 -0.243707 -0.274931 -0.591995
120  -0.585813  3.028873 -0.929318 -0.595114  1.924173 -0.561823  1.293361
824  -0.700719 -0.267148  0.046341  0.185312 -0.413736 -0.944346 -0.926494
1472 -0.413454 -0.993390  1.689555  0.043416 -0.307468  0.681377 -0.075043

            7         8         9        10
145   0.028261 -0.914312 -0.225128 -0.960246
345   0.611276  1.871778  0.896120 -0.490910
603   2.042313 -1.367861 -0.579207 -1.335715
319   1.008786 -0.072005 -0.107102 -0.021574
1544 -0.660757 -0.914312  0.896120  0.729364
...        ...       ...       ...       ...
666   0.664277 -0.849519 -0.343154 -0.866379
1577 -0.279147  1.483021 -0.343154  1.386435
120  -0.289747 -0.072005 -0.520193 -1.335715
824   0.038861 -0.460762 -0.756246 -0.115441
1472 -0.098943  0.446337  0.778094  0.635497

[480 rows x 11 columns]
```

[186]:

```
fig, ax = plt.subplots()
ax.scatter(y_test, y_hat)
ax.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', lw=1)
ax.set_xlabel('Actual (Ground Truth)')
ax.set_ylabel('Predicted')
ax.set_title("Ground Truth vs Predicted")
```

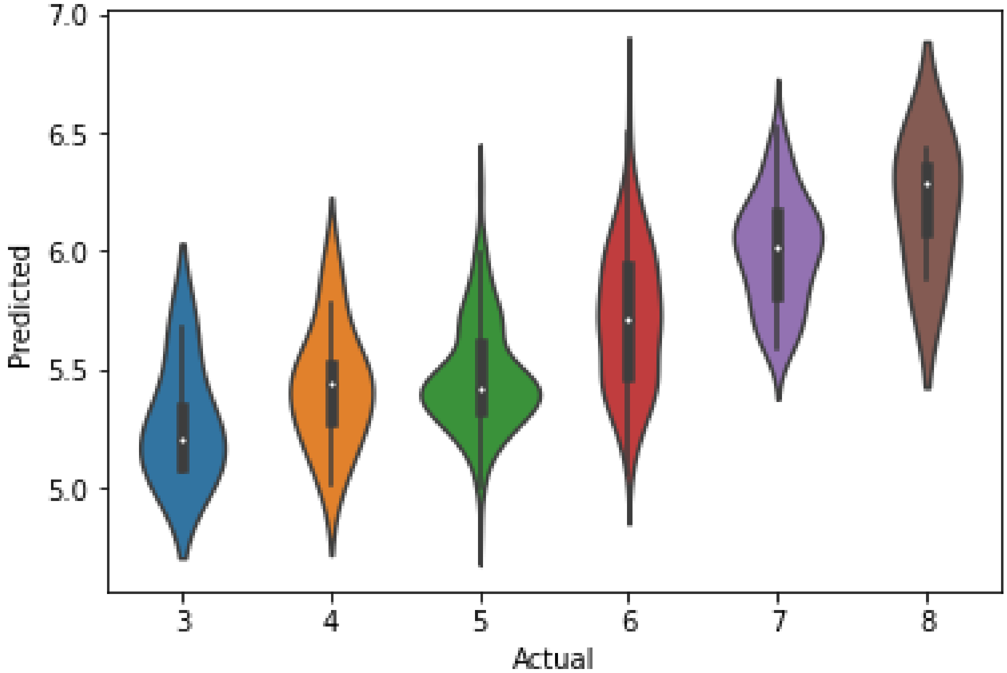
[186]:

```
Text(0.5, 1.0, 'Ground Truth vs Predicted')
```




[187]:
```
import seaborn as sns
datatoplot =  pd.DataFrame(
    {'Actual': y_test,
     'Predicted': y_hat
    })

sns.violinplot(x="Actual", y="Predicted", data=datatoplot)
```

[187]: `<AxesSubplot:xlabel='Actual', ylabel='Predicted'>`



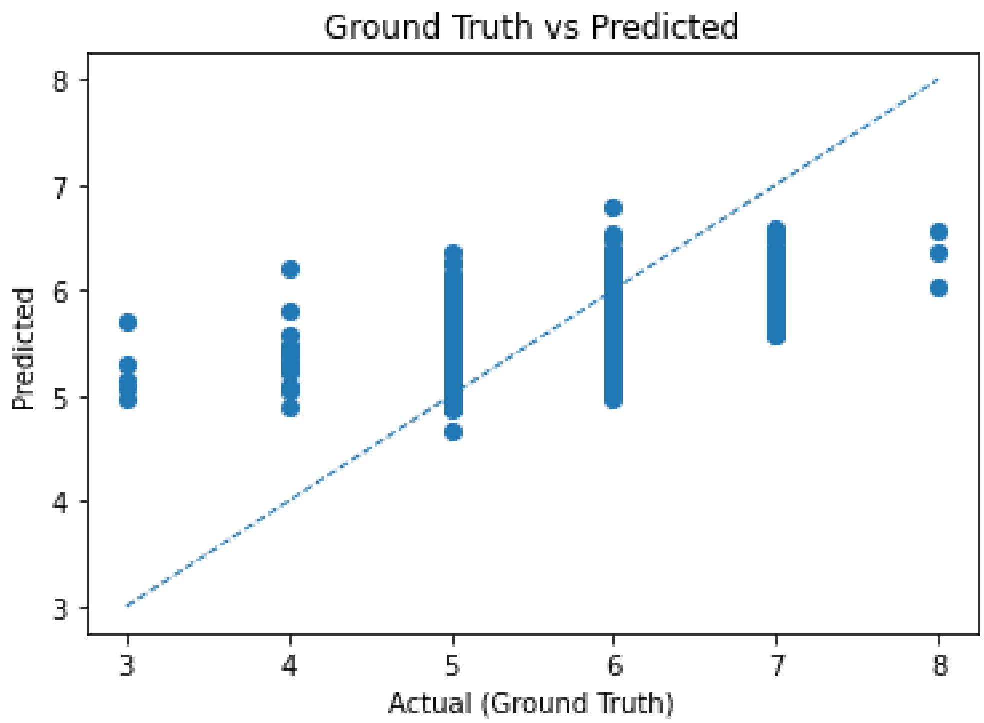

### 0.1.5 ElasticNet Regression

[188]:
```
model = ElasticNet(alpha=.1)

model.fit(X_train, y_train)
y_hat = model.predict(X_test)

R2_train = model.score(X_train, y_train)
R2_test = model.score(X_test, y_test)

mse = mean_squared_error(y_test, y_hat)

model, R2_train, R2_test, mse
```

[188]:
```
(ElasticNet(alpha=0.1),
 0.33850662820818755,
 0.31060187812668194,
 0.42768715516476874)
```

[189]:
```
fig, ax = plt.subplots()
ax.scatter(y_test, y_hat)
ax.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', lw=1)
ax.set_xlabel('Actual (Ground Truth)')
ax.set_ylabel('Predicted')
ax.set_title("Ground Truth vs Predicted")
```

[189]: Text(0.5, 1.0, 'Ground Truth vs Predicted')




### 0.1.6 Decision Tree Regression

[190]:
```
from sklearn.tree import DecisionTreeRegressor
```

[191]:
```
# There are many parameters that may be tuned here in the␣
↪DecisionTreeRegressor() object.
# The technique is shown in the tutorial for Assignment 7 using GridSearchCV

dtr= DecisionTreeRegressor()
dtr.fit(X_train,y_train)

y_pred = dtr.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred)

y_pred_train = dtr.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)

test_mse, train_mse, dtr.score(X_test, y_test)
```

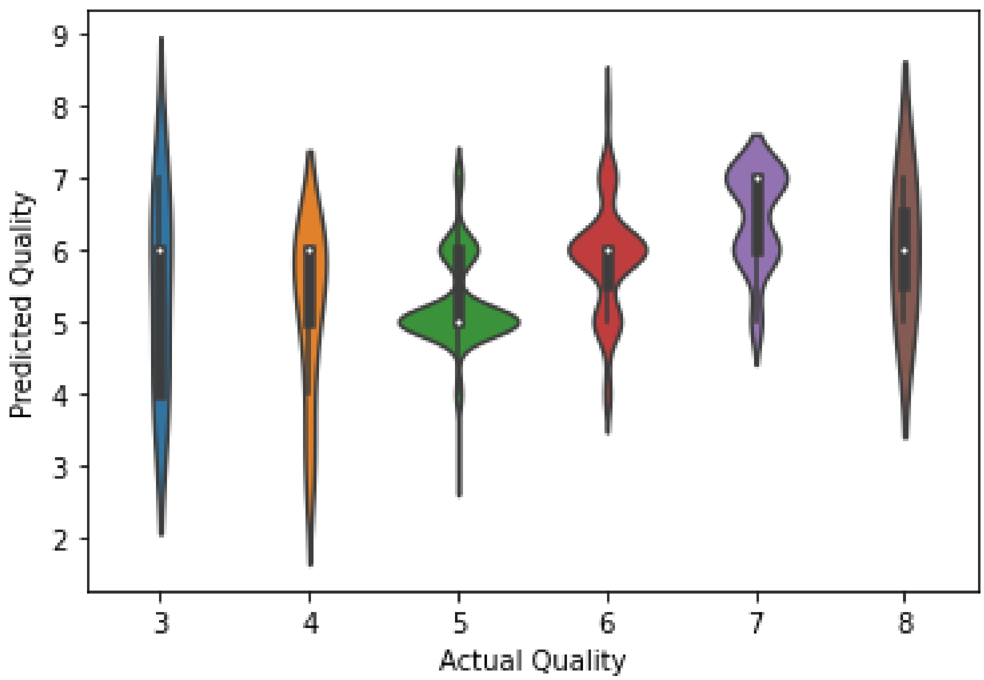
[191]: (0.6854166666666667, 0.0, -0.10483786336446643)

[192]:
```
df2 = pd.DataFrame({'Actual Quality': y_test, 'Predicted Quality': y_pred[:
↪len(y_test)]})
df2.head()
```

[192]:
```
      Actual Quality  Predicted Quality
145                5                5.0
345                5                6.0
603                6                6.0
319                6                6.0
1544               7                6.0
```

[193]:
```
sns.violinplot(x="Actual Quality", y="Predicted Quality", data=df2)
```

[193]:
```
<AxesSubplot:xlabel='Actual Quality', ylabel='Predicted Quality'>
```




### 0.1.7 Random Forest Regression

[194]:
```
from sklearn.ensemble import RandomForestRegressor
```

[195]:
```
rf = RandomForestRegressor(n_estimators = 1000)

rf.fit(X_train, y_train)
y_hat = rf.predict(X_test)

errors = abs(y_hat - y_test)
acc = 1 - errors
```

```
rf.score(X_test, y_test), np.mean(acc)
```

[195]:
```
(0.42797210032532274, 0.5760520833333339)
```

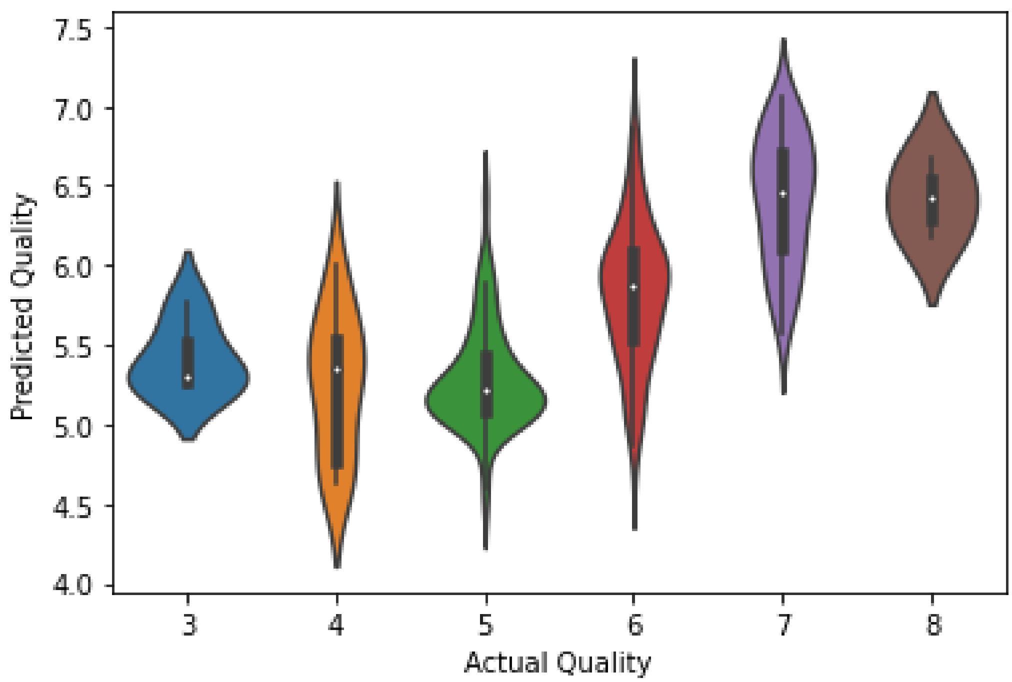
[196]:
```
df2 = pd.DataFrame({'Actual Quality': y_test, 'Predicted Quality': y_hat[:
 ↪len(y_test)]})
df2.head()
```

[196]:

| | Actual Quality | Predicted Quality |
|---|---|---|
| 145 | 5 | 5.052 |
| 345 | 5 | 5.281 |
| 603 | 6 | 5.584 |
| 319 | 6 | 5.806 |
| 1544 | 7 | 6.618 |

[197]:
```
sns.violinplot(x="Actual Quality", y="Predicted Quality", data=df2)
```

[197]:
```
<AxesSubplot:xlabel='Actual Quality', ylabel='Predicted Quality'>
```




### 0.1.8 Support Vector Regression

[198]:
```
from sklearn.svm import SVR
```

[199]:
```
svr = SVR(kernel='linear')

svr.fit(X_train, y_train)
```

```
y_hat = svr.predict(X_test)

print(svr)
print(svr.score(X_test, y_test))
print(svr.score(X_test, y_test))
print(r2_score(y_test,y_hat))
```

```
SVR(kernel='linear')
0.33160148252668487
0.33160148252668487
0.33160148252668487
```

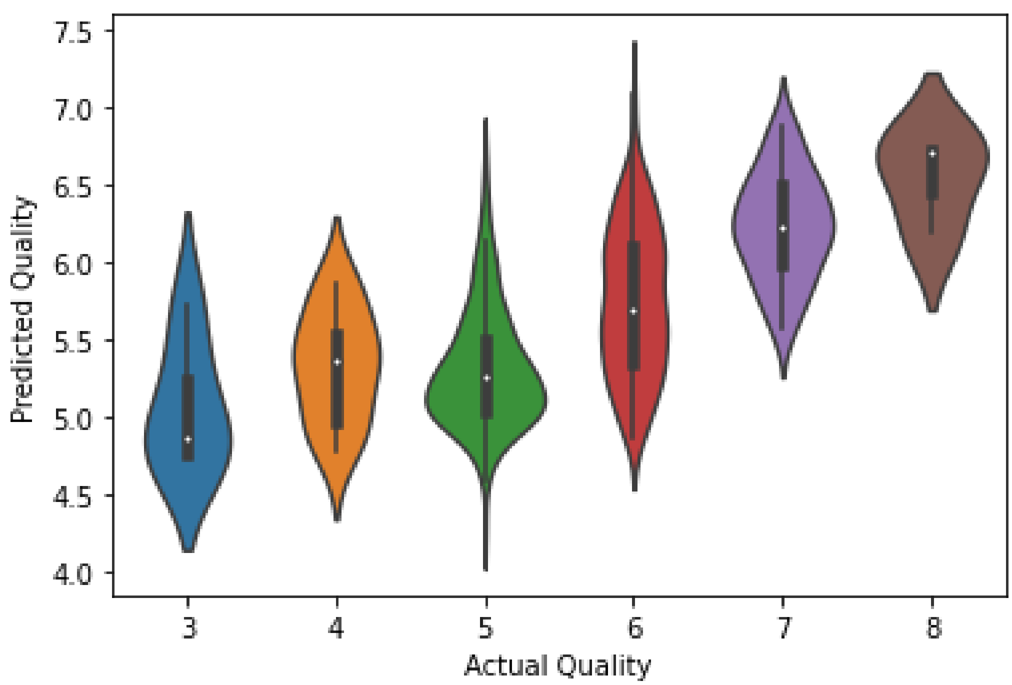
[200]:
```
df2 = pd.DataFrame({'Actual Quality': y_test, 'Predicted Quality': y_hat[:
↪len(y_test)]})
df2.head()
```

[200]:

| | Actual Quality | Predicted Quality |
|---|---|---|
| 145 | 5 | 4.777662 |
| 345 | 5 | 5.373264 |
| 603 | 6 | 5.290415 |
| 319 | 6 | 5.372867 |
| 1544 | 7 | 6.326596 |

[201]:
```
sns.violinplot(x="Actual Quality", y="Predicted Quality", data=df2)
```

[201]: <AxesSubplot:xlabel='Actual Quality', ylabel='Predicted Quality'>



```
[ ]:
```