In this examination of data we will analyze data on red wine and use three different machine learning algorithms to predict if a particular wine is good or bad.

The three models we will use will be: Random Forrest Classification model Multi-Layer Perceptron Classificaiton Model (Artificial Neural Network) Support Vector Classification Model

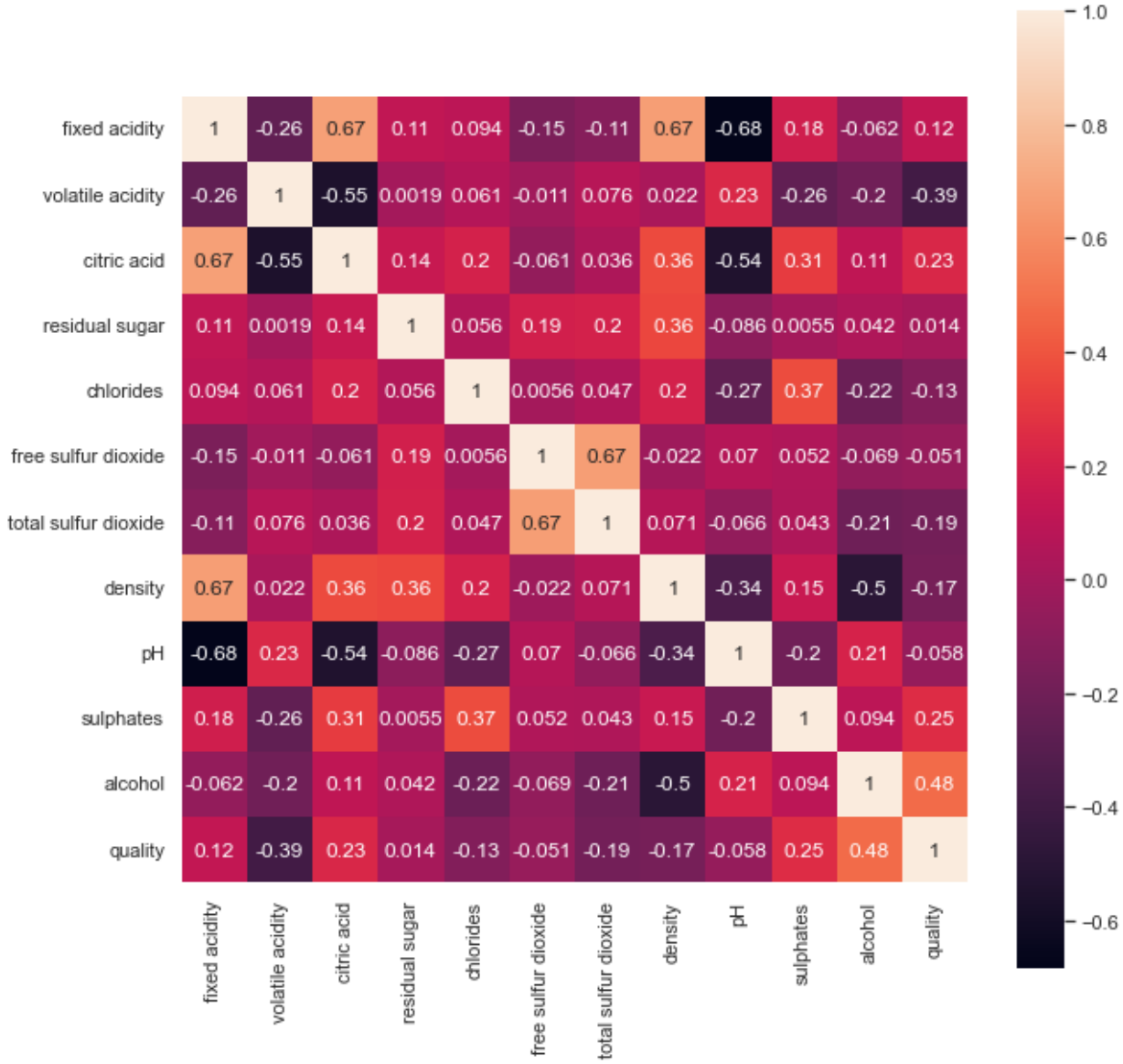
The data originally has 11 feature columns that represent different attributes of the wine and 1 column called quality that we use as our feature that we are trying to predict.

In [20]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import pyplot
sns.set(style='white', context='notebook', palette='deep')
from scipy.stats import pearsonr


from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix


%config Completer.use_jedi = False
```

In [21]:
```
columns=["fixed acidity","volatile acidity","citric acid","residual sugar","chl
wine = pd.read_csv(r'/Users/jonnywerthman/Desktop/Dev_stuff/417-machine learnir
wine.quality.value_counts()
```

Out[21]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [22]:
```
plt.figure(figsize=(10, 10))
sns.heatmap(wine.corr(method='pearson'), annot=True, square=True)
plt.show()

print('Correlation of different features of red wine dataset with quality:')
for i in wine.columns:
  corr, _ = pearsonr(wine[i], wine['quality'])
  print('%s : %.4f' %(i,corr))
```




```
Correlation of different features of red wine dataset with quality:
fixed acidity : 0.1241
volatile acidity : -0.3906
citric acid : 0.2264
residual sugar : 0.0137
chlorides : -0.1289
free sulfur dioxide : -0.0507
total sulfur dioxide : -0.1851
density : -0.1749
pH : -0.0577
sulphates : 0.2514
alcohol : 0.4762
quality : 1.0000
```

In this step we went created a heatmap to visualize all of the correlations each feature had with one another.

We then printed out the correlation between each feature and the quality because that is the result feature, and that correlation is the relationship we were interested in.

We then looked at which features correlated the least with the quality in hopes that we could drop those features and get a more accurate prediction of the quality of wine.

We also didn't want to go through and drop anything without reason, even if it might have caused our model to be more accurate, because that would be bad practice and unethical.

Below is the line where we dropped the features we thought were least useful.

In [23]:
```
wine = wine.drop('residual sugar', axis=1)
wine = wine.drop('fixed acidity', axis=1)
wine = wine.drop('free sulfur dioxide', axis=1)
wine
```

Out[23]:

| | volatile acidity | citric acid | chlorides | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.700 | 0.00 | 0.076 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| **1** | 0.880 | 0.00 | 0.098 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.8 | 5 |
| **2** | 0.760 | 0.04 | 0.092 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.8 | 5 |
| **3** | 0.280 | 0.56 | 0.075 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.8 | 6 |
| **4** | 0.700 | 0.00 | 0.076 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1594** | 0.600 | 0.08 | 0.090 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 | 5 |
| **1595** | 0.550 | 0.10 | 0.062 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 | 6 |
| **1596** | 0.510 | 0.13 | 0.076 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 | 6 |
| **1597** | 0.645 | 0.12 | 0.075 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 | 5 |
| **1598** | 0.310 | 0.47 | 0.067 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 | 6 |

1599 rows × 9 columns

In [24]:
```
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group_names)
wine['quality'] = wine['quality'].map({'bad' : 0, 'good' : 1})
wine.head(15)
```

Out[24]:

| | volatile acidity | citric acid | chlorides | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.700 | 0.00 | 0.076 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 0 |
| **1** | 0.880 | 0.00 | 0.098 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 | 0 |
| **2** | 0.760 | 0.04 | 0.092 | 54.0 | 0.9970 | 3.26 | 0.65 | 9.8 | 0 |
| **3** | 0.280 | 0.56 | 0.075 | 60.0 | 0.9980 | 3.16 | 0.58 | 9.8 | 0 |
| **4** | 0.700 | 0.00 | 0.076 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 0 |
| **5** | 0.660 | 0.00 | 0.075 | 40.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 0 |
| **6** | 0.600 | 0.06 | 0.069 | 59.0 | 0.9964 | 3.30 | 0.46 | 9.4 | 0 |
| **7** | 0.650 | 0.00 | 0.065 | 21.0 | 0.9946 | 3.39 | 0.47 | 10.0 | 1 |
| **8** | 0.580 | 0.02 | 0.073 | 18.0 | 0.9968 | 3.36 | 0.57 | 9.5 | 1 |
| **9** | 0.500 | 0.36 | 0.071 | 102.0 | 0.9978 | 3.35 | 0.80 | 10.5 | 0 |
| **10** | 0.580 | 0.08 | 0.097 | 65.0 | 0.9959 | 3.28 | 0.54 | 9.2 | 0 |
| **11** | 0.500 | 0.36 | 0.071 | 102.0 | 0.9978 | 3.35 | 0.80 | 10.5 | 0 |
| **12** | 0.615 | 0.00 | 0.089 | 59.0 | 0.9943 | 3.58 | 0.52 | 9.9 | 0 |
| **13** | 0.610 | 0.29 | 0.114 | 29.0 | 0.9974 | 3.26 | 1.56 | 9.1 | 0 |
| **14** | 0.620 | 0.18 | 0.176 | 145.0 | 0.9986 | 3.16 | 0.88 | 9.2 | 0 |

Going forward we plan to use categorical Machine Learning Algorithms, so we decided that any wine with a quality rating better than 6.5 was considered a good wine.

We then split all of the wines into a category of good or bad based on that metric (quality feature greater or lesser than 6.5).

Then we assigned binary values to the wines so that bad wines had a value of 0 and good wines had a value of 1.

This way we could categorically analyze the data and make predictions.

It is significantly easier to accurately predict categorical data than it is to predict quantitative data when there are this many factors, which is why we decided to change the form of the data in this way.

In [25]:
```
X = wine.drop('quality', axis = 1)
y = wine['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, rando


sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In this cell all we did was randomly split the data into training and testing groups.

we decided that 20 percent of the data would be in the testing groups because we felt that that was enough to verify our results without skewing the data in an inacurrate way.

In [26]:
```
import itertools
def plot_confusion_matrix(cm, classes, normalize = False,
                          title='Confusion matrix',
                          cmap=plt.cm.Greens): # can change color
    plt.figure(figsize = (5, 5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, size = 24)
    plt.colorbar(aspect=4)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, size = 14)
    plt.yticks(tick_marks, classes, size = 14)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    # Label the plot
    for i, j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
            fontsize = 20,
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label', size = 18)
    plt.xlabel('Predicted label', size = 18)
```

The function above is a function that we borrowed from class in order to accurately visualize our findings in a confusion matrix.

Doing this gave us a really clear representation of our accuracy as well as our type 1 and type 2 errors.

The code directly below this cell is our first actual algorithm ran on the data set.

We ran a Random Forest Classifier and our initial Hypothesis is that of the three algorithms this one would fit the best because it would pick out the features that had the largest impact and give those features the highest level of impact on the predictions.

We ran the algorithm with 100 different instances meaning that it would randomly generate 100 trees and as it got closer to the actual classification it would create better and better trees.

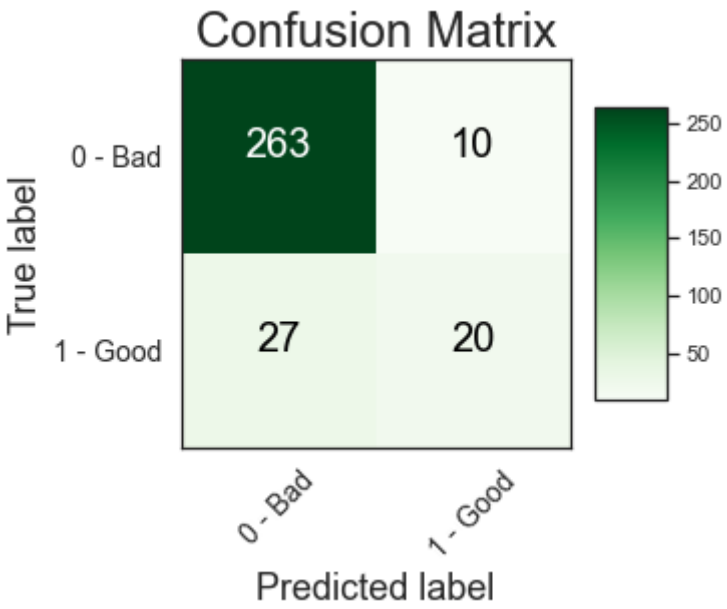
We chose to run it 100 times because we wanted to get as accurate as possible without overfitting the data.

All the error calculations for each algorithm is at the end.

In [27]:
```
rfc = RandomForestClassifier(n_estimators=100, criterion='gini', random_state=(
rfc.fit(X_train, y_train)
pred_rf = rfc.predict(X_test)
Y_compare_rfc = pd.DataFrame({'Actual' : y_test, 'Predicted' : pred_rf})
print(Y_compare_rfc.head())
print('\nConfussion matrix:')
```

```
print(confusion_matrix(y_test, pred_rf))


cm = confusion_matrix(y_test, pred_rf)
plot_confusion_matrix(cm, classes = ['0 - Bad', '1 - Good'],
                      title = 'Confusion Matrix')
```

```
     Actual  Predicted
803       0          0
124       0          0
350       0          0
682       0          0
1326      0          0

Confussion matrix:
[[263  10]
 [ 27  20]]
```




Below is our Multi-Layer Perceptron Classifier algorithm. In essence it is an Artificial Neural Network.

In our ANN implementation we gave gave the algorithm a hidden layer size of 25x25 meaning there are 25 layers each with 25 nodes.

We also decided to give each node a weight of 1e-5 because we needed it to be small so that the ANN wouldn't be wildly changing predictions everytime it got an answer wrong. This way it gradually approaches the correct answer.

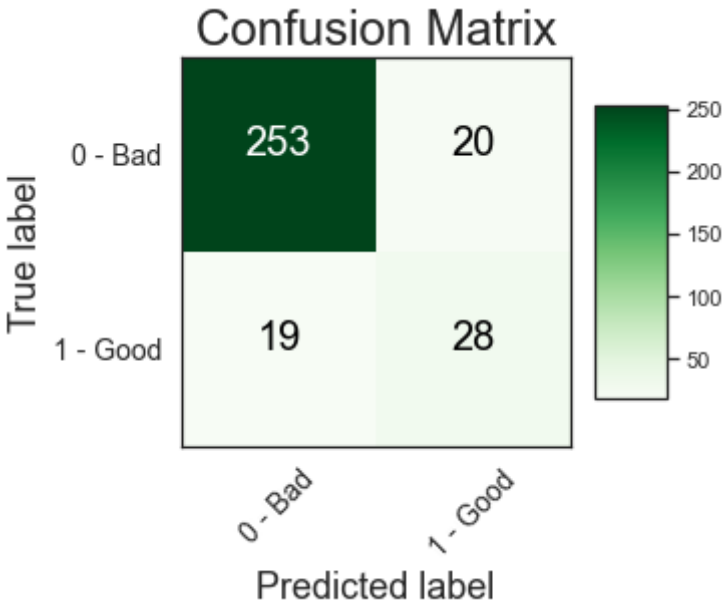
I am sure there is a way to visualize and calculate how many hidden nodes you should have so that your algorithm stops at the point of maximum accuracy without overfitting, however this was above our level of knowledge on the concept, so we tested a few layer sizes and alpha values and this is what we came up with.

In [28]:
```
ANN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(25,25,), ra
ANN.fit(X_train, y_train)
pred_ANN = ANN.predict(X_test)
Y_compare_ANN = pd.DataFrame({'Actual' : y_test, 'Predicted' : pred_ANN})
```

```
print(Y_compare_ANN.head())
print('\nConfussion matrix:')
print(confusion_matrix(y_test, pred_ANN))


cm = confusion_matrix(y_test, pred_ANN)
plot_confusion_matrix(cm, classes = ['0 - Bad', '1 - Good'],
                      title = 'Confusion Matrix')
```

```
     Actual  Predicted
803       0          0
124       0          0
350       0          0
682       0          0
1326      0          0

Confussion matrix:
[[253  20]
 [ 19  28]]
```

```
/Users/jonnywerthman/opt/miniconda3/envs/cse468/lib/python3.7/site-packages/sk
learn/neural_network/_multilayer_perceptron.py:549: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```




Lastly, We implemented a Support Vector model to analyze and predict the data.

In this implementation we decided to give the polynomial function a degree of 10, which at the time seemed fair because of our understanding of the model.

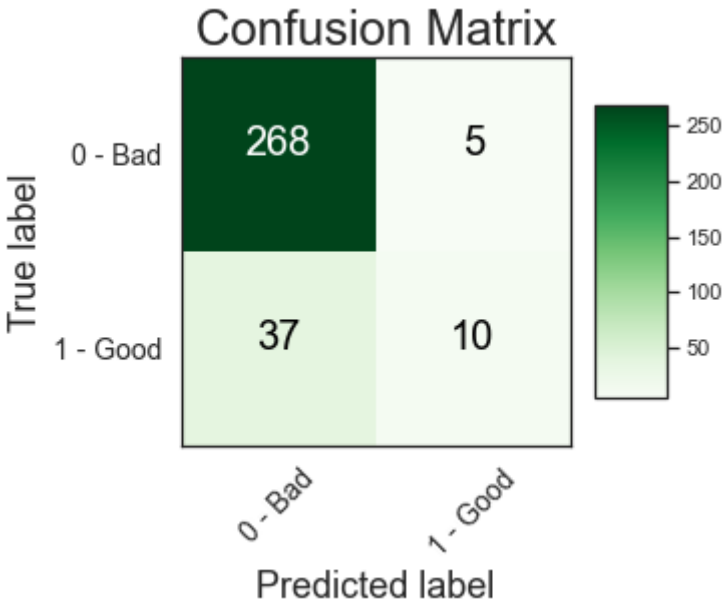
In most cases using a SVC with a degree greater than 15 is subjecting the algorithm to overfitting, which is why we chose 10.

In hindsight 10 might have been too high of a number because it would allow for a lot more variance than one would want when predicting wine, but it did give a very accurate

prediction of the wine regardless.

In [29]:
```
svm = SVC(kernel='rbf', degree=10, random_state=1,)
svm.fit(X_train, y_train)
pred_svm = svm.predict(X_test)
Y_compare_svm = pd.DataFrame({'Actual' : y_test, 'Predicted' : pred_svm})
print(Y_compare_svm.head())
print('\nConfussion matrix:')
print(confusion_matrix(y_test, pred_svm))


cm = confusion_matrix(y_test, pred_svm)
plot_confusion_matrix(cm, classes = ['0 - Bad', '1 - Good'],
                      title = 'Confusion Matrix')
```

```
     Actual  Predicted
803       0          0
124       0          0
350       0          0
682       0          0
1326      0          0

Confussion matrix:
[[268   5]
 [ 37  10]]
```




In [30]:
```
modelNames = ['Random Forrest', 'Artificial Neural Network', 'Support Vector Mc
modelClassifiers = [rfc, ANN, svm]
models = pd.DataFrame({'modelNames' : modelNames, 'modelClassifiers' : modelCla
counter=0
score=[]
for i in models['modelClassifiers']:
  accuracy = cross_val_score(i, X_train, y_train, scoring='accuracy', cv=10)
  print('Accuracy of %s Classification model is %.2f' %(models.iloc[counter,0],
  score.append(accuracy.mean())
  counter+=1
```

```
Accuracy of Random Forrest Classification model is 0.91
```

```
/Users/jonnywerthman/opt/miniconda3/envs/cse468/lib/python3.7/site-packages/sk
learn/neural_network/_multilayer_perceptron.py:549: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
/Users/jonnywerthman/opt/miniconda3/envs/cse468/lib/python3.7/site-packages/sk
learn/neural_network/_multilayer_perceptron.py:549: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
Accuracy of Artificial Neural Network Classification model is 0.87
Accuracy of Support Vector Model Classification model is 0.89
```

To assess our models we used sklearn's cross evaluation scoring method, which compares the accuracy of the findings against multiple training instances of itself to find its rate of innaccuracy. For more detail on how it works look at: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

And as you can tell our hypothesis about the prediction accuracy of Random Forrest Classification was correct.

All of the results yielded very good results and would be great predicters in telling if this set of wine is good or not.

However, I am not convinced that this would hold true for other wines. It may hold true for other red wines becuase that is the type we ran our trials on, but I don't think it would work so well on white wines or bubbly wines.

In [ ]: