Question 1: Effect of Normalization, FeatureExtraction and Distance Metrics

1.1 Tasks

1.1.1 Train/Test Data Split

In [108]:
```
# loading the librarries libraries
import numpy as np
import pandas as pd
import random
import seaborn as sns
sns.set(style="ticks", color_codes=True)
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [109]:
```
# reading and loading the data
#Columns/Features
D = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlo
L = 'quality'
C = 'color'
DL = D + [L]
DC = D + [C]
DLC = DL + [C]

#Loading Data set
wine_r = pd.read_csv("winequality-red.csv", sep=';')
#Loading Data set
wine_w = pd.read_csv("winequality-white.csv", sep=';')
wine_w= wine_w.copy()
wine_w[C]= np.zeros(wine_w.shape[0])
wine_r[C]= np.ones(wine_r.shape[0])
wine = pd.concat([wine_w,wine_r])


# lets understand the dataframes we have with us right now
print("Lets understand the data shapes : ")
print(" ")
print("number of columns represented in D is :",len(D))
print("shape of wine_w is : ", wine_w.shape)
print("shape of wine_r is : ", wine_r.shape)
print("shape of dataframe wine is :", wine.shape)
print("shape of wine [D] is", wine[D].shape)
print(" ")
print(" ")

# Splitting the dataset into train and test data
X = wine[D]
y_c = wine[C]
y_q = wine[L]
ran = 42
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X, y_c, test_size=0.
X_train_q, X_test_q, y_train_q, y_test_q = train_test_split(X, y_q, test_size=0.

#Lets see the shape of splitted data
print("Shape of dataframes used/created in the split :")
print(" ")
print("shape of X is : ", X.shape)
print("shape of y_c is : ", y_c.shape)
print("shape of X_train_c is :", X_train_c.shape)
print("shape of X_test_c is :", X_test_c.shape)
print("shape of y_train_c is :", y_train_c.shape)
print("shape of y_test_c is :", y_test_c.shape)
print("shape of y_q is : ", y_q.shape)
print("shape of X_train_q is :", X_train_q.shape)
print("shape of X_test_q is :", X_test_q.shape)
print("shape of y_train_q is :", y_train_q.shape)
print("shape of y_test_q is :", y_test_q.shape)
```

```
number of columns represented in D is : 11
shape of wine_w is :  (4898, 13)
```

```
shape of wine_w is :  (4898, 13)
shape of wine_r is :  (1599, 13)
shape of dataframe wine is : (6497, 13)
shape of wine [D] is (6497, 11)


Shape of dataframes used/created in the split :

shape of X is :  (6497, 11)
shape of y_c is :  (6497,)
shape of X_train_c is : (5197, 11)
shape of X_test_c is : (1300, 11)
shape of y_train_c is : (5197,)
shape of y_test_c is : (1300,)
shape of y_q is :  (6497,)
shape of X_train_q is : (5197, 11)
shape of X_test_q is : (1300, 11)
shape of y_train_q is : (5197,)
shape of y_test_q is : (1300,)
```

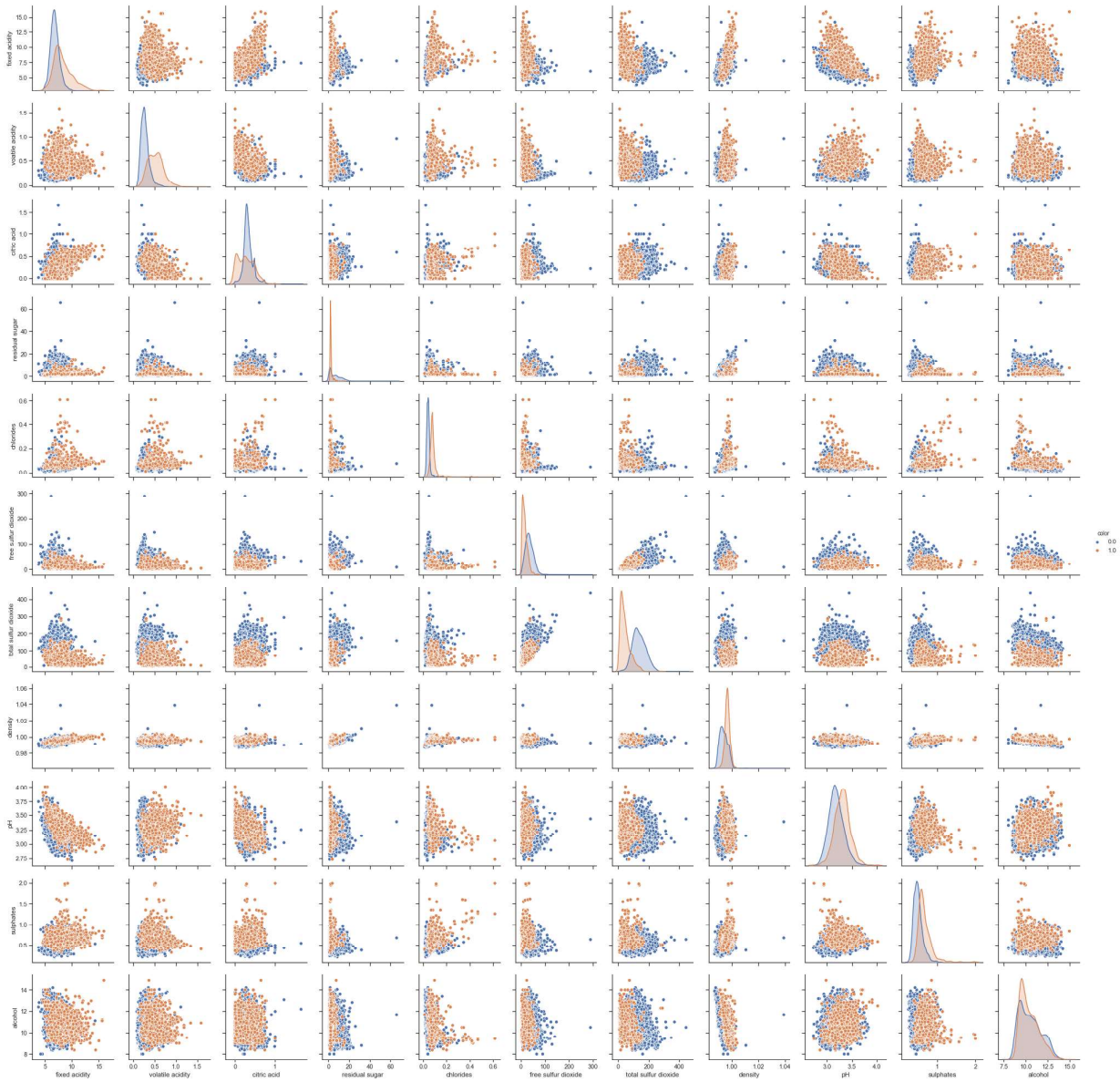
1.1.2 Normalization

In [110]:
```
#importing standard scalar, used for z-score normalization
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler(copy=False, with_mean=True, with_std=True)
X_scaled = wine[D]
X_scaled =scalar.fit_transform(X_scaled)
X_scaled = pd.DataFrame(data = X_scaled, columns = D)
scalar = StandardScaler()
X_train_scaled_c = X_train_c.copy()
X_train_scaled_q = X_train_q.copy()
X_train_scaled_c = scalar.fit_transform(X_train_scaled_c)
X_test_scaled_c = scalar.transform(X_test_c)
X_train_scaled_q = scalar.fit_transform(X_train_scaled_q)
X_test_scaled_q = scalar.transform(X_test_q)
```

Pairplot for non-normalized Data

In [111]:
```
sns.pairplot(wine[DC], vars = wine[DC].columns[:-1], hue = 'color')
```

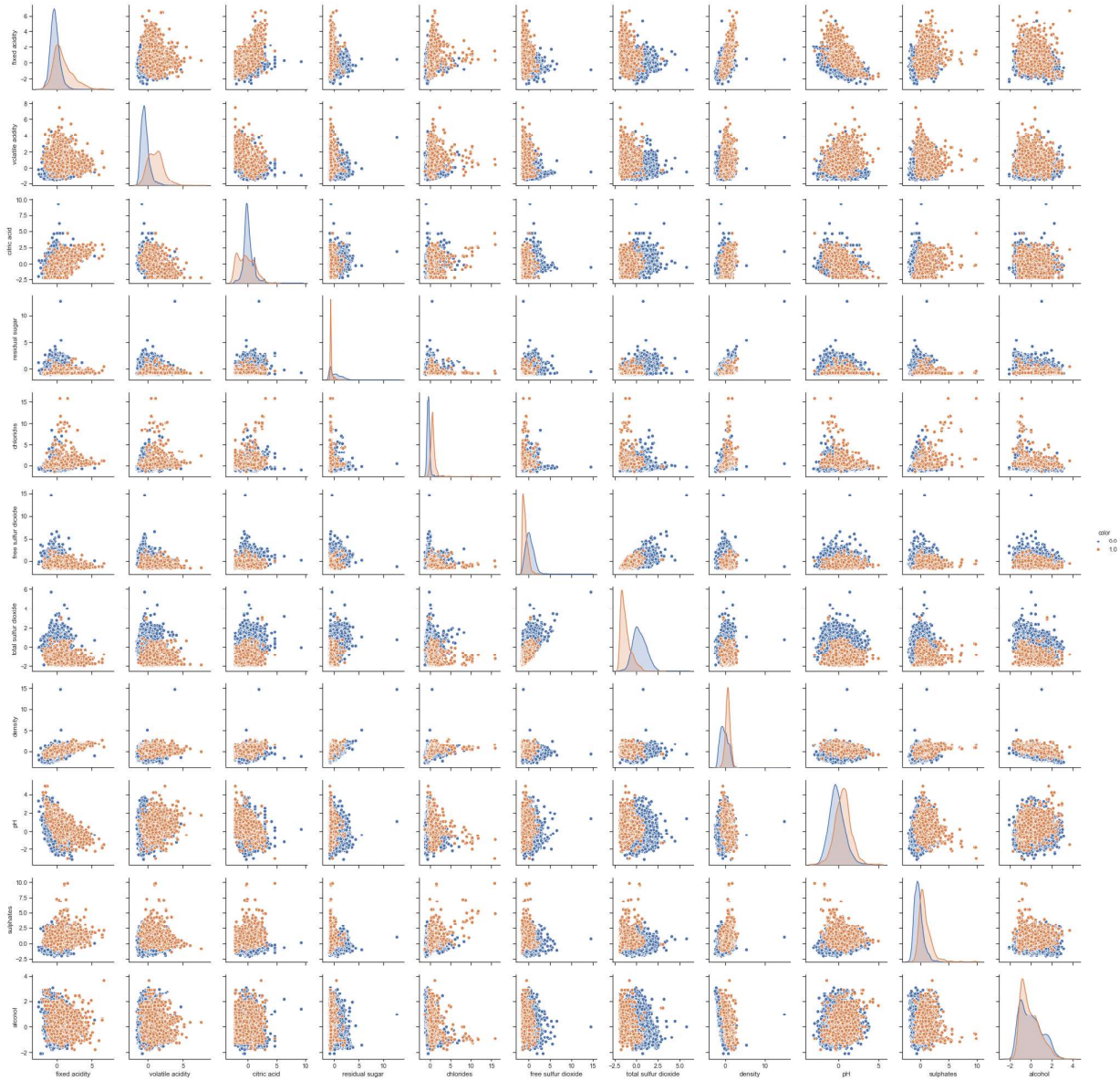
Out[111]:
```
<seaborn.axisgrid.PairGrid at 0x1e204a0f518>
```



Pairplot for normailized Dataset

In [112]:
```
yc_new= y_c.reset_index()
yc_new = yc_new.drop(['index'], axis =1)
X_scaled ['color'] = yc_new
sns.pairplot(X_scaled,vars = X_scaled.columns[:-1], hue = 'color')
```

Out[112]: <seaborn.axisgrid.PairGrid at 0x1e2122ca7f0>



Comparing Pairplot for normalized and non- nonrmalized features

It can be visualized from above plots :

the normalized plots are a bit more symmetric and not tend to look too elliptical compared to non-normalized plots. I believe as the normalized plot, are Z-score normalized and hence they represent the variation of normalized variance among the features, whereas the non-normalized

plot show variation of features among each other. Hence, the non-normalized plot tend to look affected by the actual values (cause of different scales of each feature), and hence do not provide a clear picture of variations of variances amongst different features.

Z-score converts all features to a common scale with an average of zero and standard deviation of one. The average of zero means that it avoids introducing aggregation distortions stemming from differences in feature means.

As different features have different scales, normalizing these features by z-score would make the plots normalized, the plots would still show the relationship between two variables maintaining the dispersion between features but would be less affected by the difference in scales of features.

1.1.3 Classification : Color

KNN Classification for wine color on normalized data with all features

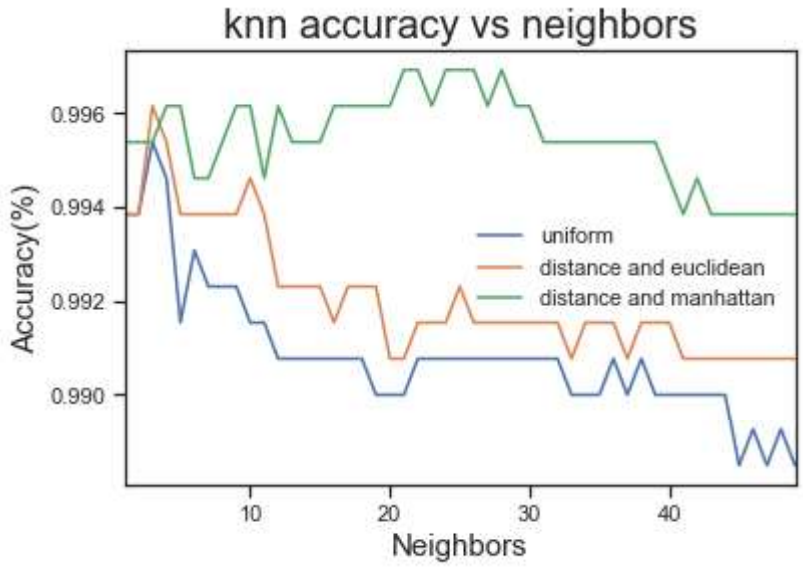
In [113]:
```
## lets work on normalized data for color prediction for the wine
#we must not forget to transform the test data too

scalar = StandardScaler(copy=False, with_mean=True, with_std=True)
X_train_scaled_c = X_train_c.copy()
X_train_scaled_q = X_train_q.copy()
X_train_scaled_c = scalar.fit_transform(X_train_scaled_c)
X_train_scaled_q  = scalar.fit_transform(X_train_scaled_q)



X_test_scaled_c = scalar.transform(X_test_c)
n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_train_scaled_c, y_train_c)
    y_pred_c = neigh.predict(X_test_scaled_c)
    accscore = accuracy_score(y_test_c, y_pred_c)
    acc.at[k,col_names[0]] = accscore


for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_c, y_train_c)
    y_pred_c = neigh.predict(X_test_scaled_c)
    accscore = accuracy_score(y_test_c, y_pred_c)
    acc.at[k,col_names[1]] = accscore


for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_c, y_train_c)
    y_pred_c = neigh.predict(X_test_scaled_c)
    accscore = accuracy_score(y_test_c, y_pred_c)
    acc.at[k,col_names[2]] = accscore


acc.describe()
acc.head()
```

Out[113]:

	uniform	distance and euclidean	distance and manhattan
0	0.000000	0.000000	0.000000
1	0.993846	0.993846	0.995385
2	0.993846	0.993846	0.995385
3	0.995385	0.996154	0.995385
4	0.994615	0.995385	0.996154

In [114]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

Out[114]:
```
Text(0, 0.5, 'Accuracy(%)')
```



Trying diffrent metrics on normalized wine data for color to compare performances (All features are used)

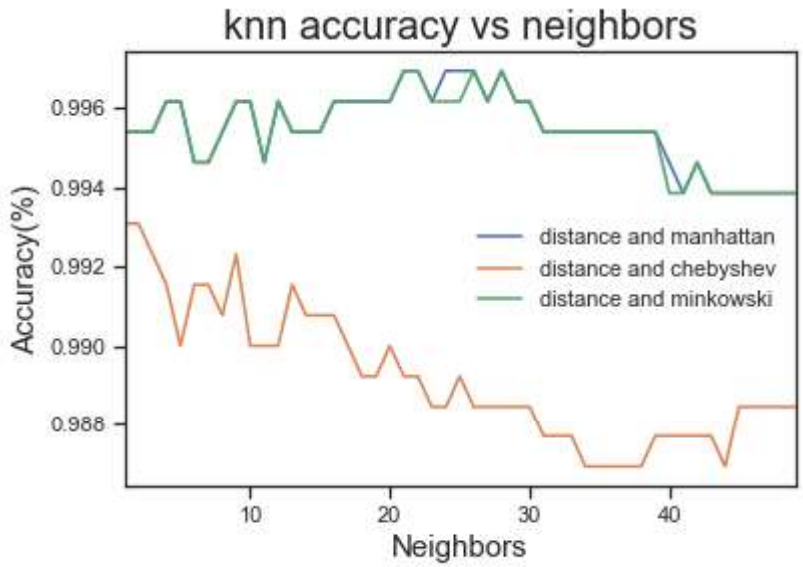
In [185]:
```
# Let us try multiple possilities for normalized datset
n_neighborslist = list(range(1,50))
col_names=['distance and manhattan','distance and chebyshev', 'distance and mink
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_c, y_train_c)
    y_pred_c = neigh.predict(X_test_scaled_c)
    accscore = accuracy_score(y_test_c, y_pred_c)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_c, y_train_c)
    y_pred_c = neigh.predict(X_test_scaled_c)
    accscore = accuracy_score(y_test_c, y_pred_c)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_c, y_train_c)
    y_pred_c = neigh.predict(X_test_scaled_c)
    accscore = accuracy_score(y_test_c, y_pred_c)
    acc.at[k,col_names[2]] = accscore


acc.describe()
acc.head()
```

Out[185]:

	distance and manhattan	distance and chebyshev	distance and minkowski
0	0.000000	0.000000	0.000000
1	0.995385	0.993077	0.995385
2	0.995385	0.993077	0.995385
3	0.995385	0.992308	0.995385
4	0.996154	0.991538	0.996154

In [186]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

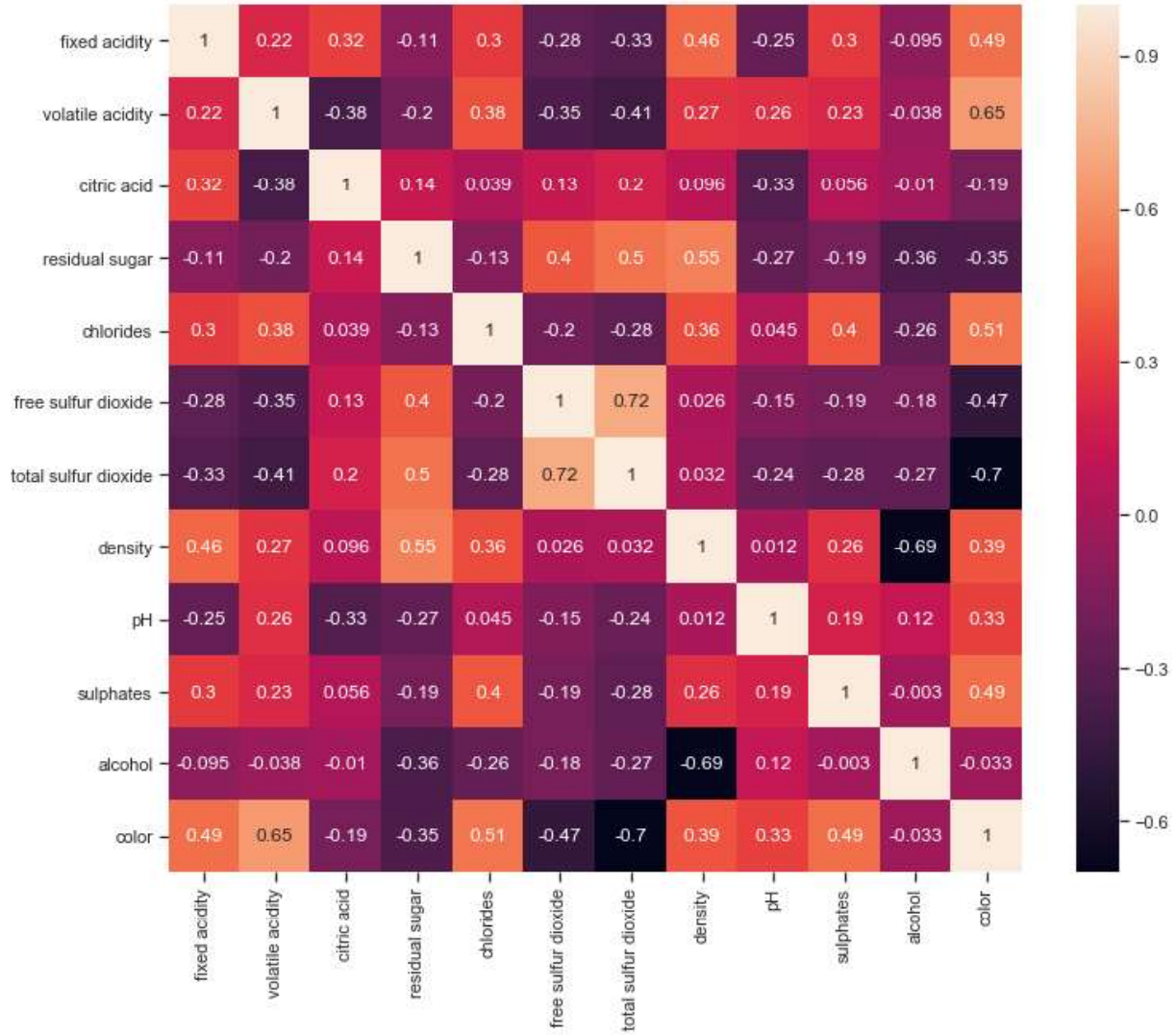
Out[186]:
```
Text(0, 0.5, 'Accuracy(%)')
```



1.1.4 Feature Selection : Color

Using Pearson Correlation coefficient for feature selection

In [117]:
```
# lets see the correlation among features
#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = X_scaled.corr()
sns.heatmap(cor, annot=True)
#cmap=plt.cm.Reds
plt.show()
```



running KNN on normalized data with a subset of features selected using pearson correlation coefficient with color

In [118]:

```
#Selecting the 4 most correlated features
#Total sulpher oxide(-0.7),Volatile acidity(0.65), chloride(0.51), fixed acidity
selected_features_c  = ['total sulfur dioxide','volatile acidity', 'chlorides','
X_sel_c = wine[selected_features_c]
X_sel_train_c, X_sel_test_c, y_sel_train_c, y_sel_test_c = train_test_split(X_se
X_sel_train_scaled_c = X_sel_train_c.copy()
scalar.fit_transform(X_sel_train_scaled_c)
#we must not forget to transform the test data too
X_sel_test_scaled_c = scalar.transform(X_sel_test_c)

n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_sel_train_scaled_c, y_sel_train_c)
    y_sel_pred_c = neigh.predict(X_sel_test_scaled_c)
    accscore = accuracy_score(y_sel_test_c, y_sel_pred_c)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_sel_train_scaled_c, y_sel_train_c)
    y_sel_pred_c = neigh.predict(X_sel_test_scaled_c)
    accscore = accuracy_score(y_sel_test_c, y_sel_pred_c)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_sel_train_scaled_c, y_sel_train_c)
    y_sel_pred_c = neigh.predict(X_sel_test_scaled_c)
    accscore = accuracy_score(y_sel_test_c, y_sel_pred_c)
    acc.at[k,col_names[2]] = accscore



acc.describe()
```

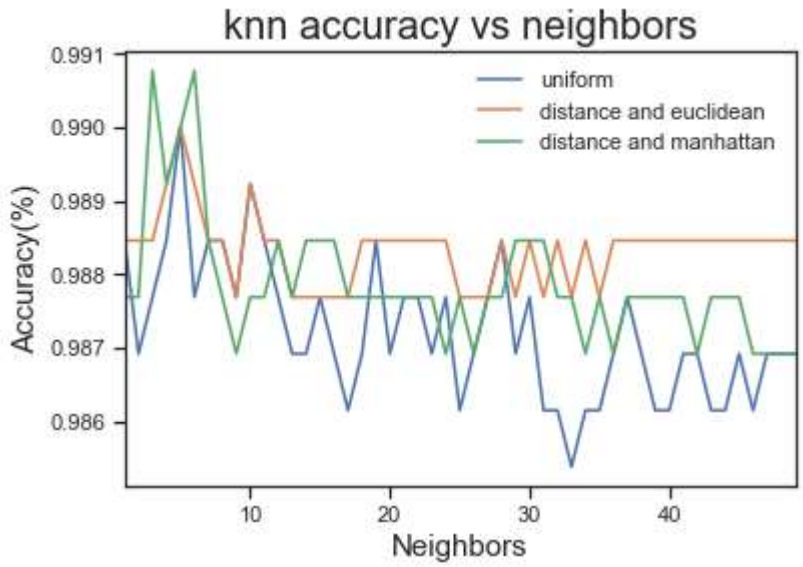
Out[118]:

	uniform	distance and euclidean	distance and manhattan
count	50.000000	50.000000	50.000000
mean	0.967477	0.968569	0.968108
std	0.139617	0.139773	0.139708
min	0.000000	0.000000	0.000000
25%	0.986346	0.987692	0.987692
50%	0.986923	0.988462	0.987692

	uniform	distance and euclidean	distance and manhattan
75%	0.987692	0.988462	0.987692
max	0.990000	0.990000	0.990769

In [119]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

Out[119]:
```
Text(0, 0.5, 'Accuracy(%)')
```



1.1.5 Feature Extraction: Color

1.1.5.1 PCA : Color

Running KNN on normalized and PCA tranformed data with 5 principal components for wine color prediction

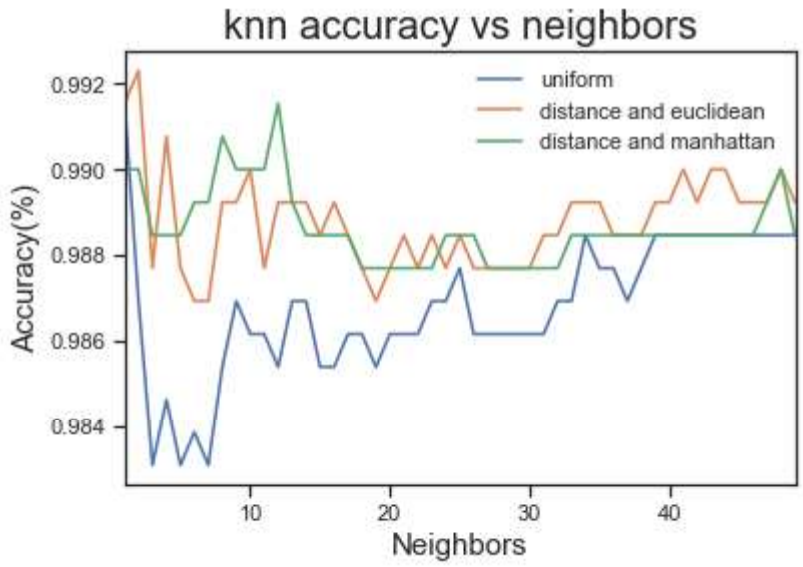
In [120]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 5,random_state = 42)
principalComponents = pca.fit_transform(X_train_scaled_c)
col_name_pca = ["Principal Component " + str(i) for i in range(1,6)]
X_pca_train_c = pd.DataFrame(data = principalComponents, columns = col_name_pca)
y_pca_train_c = y_train_c
X_pca_test_c = pca.transform(X_test_scaled_c)
y_pca_test_c = y_test_c

n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))

#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_pca_train_c, y_pca_train_c)
    y_pca_pred_c = neigh.predict(X_pca_test_c)
    accscore = accuracy_score(y_pca_test_c, y_pca_pred_c)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_pca_train_c, y_pca_train_c)
    y_pca_pred_c = neigh.predict(X_pca_test_c)
    accscore = accuracy_score(y_pca_test_c, y_pca_pred_c)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_pca_train_c, y_pca_train_c)
    y_pca_pred_c = neigh.predict(X_pca_test_c)
    accscore = accuracy_score(y_pca_test_c, y_pca_pred_c)
    acc.at[k,col_names[2]] = accscore



acc.describe()
```

Out[120]:

	uniform	distance and euclidean	distance and manhattan
count	50.000000	50.000000	50.000000
mean	0.967015	0.969000	0.968862
std	0.139557	0.139838	0.139817
min	0.000000	0.000000	0.000000
25%	0.986154	0.987692	0.987885
50%	0.986923	0.988462	0.988462
75%	0.988269	0.989231	0.988462
max	0.991538	0.992308	0.991538

In [121]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

Out[121]:
```
Text(0, 0.5, 'Accuracy(%)')
```



1.1.5.2 LDA : Color

Running KNN on normalized and LDA tranformed data with1 component for wine color prediction

In [122]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
col_name_lda = ["LDA component 1"]
lda = LinearDiscriminantAnalysis(n_components = 1)
ldaComponents = lda.fit_transform(X_train_scaled_c, y_train_c)
X_lda_train_c = pd.DataFrame(data = ldaComponents, columns = col_name_lda)
y_lda_train_c = y_train_c
y_lda_test_c = y_test_c
X_lda_test_c = lda.transform(X_test_scaled_c)
n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))



#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_lda_train_c, y_lda_train_c)
    y_lda_pred_c = neigh.predict(X_lda_test_c)
    accscore = accuracy_score(y_lda_test_c, y_lda_pred_c)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_lda_train_c, y_lda_train_c)
    y_lda_pred_c = neigh.predict(X_lda_test_c)
    accscore = accuracy_score(y_lda_test_c, y_lda_pred_c)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_lda_train_c, y_lda_train_c)
    y_lda_pred_c = neigh.predict(X_lda_test_c)
    accscore = accuracy_score(y_lda_test_c, y_lda_pred_c)
    acc.at[k,col_names[2]] = accscore



acc.describe()
```

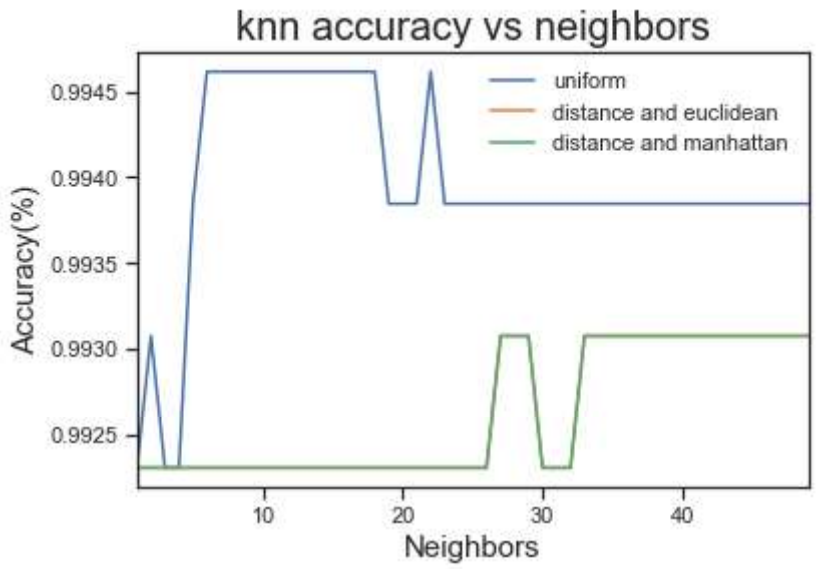
Out[122]:

	uniform	distance and euclidean	distance and manhattan
count	50.000000	50.000000	50.000000
mean	0.974077	0.972769	0.972769
std	0.140568	0.140378	0.140378
min	0.000000	0.000000	0.000000
25%	0.993846	0.992308	0.992308
50%	0.993846	0.992308	0.992308
75%	0.994615	0.993077	0.993077

	uniform	distance and euclidean	distance and manhattan
max	0.994615	0.993077	0.993077

In [123]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

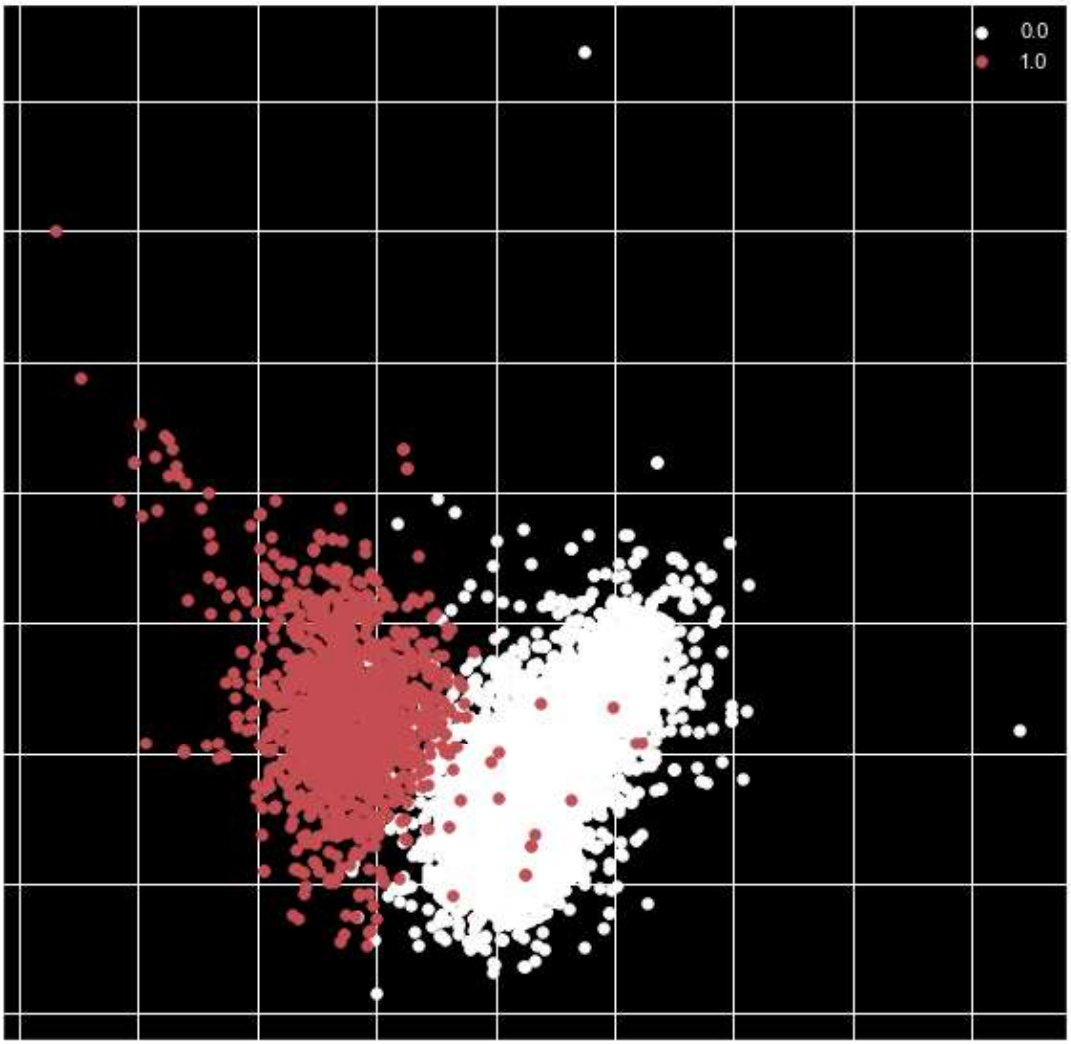
Out[123]:
```
Text(0, 0.5, 'Accuracy(%)')
```



Comparing the KNN classification accuracy on PCA and LDA features, maximum accuracy achieved using LDA features for color as a target variable is 99.46% which is greater than the accuracy obtained using PCA features of 99.23%. Thus can conclude that LDA works better compared to PCA in this case.

Visualization in 2D using PCA for Different wine colors : White and Red

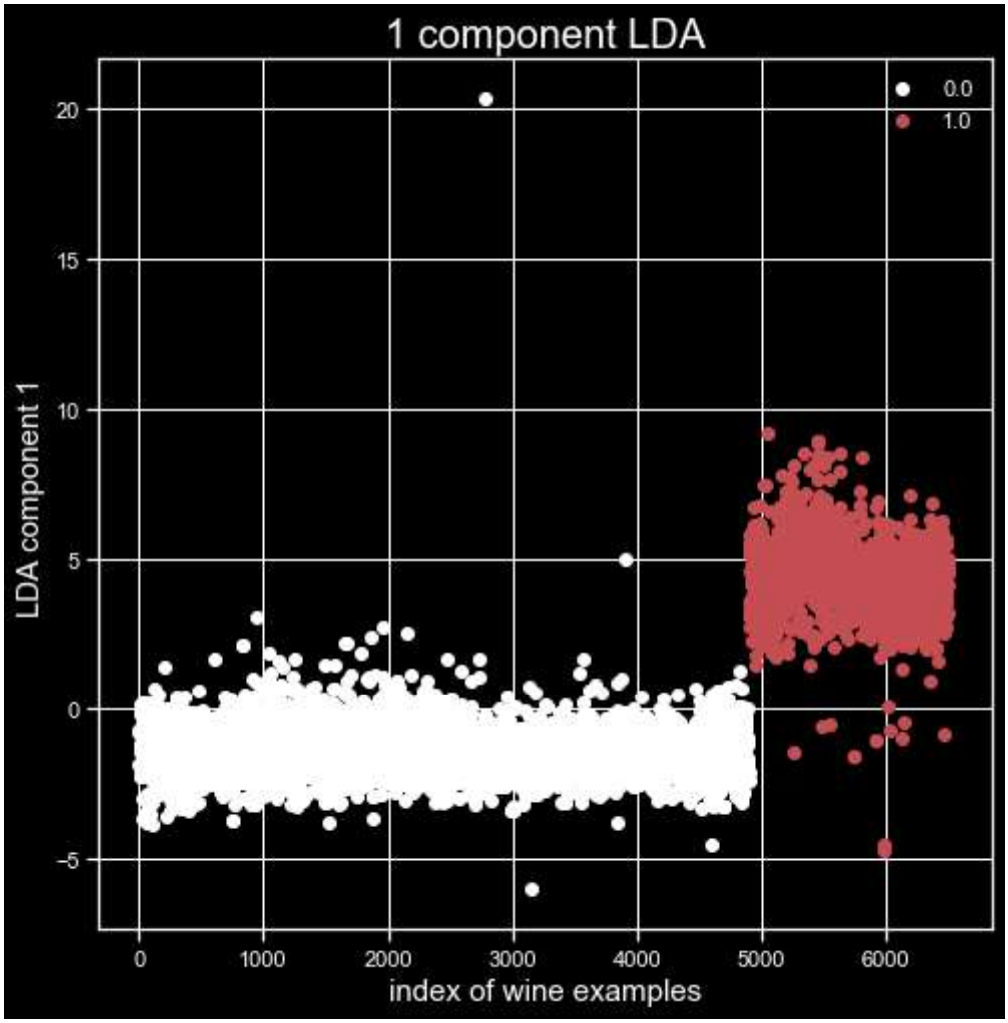
In [124]:
```
#for data visulaization I will be using full data
X_full_c = wine[D]
# further this data needs to standarized
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler(copy = False)
X_full_scaled_c = X_full_c.copy()
scalar.fit_transform(X_full_scaled_c)
# lets fit and transform data to two principal components on PCA
pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(X_full_scaled_c)
column_names = ["Principal component 1", "Principal component 2"]
PCA_2d = pd.DataFrame(data = principalComponents, columns = column_names)
yc = wine[C]
yc_new = yc.reset_index()
yc_new = yc_new.drop(['index'], axis =1)
final_PCA_2d_df = pd.DataFrame.join(self = PCA_2d, other = yc_new)
fig = plt.figure(figsize = (10,10))
plt.style.use('dark_background')
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
wine_colors = [0.0, 1.0]
colors = ['w','r']
for wine_color, color in zip(wine_colors,colors):
    indicesTokeep = final_PCA_2d_df['color'] == wine_color
    ax.scatter(final_PCA_2d_df.loc[indicesTokeep, 'Principal component 1']
               , final_PCA_2d_df.loc[indicesTokeep, 'Principal component 2']
               , c = color
               , s =30)
ax.legend(wine_colors)
ax.grid()
```



visuliazation in 1D using LDA

In [125]:
```
# I would be visualizing on the whole data
X_full_scaled_c
yc_new
plt.style.use('dark_background')
lda = LinearDiscriminantAnalysis(n_components = 1)
ldaComponents = lda.fit_transform(X_full_scaled_c, yc_new)
lda_1d_c = pd.DataFrame(data = ldaComponents, columns= ['LDA component'])
lda_1d_final_c = pd.DataFrame.join(lda_1d_c, yc_new)
lda_1d_final_c.rename_axis('index')
lda_1d_final_c =lda_1d_final_c.assign(new_index = lambda z: z.index)
lda_1d_final_c.shape

# lets visualize the thing
figure = plt.figure(figsize = (8,8))
ax = figure.add_subplot(1,1,1)
ax.set_ylabel('LDA component 1', fontsize = 15)
ax.set_xlabel('index of wine examples', fontsize = 15)
ax.set_title('1 component LDA', fontsize = 20)
wine_colors = [0.0, 1.0]
colors = ['w','r']
for wine_color, color in zip(wine_colors,colors):
    indicesTokeep = lda_1d_final_c['color'] == wine_color
    ax.scatter(x = lda_1d_final_c.loc[indicesTokeep,'new_index'],y = lda_1d_final
               , c = color)
ax.legend(wine_colors)
ax.grid()
```

```
C:\Users\aksha\Anaconda3\lib\site-packages\sklearn\utils\validation.py:724: Dat
aConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



1.1.3 Classifiacation : Quality

1.1.3.1 KNN Classification for wine Quality on normalized data with all features

In [126]:
```
## lets work on normalized data for Quality prediction for the wine
#we must not forget to transform the test data too
from sklearn.preprocessing import StandardScaler
s = StandardScaler()
X_train_scaled_q= s.fit_transform(X_train_q)
X_test_scaled_q = s.transform(X_test_q)

sns.set(style="ticks", color_codes=True)

n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_train_scaled_q, y_train_q)
    y_pred_q = neigh.predict(X_test_scaled_q)
    accscore = accuracy_score(y_test_q, y_pred_q)
    acc.at[k,col_names[0]] = accscore


for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_q, y_train_q)
    y_pred_q = neigh.predict(X_test_scaled_q)
    accscore = accuracy_score(y_test_q, y_pred_q)
    acc.at[k,col_names[1]] = accscore


for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_q, y_train_q)
    y_pred_q = neigh.predict(X_test_scaled_q)
    accscore = accuracy_score(y_test_q, y_pred_q)
    acc.at[k,col_names[2]] = accscore


acc.describe()
```

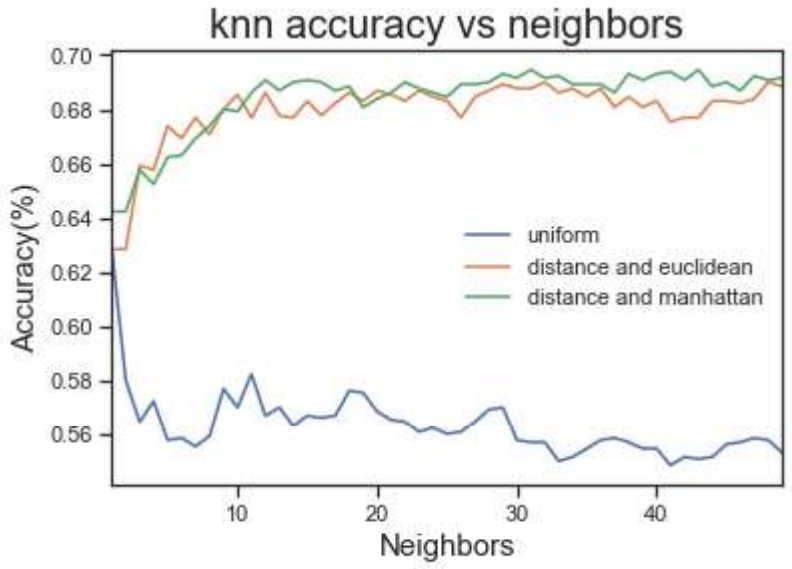
Out[126]:

	uniform	distance and euclidean	distance and manhattan
count	50.000000	50.000000	50.000000
mean	0.551954	0.665769	0.670262
std	0.080610	0.096883	0.097561
min	0.000000	0.000000	0.000000
25%	0.556346	0.676923	0.684038
50%	0.559615	0.683077	0.689231
75%	0.566923	0.686154	0.690769

	uniform	distance and euclidean	distance and manhattan
max	0.628462	0.690000	0.694615

In [127]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

Out[127]:
```
Text(0, 0.5, 'Accuracy(%)')
```



Trying diffrent metrics on normalized wine data for quality to compare performances (All features are used)

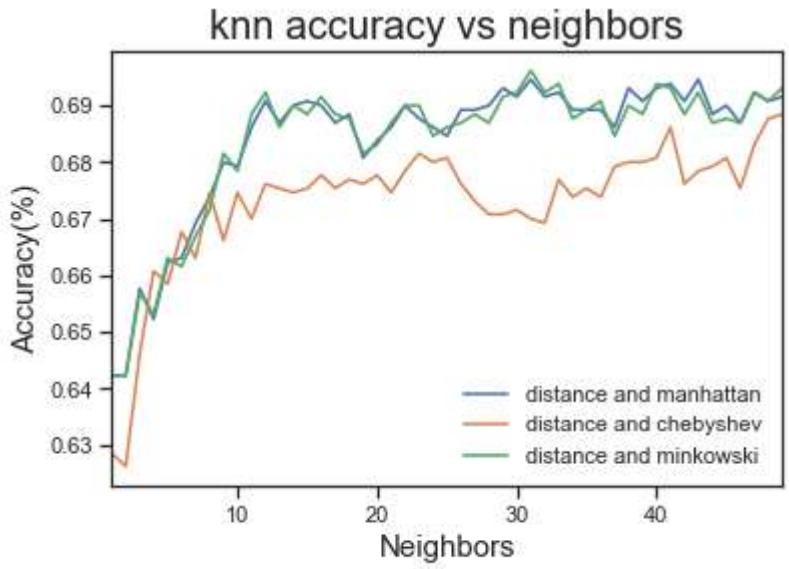
In [128]:
```
# Let us try multiple possilities for normalized datset
n_neighborslist = list(range(1,50))
col_names=['distance and manhattan','distance and chebyshev', 'distance and mink
#'distance and minkowski','distance and wminkowski', 'distance and seuclidean','
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_q, y_train_q)
    y_pred_q = neigh.predict(X_test_scaled_q)
    accscore = accuracy_score(y_test_q, y_pred_q)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_q, y_train_q)
    y_pred_q = neigh.predict(X_test_scaled_q)
    accscore = accuracy_score(y_test_q, y_pred_q)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_train_scaled_q, y_train_q)
    y_pred_q = neigh.predict(X_test_scaled_q)
    accscore = accuracy_score(y_test_q, y_pred_q)
    acc.at[k,col_names[2]] = accscore

acc.describe()
```

Out[128]:

	distance and manhattan	distance and chebyshev	distance and minkowski
count	50.000000	50.000000	50.000000
mean	0.670262	0.659477	0.669954
std	0.097561	0.095904	0.097524
min	0.000000	0.000000	0.000000
25%	0.684038	0.670769	0.683462
50%	0.689231	0.675385	0.688077
75%	0.690769	0.679038	0.690769
max	0.694615	0.688462	0.696154

In [129]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

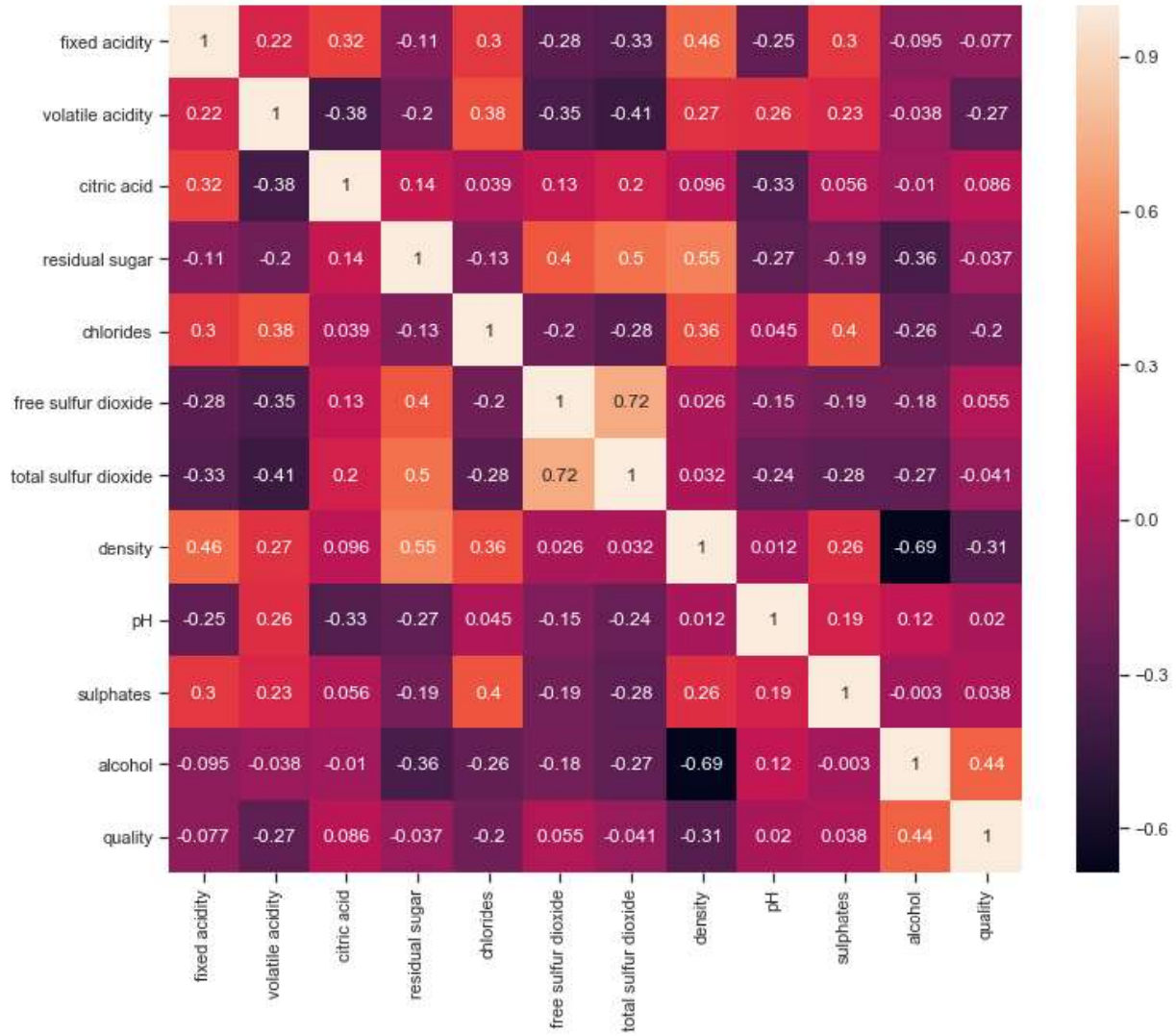
Out[129]:
```
Text(0, 0.5, 'Accuracy(%)')
```



1.1.4 Feature Selection : Quality

Using Pearson Correlation coefficient to select features

In [130]:
```
#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = wine[DL].corr()
sns.heatmap(cor, annot=True)
#cmap=plt.cm.Reds
plt.show()
```



1.1.4.1 Running KNN on normalized data for quality prediction with a subset of features ('alcohol','density', 'volatile acidity','chlorides')

In [131]:
```
# selecting most correlated features : alcohol (0.44),density(-0.31),volatile ac
selected_features_q  = ['alcohol','density', 'volatile acidity','chlorides']
X_sel_q = wine[selected_features_q]
X_sel_train_q, X_sel_test_q, y_sel_train_q, y_sel_test_q = train_test_split(X_se
X_sel_train_scaled_q = X_sel_train_q.copy()
scalar.fit_transform(X_sel_train_scaled_q)
## lets work on normalized data for quality prediction for the wine
#we must not forget to transform the test data too
X_sel_test_scaled_q = scalar.transform(X_sel_test_q)
n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_sel_train_scaled_q, y_sel_train_q)
    y_sel_pred_q = neigh.predict(X_sel_test_scaled_q)
    accscore = accuracy_score(y_sel_test_q, y_sel_pred_q)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_sel_train_scaled_q, y_sel_train_q)
    y_sel_pred_q = neigh.predict(X_sel_test_scaled_q)
    accscore = accuracy_score(y_sel_test_q, y_sel_pred_q)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights='distance', me
    neigh.fit(X_sel_train_scaled_q, y_sel_train_q)
    y_sel_pred_q = neigh.predict(X_sel_test_scaled_q)
    accscore = accuracy_score(y_sel_test_q, y_sel_pred_q)
    acc.at[k,col_names[2]] = accscore




acc.describe()
```

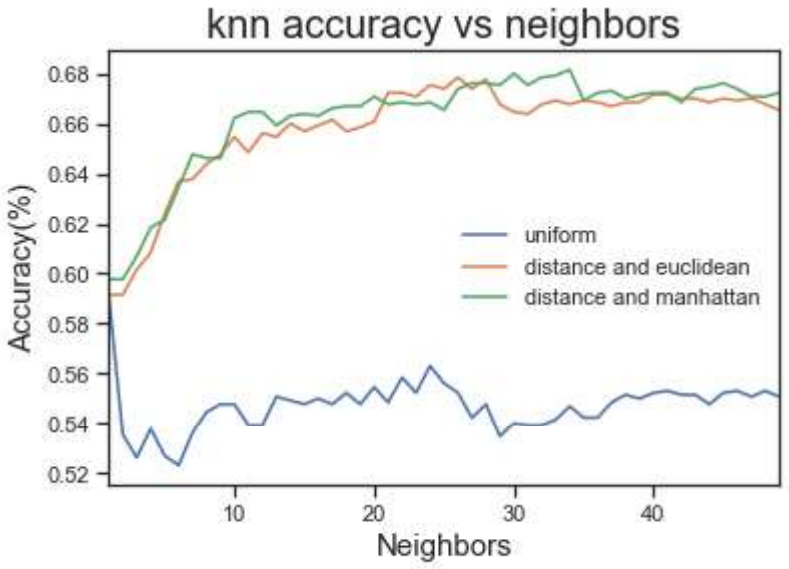
Out[131]:

	uniform	distance and euclidean	distance and manhattan
count	50.000000	50.000000	50.000000
mean	0.536169	0.644831	0.648877
std	0.078055	0.095380	0.095834
min	0.000000	0.000000	0.000000
25%	0.540385	0.655000	0.663077
50%	0.547692	0.667308	0.668462
75%	0.552115	0.670000	0.673654

	uniform	distance and euclidean	distance and manhattan
max	0.591538	0.678462	0.681538

In [132]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

Out[132]:
```
Text(0, 0.5, 'Accuracy(%)')
```



1.1.5 Feature Extraction: Quality

1.1.5.1. PCA : Quality

Running KNN on normalized and PCA tranformed data with 5 principal components for wine Quality prediction

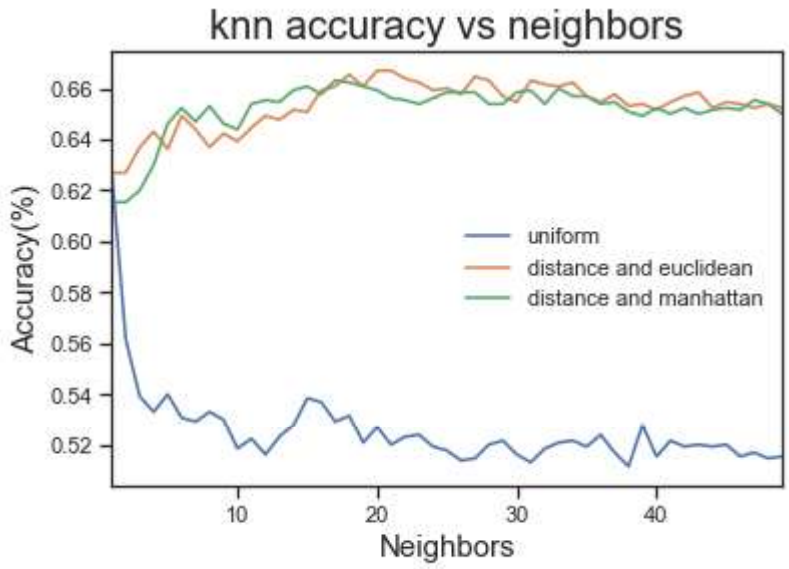
In [133]:
```
pca = PCA(n_components = 5,random_state =42)
principalComponents = pca.fit_transform(X_train_scaled_q)
col_name_pca = ["Principal Component " + str(i) for i in range(1,6)]
X_pca_train_q = pd.DataFrame(data = principalComponents, columns = col_name_pca)
y_pca_train_q = y_train_q
X_pca_test_q = pca.transform(X_test_scaled_q)
y_pca_test_q = y_test_q
n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_pca_train_q, y_pca_train_q)
    y_pca_pred_q = neigh.predict(X_pca_test_q)
    accscore = accuracy_score(y_pca_test_q, y_pca_pred_q)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_pca_train_q, y_pca_train_q)
    y_pca_pred_q = neigh.predict(X_pca_test_q)
    accscore = accuracy_score(y_pca_test_q, y_pca_pred_q)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_pca_train_q, y_pca_train_q)
    y_pca_pred_q = neigh.predict(X_pca_test_q)
    accscore = accuracy_score(y_pca_test_q, y_pca_pred_q)
    acc.at[k,col_names[2]] = accscore



acc.describe()
acc.head()
```

Out[133]:

	uniform	distance and euclidean	distance and manhattan
0	0.000000	0.000000	0.000000
1	0.626923	0.626923	0.615385
2	0.561538	0.626923	0.615385
3	0.539231	0.636923	0.620000
4	0.533077	0.643077	0.630000

In [134]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

Out[134]:
```
Text(0, 0.5, 'Accuracy(%)')
```



1.1.5.2 LDA : Quality

1.1.5.2.1 Running KNN on normalized and LDA tranformed data with 5 principal components for wine Quality prediction

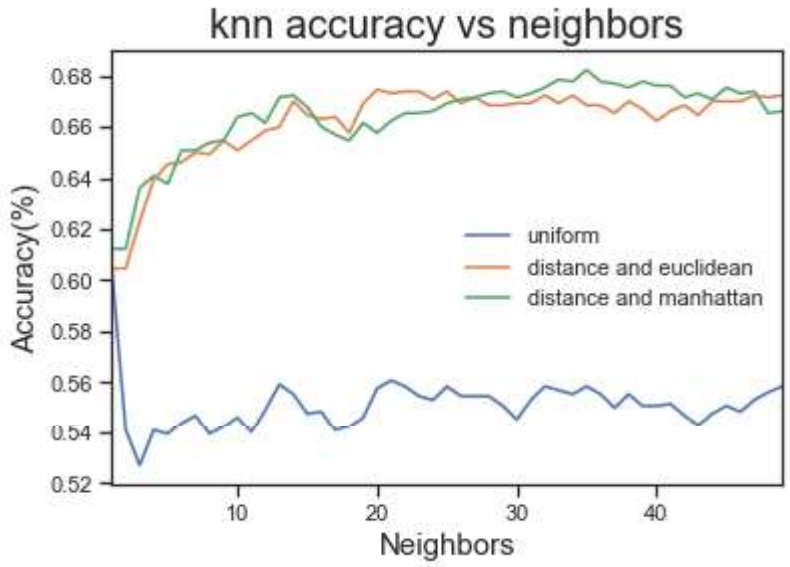
In [135]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
col_name_lda = ["LDA component 1","LDA component 2","LDA component 3","LDA compo
lda = LinearDiscriminantAnalysis(n_components = 5)
ldaComponents = lda.fit_transform(X_train_scaled_q, y_train_q)
X_lda_train_q = pd.DataFrame(data = ldaComponents, columns = col_name_lda)
y_lda_train_q = y_train_q
y_lda_test_q = y_test_q
X_lda_test_q = lda.transform(X_test_scaled_q)
n_neighborslist = list(range(1,50))
col_names=['uniform','distance and euclidean','distance and manhattan']
accarray = np.zeros((len(n_neighborslist),3))


#add multiple plots to same chart, one for each weighting approach
acc=pd.DataFrame(accarray, columns=col_names)

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=col_names[0])
    neigh.fit(X_lda_train_q, y_lda_train_q)
    y_lda_pred_q = neigh.predict(X_lda_test_q)
    accscore = accuracy_score(y_lda_test_q, y_lda_pred_q)
    acc.at[k,col_names[0]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_lda_train_q, y_lda_train_q)
    y_lda_pred_q = neigh.predict(X_lda_test_q)
    accscore = accuracy_score(y_lda_test_q, y_lda_pred_q)
    acc.at[k,col_names[1]] = accscore

for k in n_neighborslist:
    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights= 'distance', m
    neigh.fit(X_lda_train_q, y_lda_train_q)
    y_lda_pred_q = neigh.predict(X_lda_test_q)
    accscore = accuracy_score(y_lda_test_q, y_lda_pred_q)
    acc.at[k,col_names[2]] = accscore


acc.describe()
```

Out[135]:

	uniform	distance and euclidean	distance and manhattan
count	50.000000	50.000000	50.000000
mean	0.540077	0.648523	0.650954
std	0.078624	0.094883	0.095128
min	0.000000	0.000000	0.000000
25%	0.545577	0.657885	0.658269
50%	0.550769	0.668462	0.668462
75%	0.555385	0.670000	0.673654
max	0.604615	0.674615	0.682308

In [136]:
```
graph = acc[1:].plot.line()
graph.set_title('knn accuracy vs neighbors', fontsize = 20)
graph.set_xlabel("Neighbors", fontsize = 15)
graph.set_ylabel("Accuracy(%)", fontsize = 15)
```

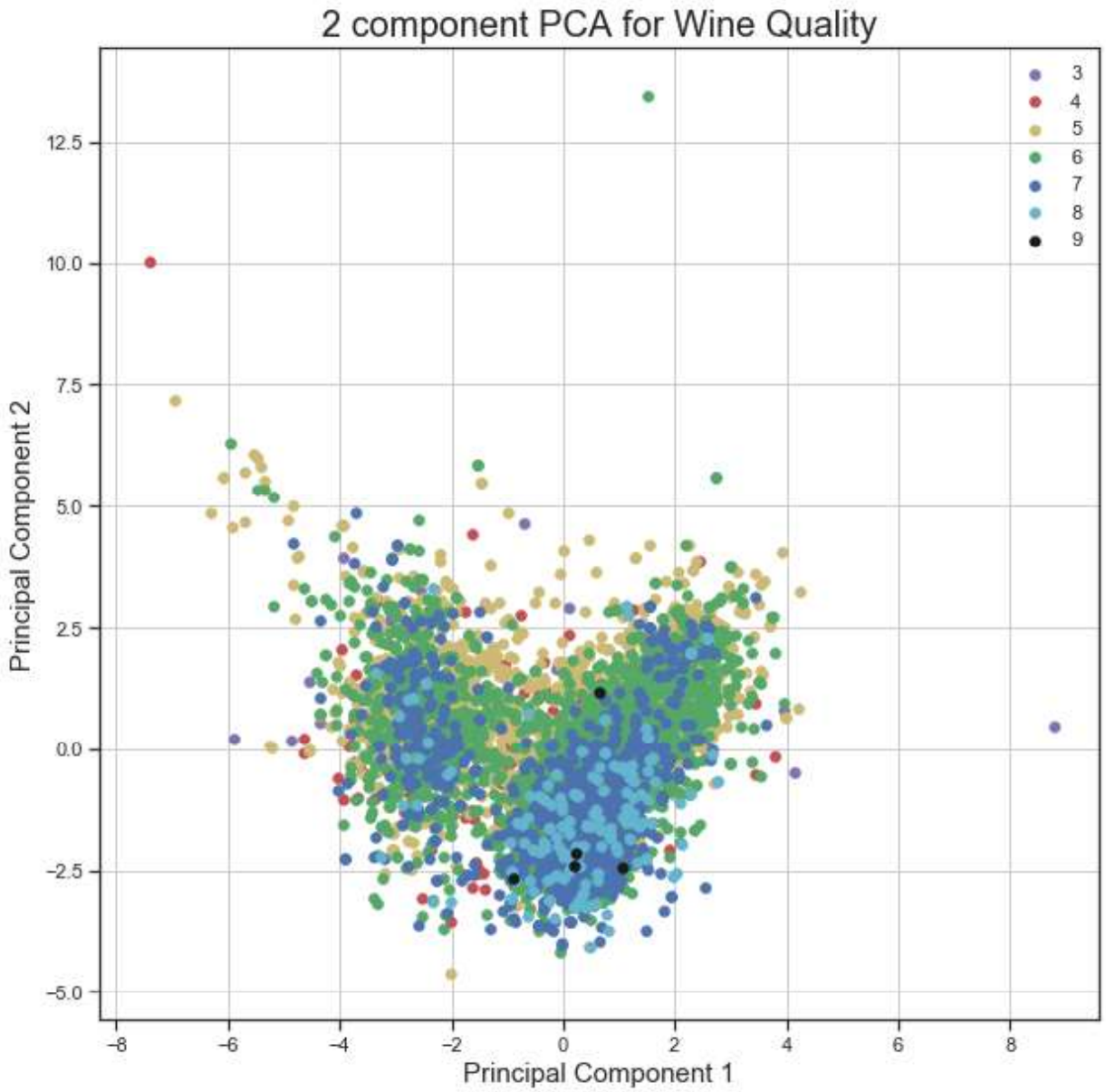
Out[136]:
```
Text(0, 0.5, 'Accuracy(%)')
```



Comparing the KNN classification accuracy on PCA and LDA features, maximum accuracy achieved using LDA features for quality target variable is 68.23% which is greater than the accuracy obtained using PCA features of 66.69% respectively. Thus can conclude that LDA works better compared to PCA in this case.

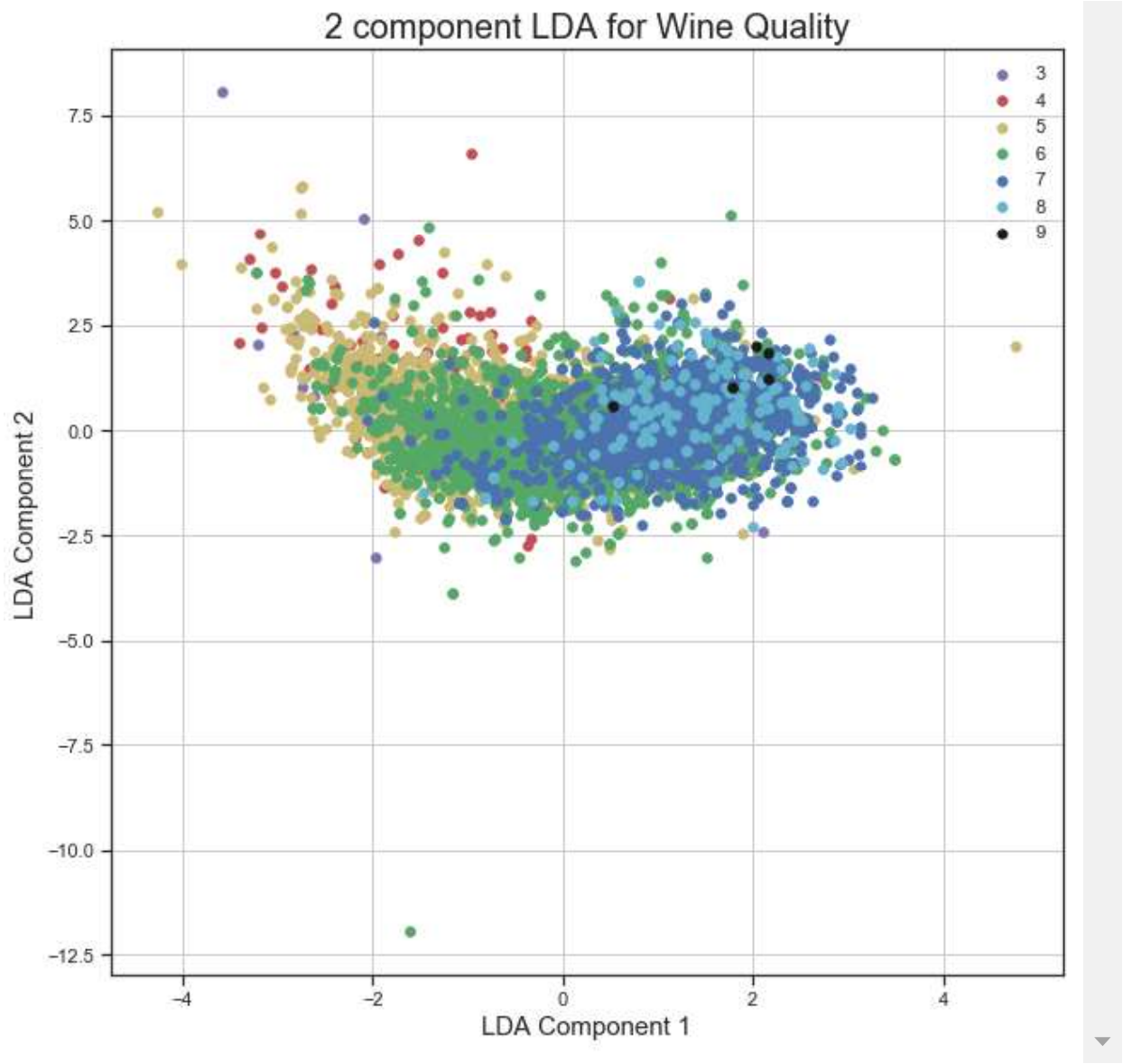
Visualization in 2D using PCA for different Wine Quality : 3,4,5,6,7,8,9

In [137]:
```
# further this data needs to standarized
#for data visulaization I will be using full data
X_full_q = wine[D]
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler(copy = False)
X_full_scaled_q = X_full_q.copy()
scalar.fit_transform(X_full_scaled_q)
# lets fit and transform data to two principal components on PCA
pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(X_full_scaled_q)
column_names = ["Principal component 1", "Principal component 2"]
PCA_2d_q = pd.DataFrame(data = principalComponents, columns = column_names)
yq = wine[L]
yq_new = yq.reset_index()
yq_new = yq_new.drop(['index'], axis =1)
final_PCA_2d_df = pd.DataFrame.join(self = PCA_2d_q, other = yq_new)

fig = plt.figure(figsize = (10,10))
plt.style.use('seaborn-ticks')
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA for Wine Quality', fontsize = 20)
wine_qualities = [3, 4,5,6,7,8,9]
colors = ['m','r','y','g','b','c','k']
for wine_quality, color in zip(wine_qualities,colors):
    indicesTokeep = final_PCA_2d_df['quality'] == wine_quality
    ax.scatter(final_PCA_2d_df.loc[indicesTokeep, 'Principal component 1']
               , final_PCA_2d_df.loc[indicesTokeep, 'Principal component 2']
               , c = color
               , s =30)
ax.legend(wine_qualities)
ax.grid()
```



visuliazation in 2D using LDA for Wine Quality

In [138]:
```
# I would be visualizing on the whole data
lda = LinearDiscriminantAnalysis(n_components = 2)
ldaComponents = lda.fit_transform(X_full_scaled_q, np.ravel(yq_new.to_numpy()))
lda_2d_q = pd.DataFrame(data = ldaComponents, columns= ['LDA component 1', 'LDA
lda_2d_final_q = pd.DataFrame.join(lda_2d_q, yq_new)
lda_2d_final_q.rename_axis('index')
lda_2d_final_q =lda_2d_final_q.assign(new_index = lambda z: z.index)
lda_2d_final_q.shape
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('LDA Component 1', fontsize = 15)
ax.set_ylabel('LDA Component 2', fontsize = 15)
ax.set_title('2 component LDA for Wine Quality', fontsize = 20)
wine_qualities = [3,4,5,6,7,8,9]
colors = ['m','r','y','g','b','c','k']
for wine_quality, color in zip(wine_qualities,colors):
    indicesTokeep = lda_2d_final_q['quality'] == wine_quality
    ax.scatter(lda_2d_final_q.loc[indicesTokeep, 'LDA component 1']
               , lda_2d_final_q.loc[indicesTokeep, 'LDA component 2']
               , c = color
               , s =30)
ax.legend(wine_qualities)
ax.grid()
```



1.1.6 Analysis and Discussion

1.1.6.1 K plots :

These plots are plotted above, please find these plots in the relevant parts of the notebook

1.1.6.2 Features :

From the correlation heatmap plotted for feature selection methods and by looking at the scatter plots, we can say that following features have higher effect on each other :

1. Free sulpher dioxide and total sulfur di oxide share have a correlation of .72 and share 51.84% of variance.
2. Density and alcohol have a correlation of .69 and share a variance of 47.61% of variance.
3. Residual sugar and density share a high correlation of .55 and hence share 30.25 % of variance.

4. Fixed acidity and density share a correlation of .46 and hence share a medium variance of 21.16 % of variance.

1.1.6.3 Selected Features:

1. The best accuracy achieved on wine color using KNN is 99.61% , where as best accuracy achieved on color using selected features as 'total sulpher dioxide', 'volatile acidity', 'chlorides' and 'sulphates' is 99.07%.
2. The best accuracy achieved on wine quality using KNN is 69.46% , whereas best accuracy achieved on quality using selected features as 'alcohol', 'density','volatile acidity' and 'chlorides' is 68.15%.
3. It can be seen that better accuracy is achievd using the full features instead of a selection of features.
4. The best performance of PCA(with 5 components) on wine color is achieved as 99.23%, whereas best performance of PCA(with 5 features) on wine quality as 66.69 %
5. Best performance of LDA (1 component) for wine color is achieved as 99.46% and best performance of LDA (5 component) for wine quality is achieved as 68.23%

To conclude, we can say that PCA(5 components) and LDA(1 component) performed better for wine color prediction compared to four features selected using the correlation among features and wine colors. Whereas, In the case of wine quality prediction, four selected features based model (using correlation among features and wine quality), performed better than PCA(5 components) based model, but could not perform better than LDA(5 components).

1.1.6.4 PCA vs. LDA:

1. From the above k-plots of KNN classification accuracy for different features from the dataset, PCA and LDA, it can be depicted that using PCA or LDA for n_components =5 does not result into better accuracy than the accuracy obtained using given dataset features.

2. Comparing the KNN classification accuracy on PCA and LDA features, maximum accuracy achieved using LDA features for color and quality target variables are 99.46% and 68.23% which is greater than the accuracy obtained using PCA features of 99.23% and 66.69% respectively. Thus can conclude that LDA works better compared to PCA in this case.

3. Normlization does affect the performance of PCA,In PCA we are interested in the components that maximize the variance. If one component (e.g. density) varies less than another (e.g. total sulfur dioxide) because of their respective scales (g/cm3 vs. ppm), PCA might determine that the direction of maximal variance more closely corresponds with the ‘density’ axis, if those features are not scaled, which would be incorrect. (Reference: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html (https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html))

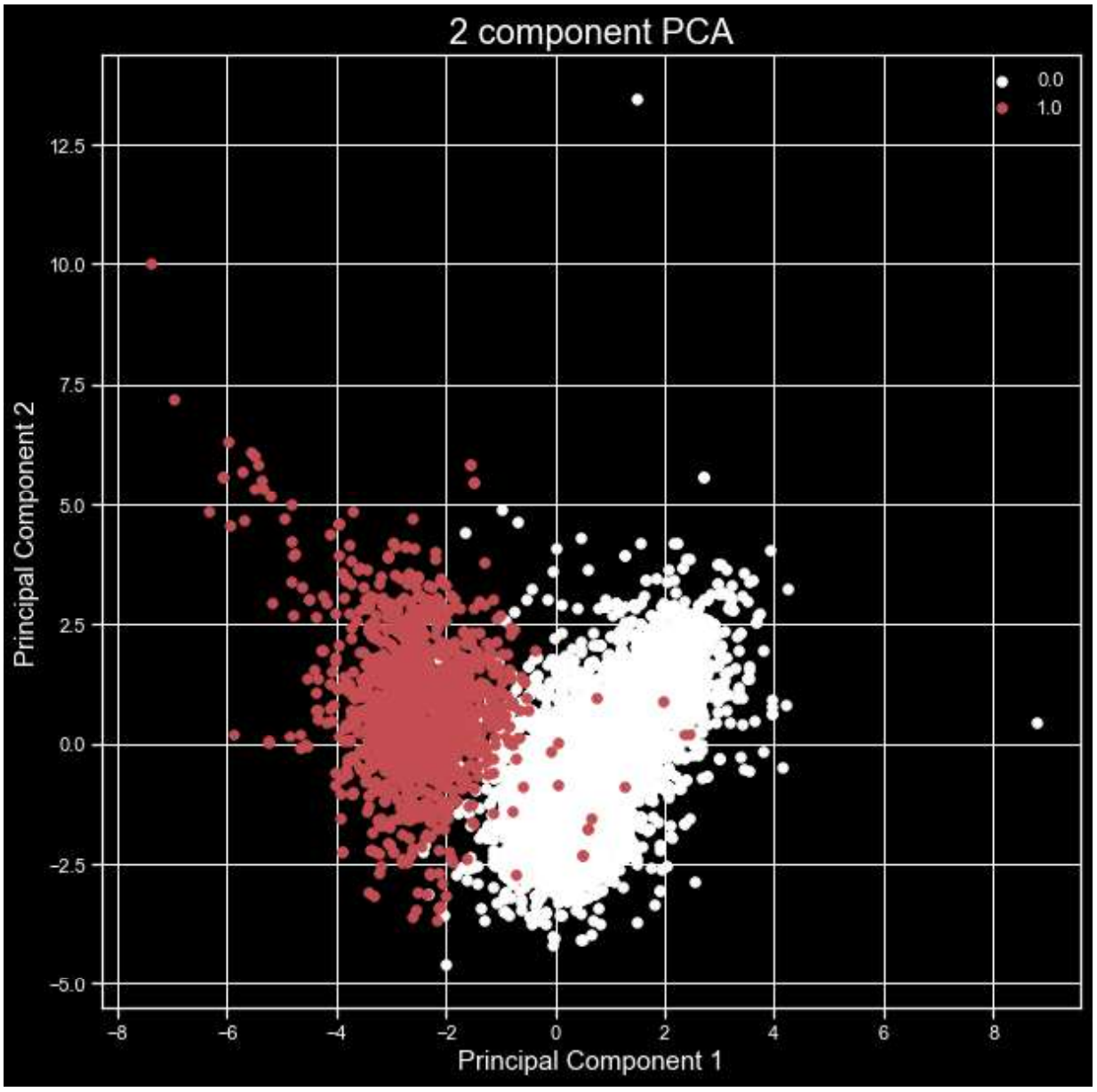
However, Normalization does not affect the LDA. LDA is the classification technique using target variable to separate variables in the lower dimension. It tries to minimise the variance within group and maximise the separation between the group which is not affected by the normalization.

(Reference: https://stats.stackexchange.com/questions/109071/standardizing-features-when-using-lda-as-a-pre-processing-step (https://stats.stackexchange.com/questions/109071/standardizing-features-when-using-lda-as-a-pre-processing-step))

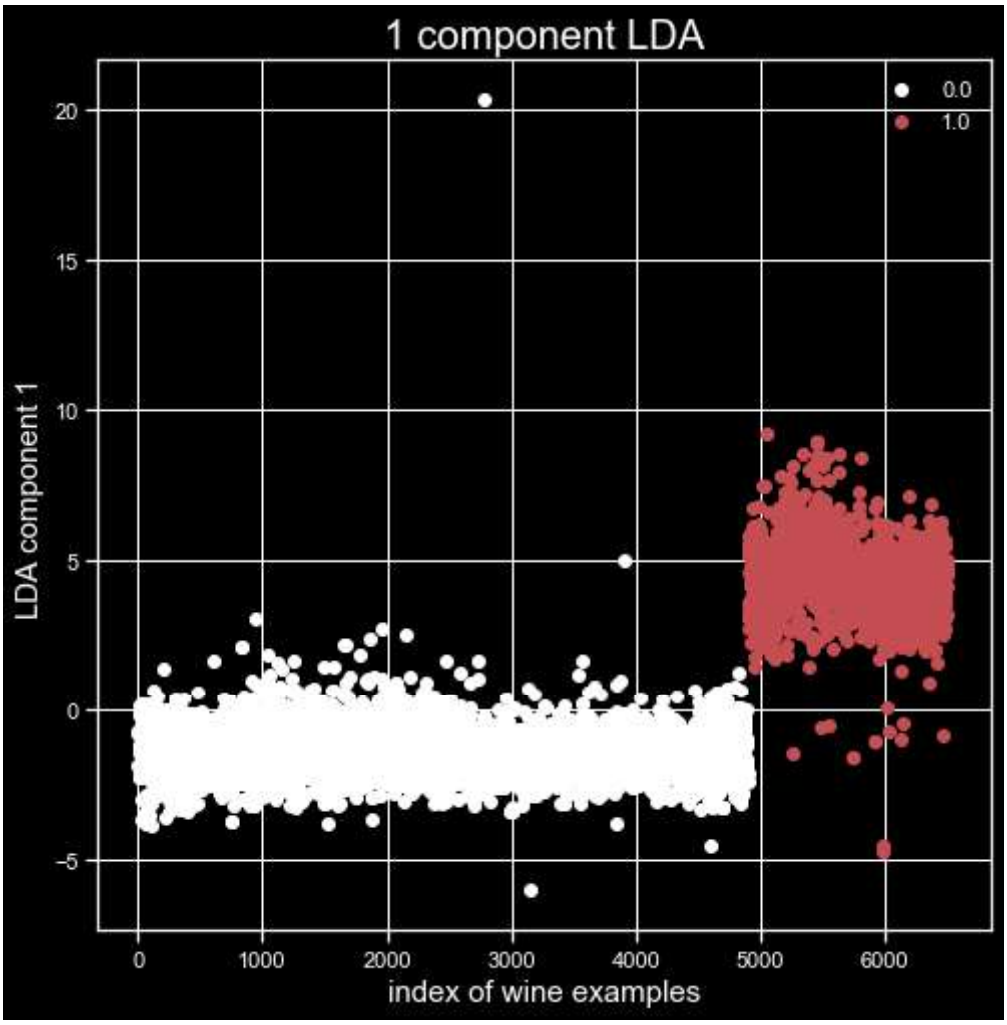
1.1.6.5 Plot

These plots are plotted above, please refer to relevant parts of code.

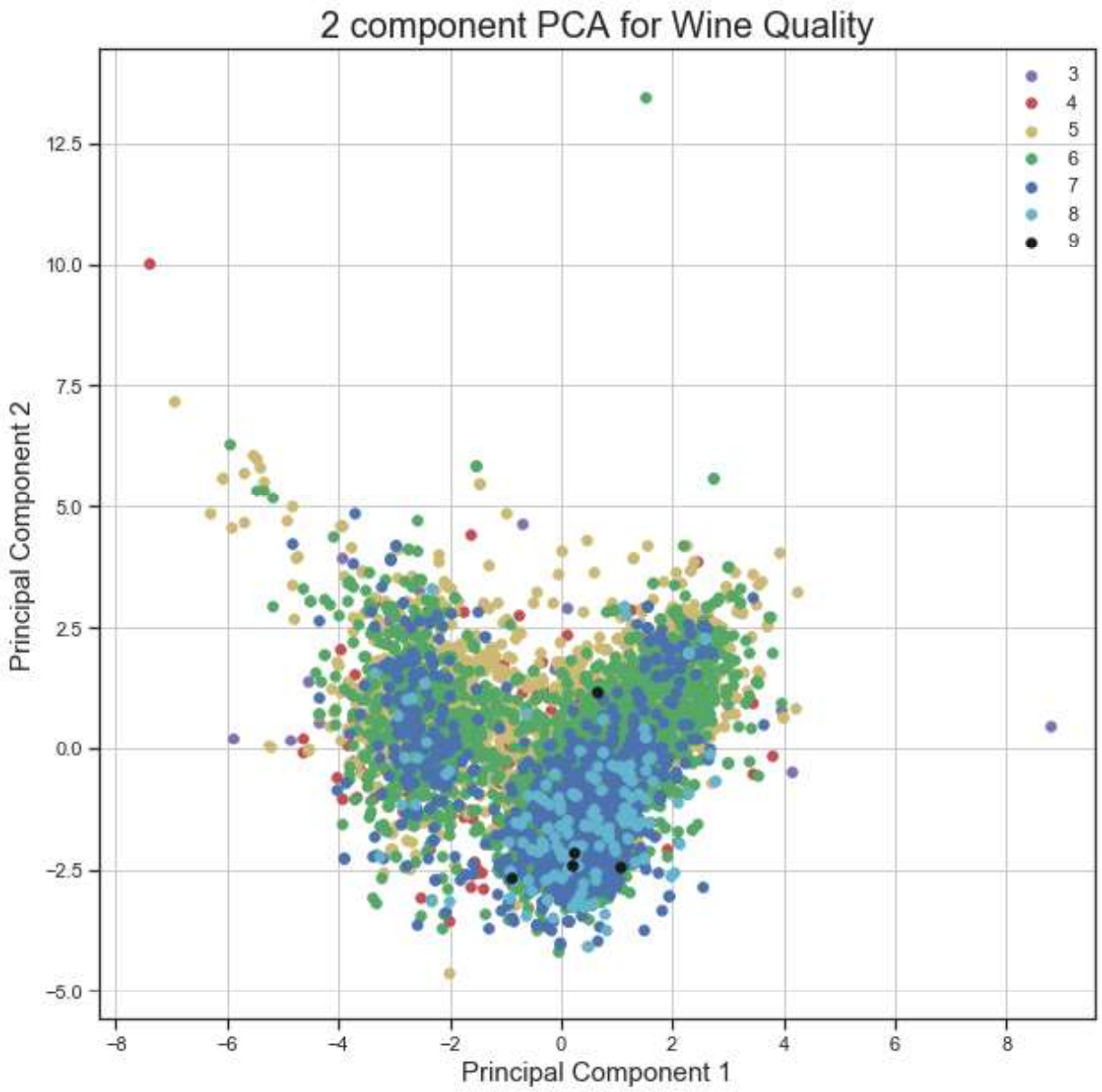
here 1 represents "red" and 0 represents "white" color for wine
Wine color projection using PCA in 2D



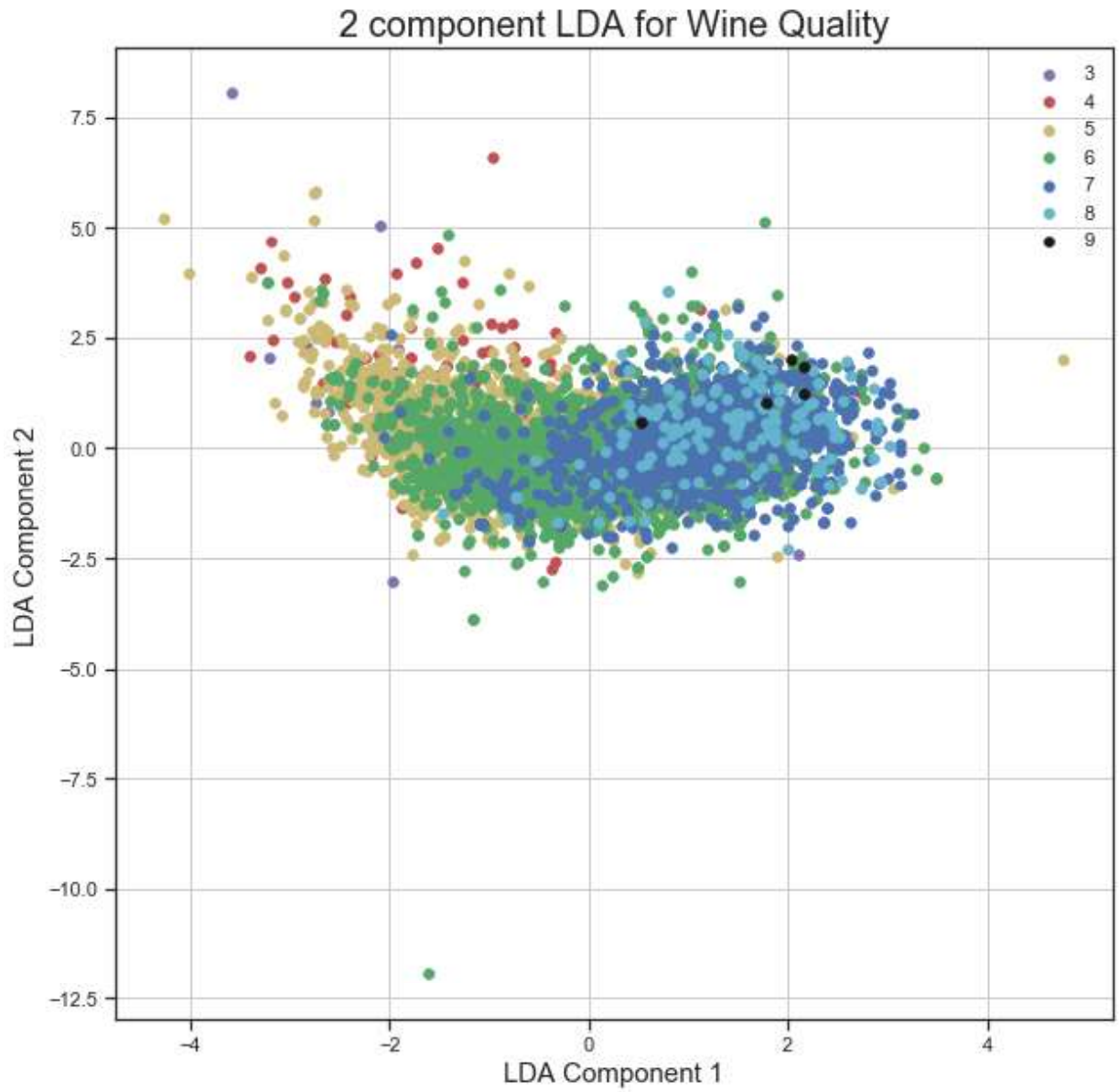
here 1 represents "red" and 0 represents "white" color for wine
Wine color projection using LDA in 1D



Wine quality projection using PCA in 2D



Wine quality rojection using LDA in 2D



By looking at the above plots and comparing these plots with pairplots plotted above, we have the following observation :

1. PCA and LDA plots are more informative compared to pairplots, and it is easier to comprehend data on the projected axis of PCA and LDAs.
2. In PCA, data is projected into principal axis which are orthogonal to each other (thus having a correlation = 0), whereas in pair plots the feature axis might share some correlation. This independence of principal components (among each other) helps in better visualization of data, as can be seen for wine color data projection on first two principal components.
3. The basis for LDA is to maximize separation among different classes while minimizing variance within all classes, thus it is expected from LDA to perform better for separating target variables. This can be seen from the plots above, The projection of wine quality data on LDA components gives good information about data separability compared to pair plots.

To conclude, it can be said, the projection of data on PCA and LDA components is better representative of separation of classes, as these components are formed using all the data features to maintain maximum data variance and to maximize separation between classes while

minimizing variance shared among all classes respectively. Thus a few components obtained using PCA and LDA(few plots using few components) might summarize the dataset well compared to same number of features used directly from dataset.

Q 2

2.1 Dataset

In [139]:

```
import numpy as np
import pandas as pd
import random
import seaborn as sns
sns.set(style="ticks", color_codes=True)
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
DataB = pd.read_csv("DataB.csv", sep=',')
DataB_features = DataB.drop(['gnd'], axis =1)
DataB_features.columns
DataB_features.rename(columns={'Unnamed: 0': "indexed"}, inplace = True)
DataB_features.indexed = DataB_features.indexed -1
DataB_features.set_index('indexed', inplace = True)
DataB_features.shape
X = DataB_features
DataB_class = DataB.gnd
DataB_class = DataB_class.to_frame()
DataB_class.rename_axis('indexed', inplace = True)
DataB_class.rename(columns={'gnd': "class"}, inplace = True)
```

2.2 Principal Component Analysis

2.2.1 Practical Questions

1) In PCA, compute the eigenvectors and eigenvalues. Plot the scree plot and visually discuss which cut-off is good.