# Question 1:

In [27]:

```
# 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
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.patheffects as PathEffects
```

In [28]:

```
#Columns/Features
D = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfu
r dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']
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()
#inserting wine colou feature
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])
```

In [29]:

```
# #checking data heads
# wine[D].describe()
# wine[D].head()
# wine[DLC].describe()
# wine[DLC].head()
```

## BEFORE NORMALIZATION

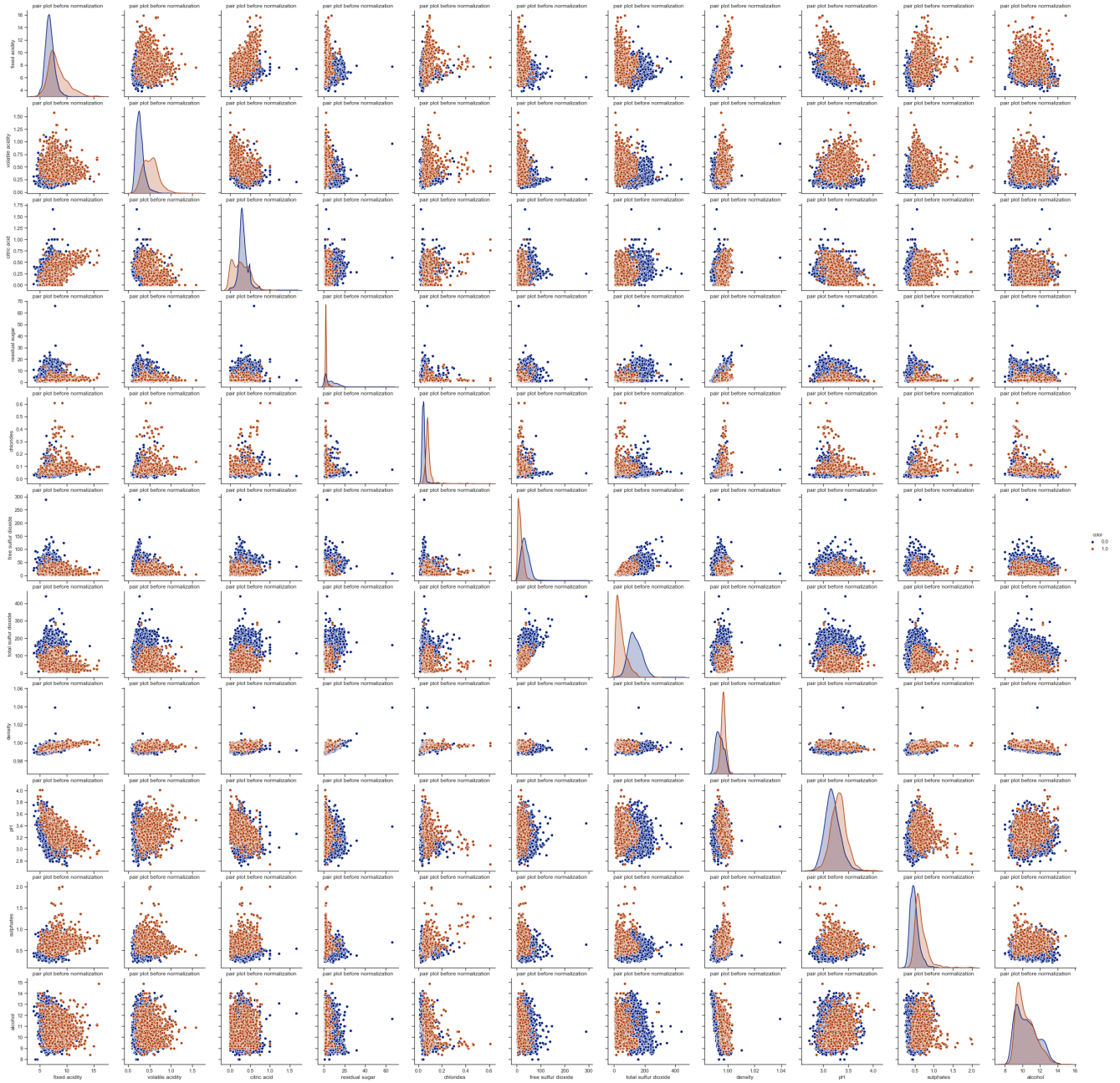
In [30]:

```
sns.set_palette("dark")

#print(wine_dc_one[DC].head())
sns_plot=sns.pairplot(wine[DC],hue='color',vars=D,height=3)
#sns_plot.savefig("output.png")

#plt.title("pair plots before normalization")
sns_plot.set(title ="pair plot before normalization")
```

Out[30]:

<seaborn.axisgrid.PairGrid at 0x1b186649248>

<seaborn.axisgrid.PairGrid at 0x1b166492e8>




***selecting 4 best features to plot pair plots as 11 x 11 cannot be visualised properly***

Using Univariate Selection Method

In [31]:

```
data = wine[DC]
#print(type(data))
X = data.iloc[:,0:10]  #independent columns
y = data.iloc[:,-1]    #target column i.e color of wine
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=6)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
# #concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(11,'Score'))  #print 10 best features
```

```
                 Specs          Score
6  total sulfur dioxide  87946.248732
5   free sulfur dioxide  14913.554060
3        residual sugar   3287.056863
```

```
5        residual sugar   3267.056865
0          fixed acidity    358.497091
1       volatile acidity    221.064880
9              sulphates     64.271565
4              chlorides     37.398554
2            citric acid     15.118752
8                     pH      5.652320
7                density      0.008961
```

From the above results selecting features with top 4 scores;

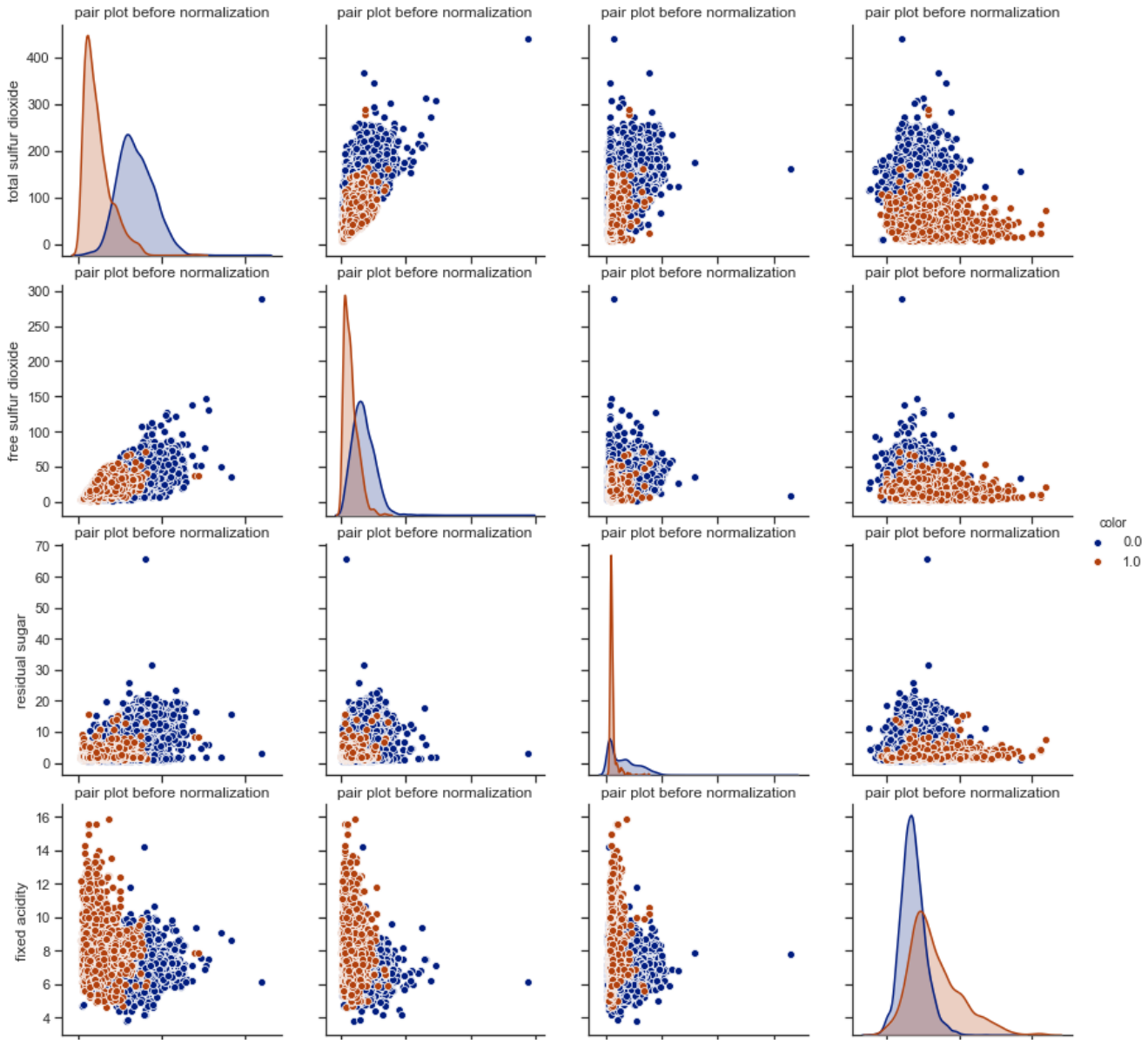
So features considered for pair plots are : ['total sulfur dioxide','free sulfur dioxide','residual sugar','fixed acidity']

In [32]:

```
#pair plots before normalization

sns.set_palette("dark")

#print(wine_dc_one[DC].head())
sns_plot=sns.pairplot(wine[DC],hue='color',vars=['total sulfur dioxide','free sulfur dioxide','res
idual sugar','fixed acidity'],height=3)
#sns_plot.savefig("output.png")

#plt.title("pair plots before normalization")
sns_plot.set(title ="pair plot before normalization")
```

Out[32]:

```
<seaborn.axisgrid.PairGrid at 0x1b191aef148>
```




***function for caluclating accuracies for different schemes and labels -> to minismize code repetition***

In [33]:

```
#function for caluclating accuracy for different schemes
def cal_accuracy(X_train,y_train,X_test,y_test):
    n_neighborslist = list(range(1,50))
    col_names_new=['uniform','distance p = 1','distance p = 2']
    accarray = np.zeros((len(n_neighborslist),3))
    col_names = ['uniform','distance']
    #add multiple plots to same chart, one for each weighting approach
    acc =pd.DataFrame(accarray, columns=col_names_new)
    acc_subfeatures =pd.DataFrame(accarray, columns=col_names_new)
    for k in n_neighborslist:
        for c in col_names:
            if c == 'distance':
                for dp in [1,2]:
                    neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=c,p=dp)
                    neigh.fit(X_train, y_train)
                    y_pred = neigh.predict(X_test)
                    accscore = accuracy_score(y_test, y_pred)
                    acc.at[k,c+' p = '+str(dp)] = accscore
            else:
                neigh = neighbors.KNeighborsClassifier(n_neighbors=k, weights=c)
                neigh.fit(X_train, y_train)
                y_pred = neigh.predict(X_test)
                accscore = accuracy_score(y_test, y_pred)
                acc.at[k,c] = accscore
    return acc
```

***part of code for calucalting and plotting graps for all features and sub- set of selected features for different schemes***

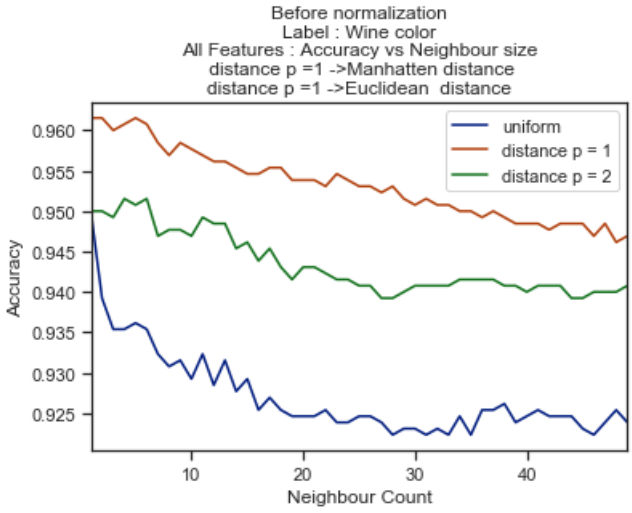
In [34]:

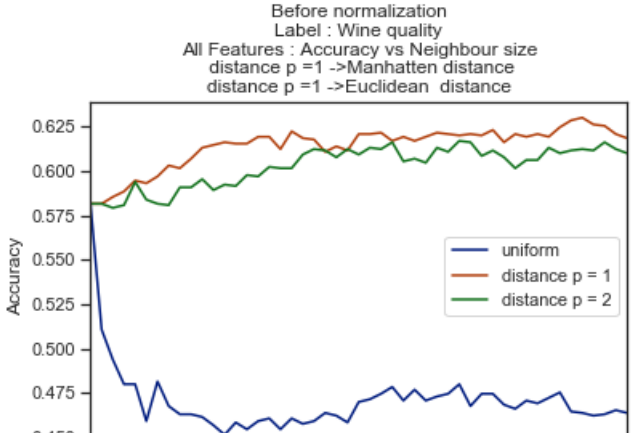
```
################################### ALL FEATURES
#####################################################################
################################### LABEL : WINE COLOR
#############################################################
X = wine[D].values
y = np.ravel(wine[[C]])

# deviding data set in train and test sets
ran = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)
# all features label : wine color
acc = cal_accuracy(X_train,y_train,X_test,y_test)
#plotting line graph
acc[1:].plot.line()
plt.title("Before normalization\nLabel : Wine color\nAll Features : Accuracy vs Neighbour size\n d
istance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
################################### LABEL : WINE QUALITY
#############################################################
X_quality = wine[D].values
y_quality = np.ravel(wine[[L]])

# deviding data set in train and test sets
ran = 42
X_train_quality, X_test_quality, y_train_quality, y_test_quality = train_test_split(X_quality, y_qu
ality, test_size=0.2, random_state = ran)
# all features label : wine color
acc_quality = cal_accuracy(X_train_quality,y_train_quality,X_test_quality,y_test_quality)
#plotting line graph
acc_quality[1:].plot.line()
plt.title("Before normalization\nLabel : Wine quality\nAll Features : Accuracy vs Neighbour size\n
distance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
################################### SUB SET OF FEATURES
```

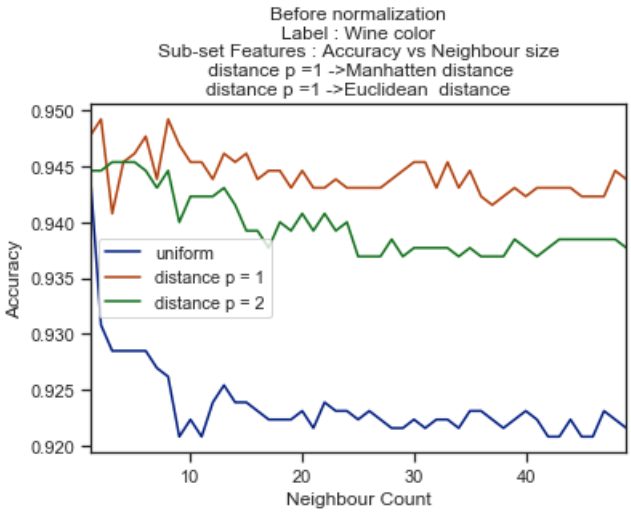
```
#################################### SUB SET OF FEATURES
#################################################################
################################## LABEL : WINE COLOR
#################################################################
X_sub = wine[['total sulfur dioxide','free sulfur dioxide','residual sugar','fixed acidity']].valu
es
y_sub = np.ravel(wine[[C]])
# deviding data set in train and test sets for the selected sub set of features
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(X_sub, y_sub, test_size=0.2, ra
ndom_state = ran)

# subset features label : wine color
acc_subfeatures = cal_accuracy(X_train_sub,y_train_sub,X_test_sub,y_test_sub)
acc_subfeatures[1:].plot.line()
plt.title("Before normalization\nLabel : Wine color\nSub-set Features : Accuracy vs Neighbour size
\n distance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
################################## LABEL : WINE COLOR
#################################################################
X_sub_quality = wine[['total sulfur dioxide','free sulfur dioxide','residual sugar','fixed acidity
']].values
y_sub_quality = np.ravel(wine[[L]])
# deviding data set in train and test sets for the selected sub set of features
X_train_sub_quality, X_test_sub_quality, y_train_sub_quality, y_test_sub_quality = train_test_split
(X_sub_quality, y_sub_quality, test_size=0.2, random_state = ran)

# subset features label : wine color
acc_subfeatures_quality = cal_accuracy(X_train_sub_quality,y_train_sub_quality,X_test_sub_quality,
y_test_sub_quality)
acc_subfeatures_quality[1:].plot.line()
plt.title("Before normalization\nLabel : Wine quality\nSub-set Features : Accuracy vs Neighbour si
ze\n distance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

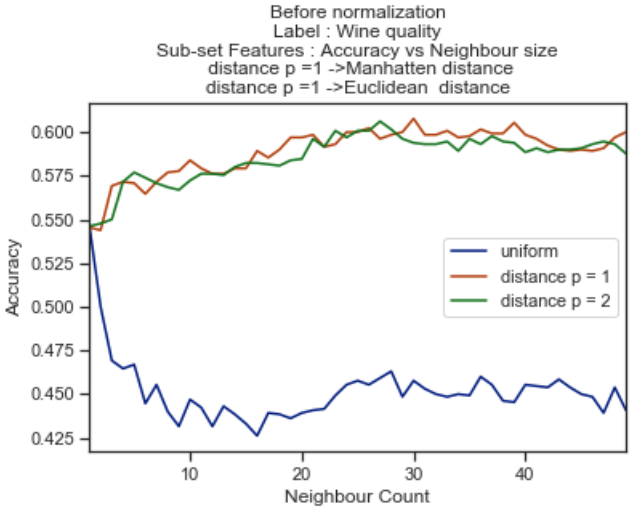
Out[34]:

```
Text(0, 0.5, 'Accuracy')
```



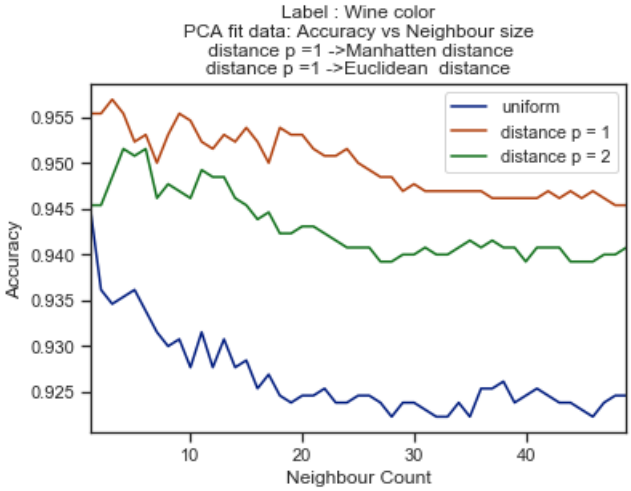

***Using PCA to fit data and use KNN for top 5 principle components and ploting graph for accuracy vs neighbor size***

In [36]:

```
X = wine[D].values
y = np.ravel(wine[[C]])
#print(type(X_train))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

#fitting PCA model only for train set
pca = PCA(n_components=5)
pca.fit(X_train)

#transforming both train and test data using fitted model
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
#print(type(X_train_pca))

#calling function to caluclate accuracies
acc_pca = cal_accuracy(X_train_pca,y_train,X_test_pca,y_test)
acc_pca[1:].plot.line()
plt.title("Label : Wine color\nPCA fit data: Accuracy vs Neighbour size\n distance p =1 ->Manhatte
n distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
acc_pca.describe()
acc_pca.head()
```

Out[36]:

| | uniform | distance p = 1 | distance p = 2 |
|---|---|---|---|
| 0 | 0.000000 | 0.000000 | 0.000000 |
| 1 | 0.945385 | 0.955385 | 0.945385 |
| 2 | 0.936154 | 0.955385 | 0.945385 |
| 3 | 0.934615 | 0.956923 | 0.948462 |
| 4 | 0.935385 | 0.955385 | 0.951538 |



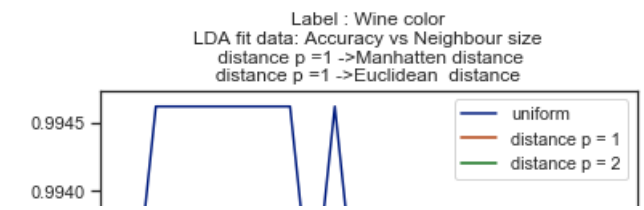

***using Linear Discriminant Analysis to fit the data and then ploting graph for accuracy vs neighbor size***

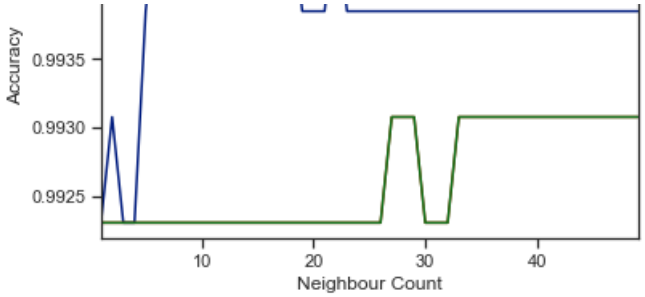
In [37]:

```
X = wine[D].values
y = np.ravel(wine[[C]])
#print(type(X_train))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

#fitting LinearDiscriminantAnalysis model only for train set
lda = LinearDiscriminantAnalysis(n_components=None)
lda.fit(X_train,y_train)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train)
X_test_lda = lda.transform(X_test)
#print(type(X_train_pca))
# print(X_train_lda.shape)

#calling function to caluclate accuracies
acc_lda = cal_accuracy(X_train_lda,y_train,X_test_lda,y_test)
# print(type(acc_lda))
acc_lda[1:].plot.line()
plt.title("Label : Wine color\nLDA fit data: Accuracy vs Neighbour size\n distance p =1 ->Manhatte
n distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
# acc_lda.describe()
# acc_lda.head()
```

Out[37]:

```
Text(0, 0.5, 'Accuracy')
```



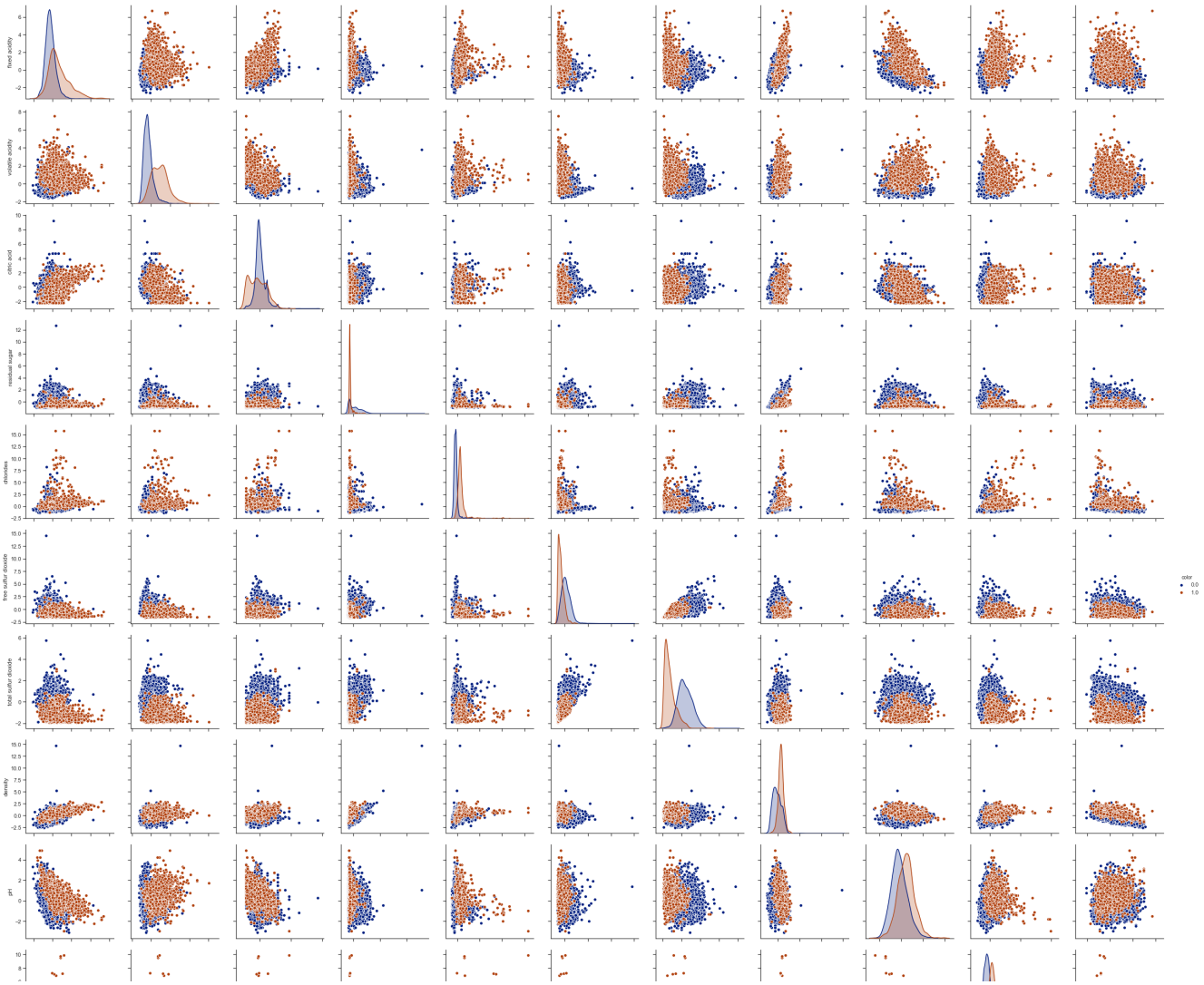


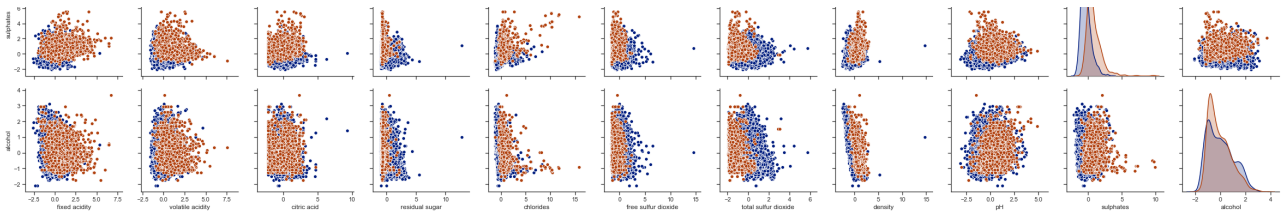

## After Z-Score Normalization

***Z Score Normalization using StandarScaler model from sklearn***

In [38]:

```
# temporary dataset for normalization
wine_dc_norm = pd.DataFrame(wine[DC])
wine_dc_norm['color']=wine_dc_norm['color'].astype(str)
features = wine_dc_norm[D]
scaler = StandardScaler().fit(features.values)
#print(scaler.mean_)
features = scaler.transform(features.values)
wine_dc_norm[D] = features
#print(wine_dc[DC].head())
sns.set_palette("dark")
sns_plot_one = sns.pairplot(wine_dc_norm,hue='color',vars= ['fixed acidity', 'volatile acidity',
'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',
'density', 'pH', 'sulphates', 'alcohol'],height=3)
sns_plot_one.savefig("output_one.png")
```



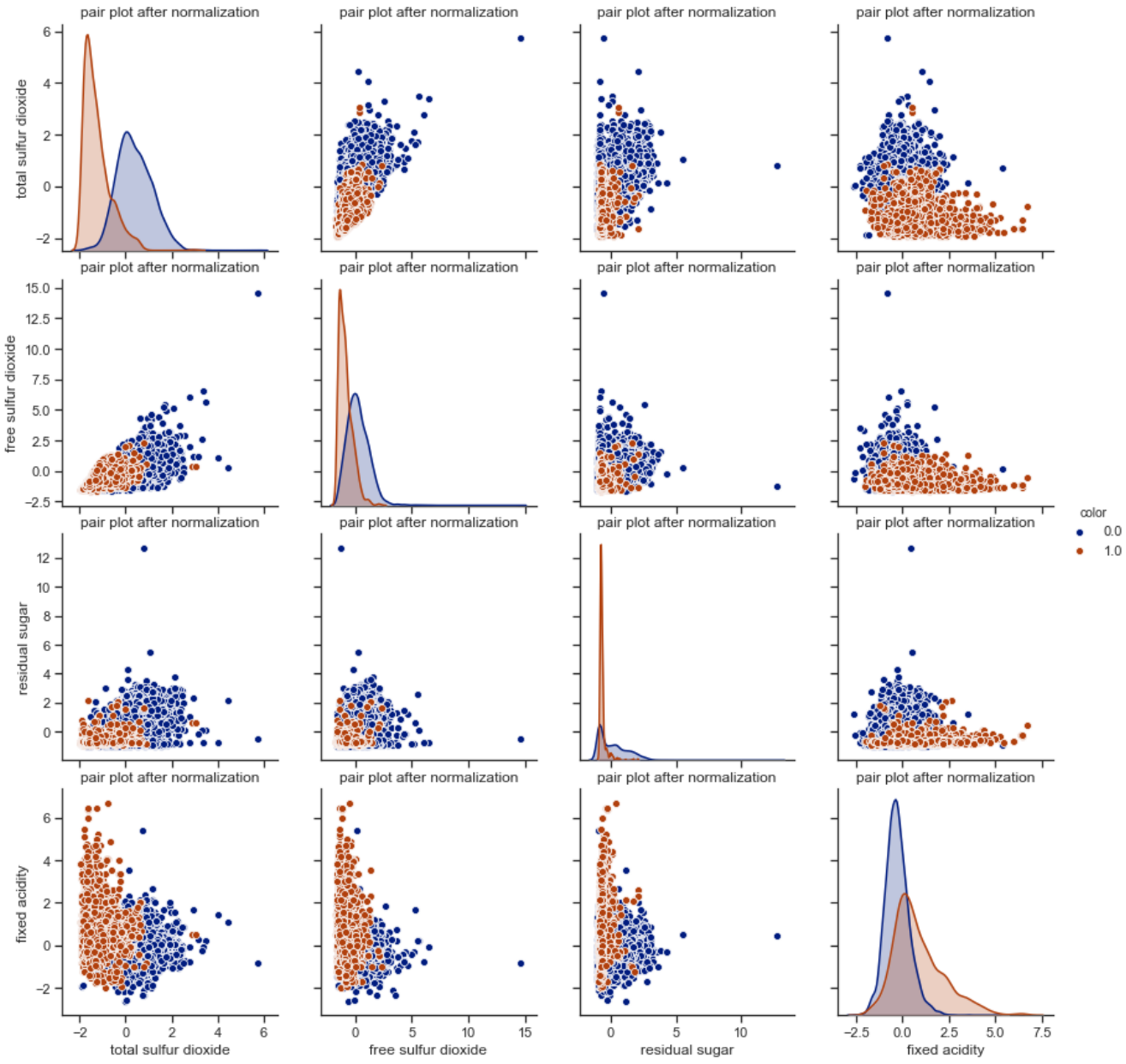


In [39]:

```
sns.set_palette("dark")

#print(wine_dc_one[DC].head())
sns_plot=sns.pairplot(wine_dc_norm,hue='color',vars=['total sulfur dioxide','free sulfur dioxide',
'residual sugar','fixed acidity'],height=3)
#sns_plot.savefig("output.png")

#plt.title("pair plots before normalization")
sns_plot.set(title ="pair plot after normalization")
```

Out[39]:

```
<seaborn.axisgrid.PairGrid at 0x1b1ada31088>
```




***part of code for calucalting and plotting graps for all features and sub- set of selected features for different schemes***

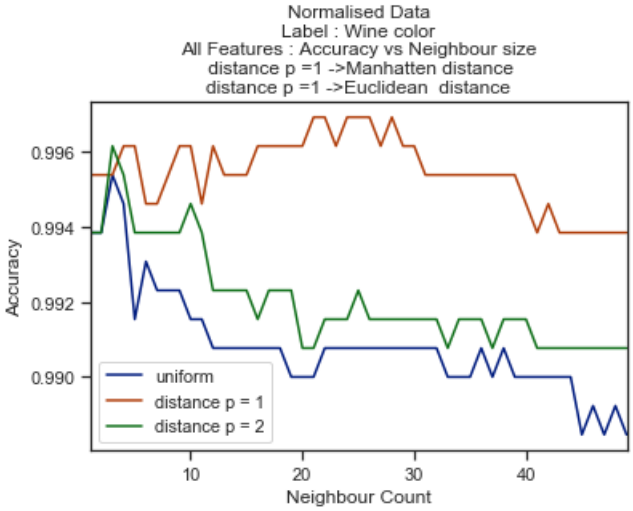
In [40]:

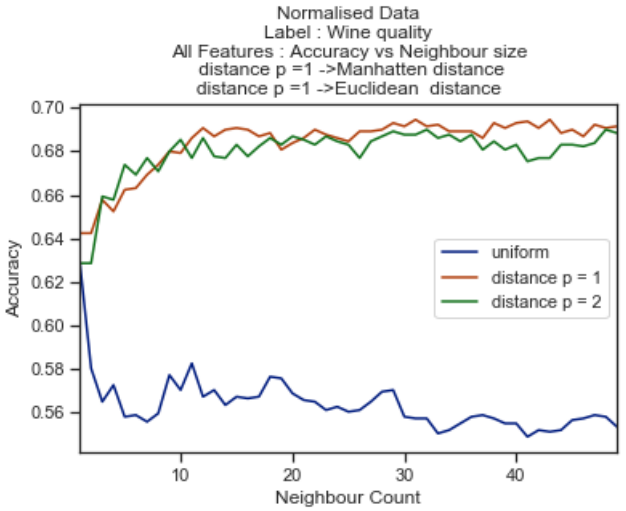
```
################################## ALL FEATURES
#####################################################################
################################## LABEL : WINE COLOR
#############################################################
X = wine[D].values
y = np.ravel(wine[[C]])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

scaler = StandardScaler().fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)

acc_norm = cal_accuracy(X_train_norm,y_train,X_test_norm,y_test)
#plotting line graph
acc_norm[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nAll Features : Accuracy vs Neighbour size\n distan
ce p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
################################## LABEL : WINE QUALITY
#############################################################
X_quality = wine[D].values
y_quality = np.ravel(wine[[L]])

X_train_quality, X_test_quality, y_train_quality, y_test_quality = train_test_split(X_quality, y_qu
ality, test_size=0.2, random_state = ran)

scaler = StandardScaler().fit(X_train_quality)
X_train_norm = scaler.transform(X_train_quality)
X_test_norm = scaler.transform( X_test_quality)

# all features label : wine color
acc_norm_quality = cal_accuracy(X_train_norm,y_train_quality,X_test_norm,y_test_quality)
#plotting line graph
acc_norm_quality[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine quality\nAll Features : Accuracy vs Neighbour size\n dist
ance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
################################## SUB SET OF FEATURES
#####################################################################
################################## LABEL : WINE COLOR
#############################################################
X_sub = wine[['total sulfur dioxide','free sulfur dioxide','residual sugar','fixed acidity']].valu
es
y_sub = np.ravel(wine[[C]])
# deviding data set in train and test sets for the selected sub set of features
X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(X_sub, y_sub, test_size=0.2, ra
ndom_state = ran)

scaler = StandardScaler().fit(X_train_sub)
X_train_norm = scaler.transform(X_train_sub)
X_test_norm = scaler.transform(X_test_sub)

# subset features label : wine color
acc_norm_subfeatures = cal_accuracy(X_train_norm,y_train_sub,X_test_norm,y_test_sub)
acc_norm_subfeatures[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nSub-set Features : Accuracy vs Neighbour size\n di
stance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")

################################## LABEL : WINE COLOR
#############################################################
X_sub_quality = wine[['total sulfur dioxide','free sulfur dioxide','residual sugar','fixed acidity
']].values
y_sub_quality = np.ravel(wine[[L]])
# deviding data set in train and test sets for the selected sub set of features
X_train_sub_quality, X_test_sub_quality, y_train_sub_quality, y_test_sub_quality = train_test_split
(X_sub_quality, y_sub_quality, test_size=0.2, random_state = ran)

scaler = StandardScaler().fit(X_train_sub_quality)
X_train_norm = scaler.transform(X_train_sub_quality)
X_test_norm = scaler.transform( X_test_sub_quality)
```

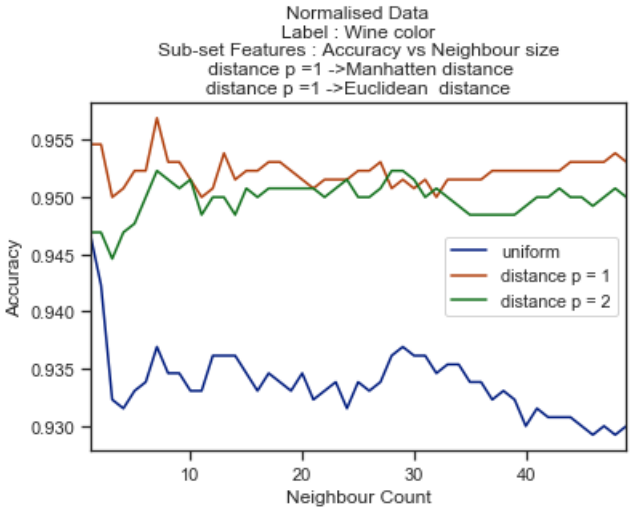
```
# subset features label : wine color
acc_subfeatures_quality =
cal_accuracy(X_train_norm,y_train_sub_quality,X_test_norm,y_test_sub_quality)
acc_subfeatures_quality[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine quality\nSub-set Features : Accuracy vs Neighbour size\n
distance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[40]:

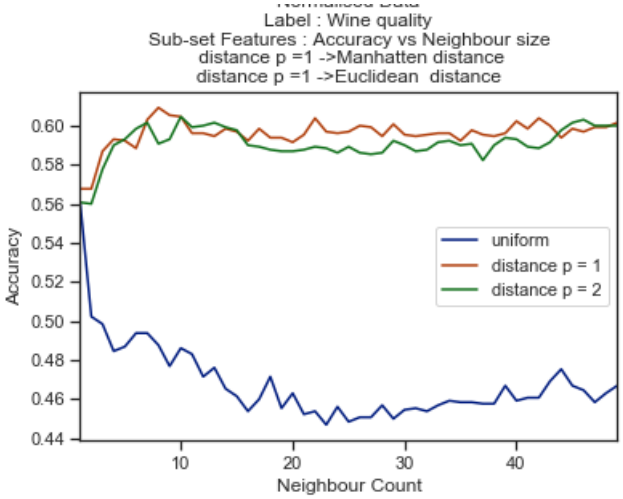
```
Text(0, 0.5, 'Accuracy')
```




Normalised Data



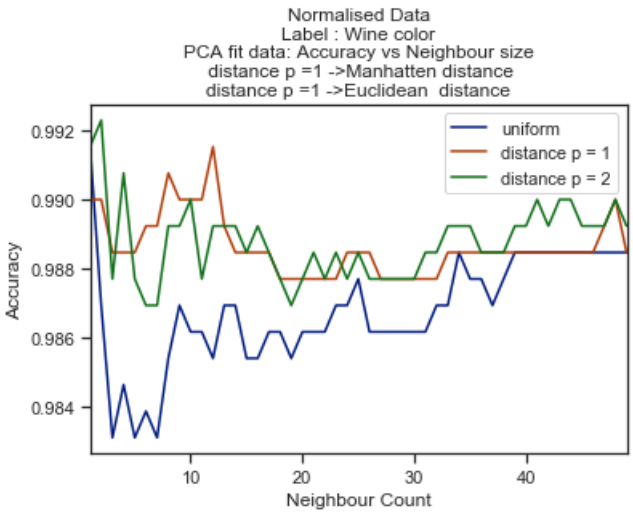

***Using PCA to fit data and use KNN for top 5 principle components and ploting graph for accuracy vs neighbor size***

In [41]:

```
X = wine[D].values
y = np.ravel(wine[[C]])
#print(type(X_train))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

scaler = StandardScaler().fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)
#fitting PCA model only for train set
pca = PCA(n_components=5)
pca.fit(X_train_norm)

#transforming both train and test data using fitted model
X_train_pca = pca.transform(X_train_norm)
X_test_pca = pca.transform(X_test_norm)
#print(type(X_train_pca))

#calling function to caluclate accuracies
acc_pca = cal_accuracy(X_train_pca,y_train,X_test_pca,y_test)
acc_pca[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nPCA fit data: Accuracy vs Neighbour size\n distanc
e p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
# acc_pca.describe()
# acc_pca.head()
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[41]:

```
Text(0, 0.5, 'Accuracy')
```



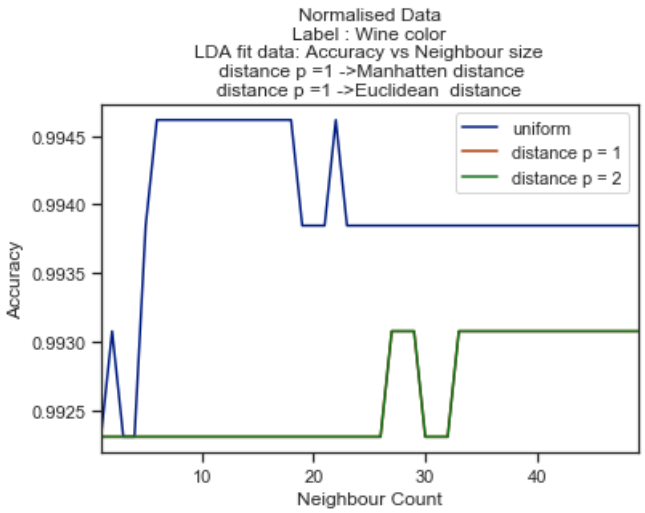
***using Linear Discriminant Analysis to fit the data and then ploting graph for accuracy vs neighbor size***

In [42]:

```
scaler = StandardScaler().fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)

#fitting LinearDiscriminantAnalysis model only for train set
lda = LinearDiscriminantAnalysis(n_components=None)
lda.fit(X_train_norm,y_train)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train_norm)
X_test_lda = lda.transform(X_test_norm)

#calling function to caluclate accuracies
acc_lda = cal_accuracy(X_train_lda,y_train,X_test_lda,y_test)
# print(type(acc_lda))
acc_lda[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nLDA fit data: Accuracy vs Neighbour size\n distanc
e p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
# acc_lda.describe()
# acc_lda.head()
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[42]:

```
Text(0, 0.5, 'Accuracy')
```




***PCA and LDA for label 'quality'***

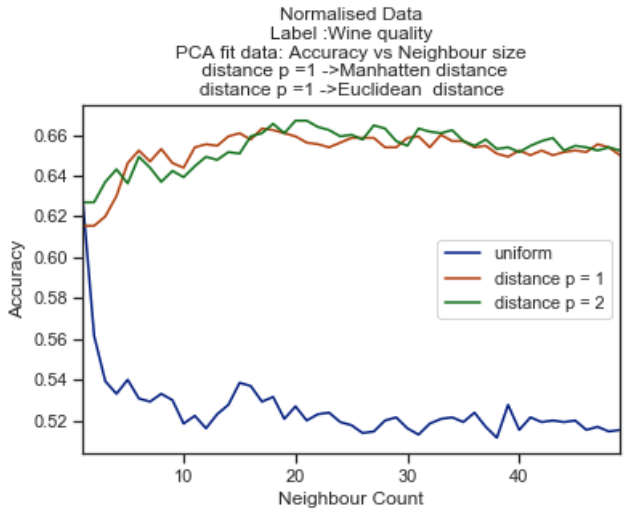
In [43]:

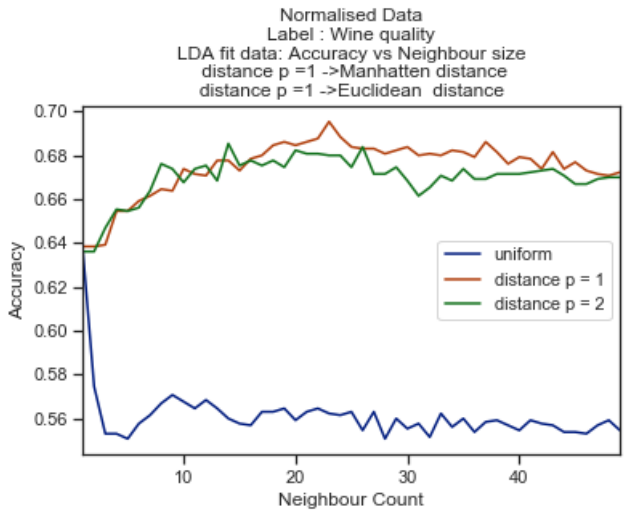
```
X = wine[D].values
y = np.ravel(wine[[L]])
#print(type(X_train))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

scaler = StandardScaler().fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)

#fitting PCA model only for train set
pca = PCA(n_components=5)
pca.fit(X_train_norm)

#transforming both train and test data using fitted model
X_train_pca = pca.transform(X_train_norm)
X_test_pca = pca.transform(X_test_norm)
```

```
#print(type(X_train_pca))

#calling function to caluclate accuracies
acc_pca = cal_accuracy(X_train_pca,y_train,X_test_pca,y_test)
acc_pca[1:].plot.line()
plt.title("Normalised Data\nLabel :Wine quality\nPCA fit data: Accuracy vs Neighbour size\n distan
ce p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
# acc_pca.describe()
# acc_pca.head()
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")

lda = LinearDiscriminantAnalysis(n_components=None)
lda.fit(X_train_norm,y_train)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train_norm)
X_test_lda = lda.transform(X_test_norm)

#calling function to caluclate accuracies
acc_lda = cal_accuracy(X_train_lda,y_train,X_test_lda,y_test)
# print(type(acc_lda))
acc_lda[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine quality\nLDA fit data: Accuracy vs Neighbour size\n dista
nce p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
# acc_lda.describe()
# acc_lda.head()
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[43]:

```
Text(0, 0.5, 'Accuracy')
```



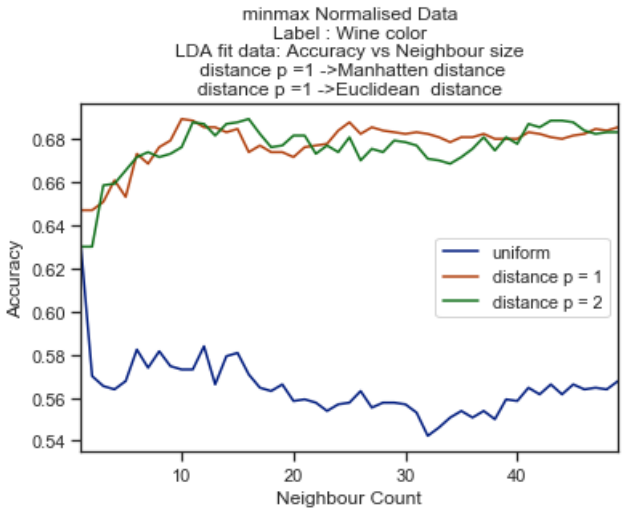
**Trying MINMAX scaler to check if it can out performe (Z-norm,Manhatten) Scheme**

In [44]:

```
scaler_minmax = MinMaxScaler().fit(X_train)
X_train_minmax = scaler_minmax.transform(X_train)
X_test_minmax = scaler_minmax.transform(X_test)

acc_minmax = cal_accuracy(X_train_minmax,y_train,X_test_minmax,y_test)
acc_minmax[1:].plot.line()
plt.title("minmax Normalised Data\nLabel : Wine color\nLDA fit data: Accuracy vs Neighbour size\n
distance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[44]:

```
Text(0, 0.5, 'Accuracy')
```




## Trials for selecting best features which can outperform dataset with all features

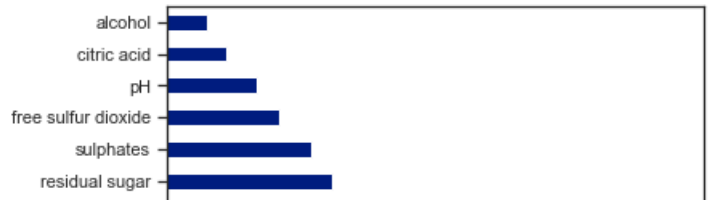
***using Feature Importance from ExtraTreeClssifiers module***

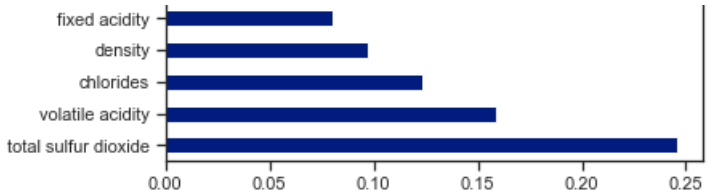
Feature importance gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable.

In [45]:

```
#using teh normalised data before feature selection as it gives better results
data = wine_dc_norm
X = data.iloc[:,0:11]  #independent columns
y = data.iloc[:,-1]    #target color column
model = ExtraTreesClassifier(n_estimators=100)
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(11).plot(kind='barh')
plt.show()
```

```
[0.08015384 0.15887316 0.02868973 0.0792929  0.1230181  0.0542896
 0.24543077 0.09758618 0.04317888 0.06978716 0.01969969]
```



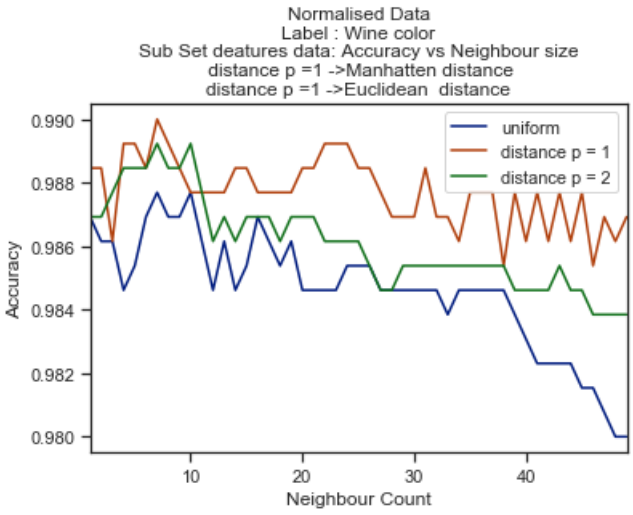



In [46]:

```
# from the above graph we can observe that below features are more valuble
best_features = ['total sulfur dioxide','volatile acidity','chlorides','density']
X = wine[best_features].values
y = np.ravel(wine[[C]])
#print(type(X_train))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

scaler = StandardScaler().fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)

acc_chi = cal_accuracy(X_train_norm,y_train,X_test_norm,y_test)
acc_chi[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nSub Set deatures data: Accuracy vs Neighbour size\
n distance p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[46]:

```
Text(0, 0.5, 'Accuracy')
```




***The above method did not out perform Z-norm and manhatten scheme for a range of 15 values***

***Trying LDA for the Data of SUB SET of Features***

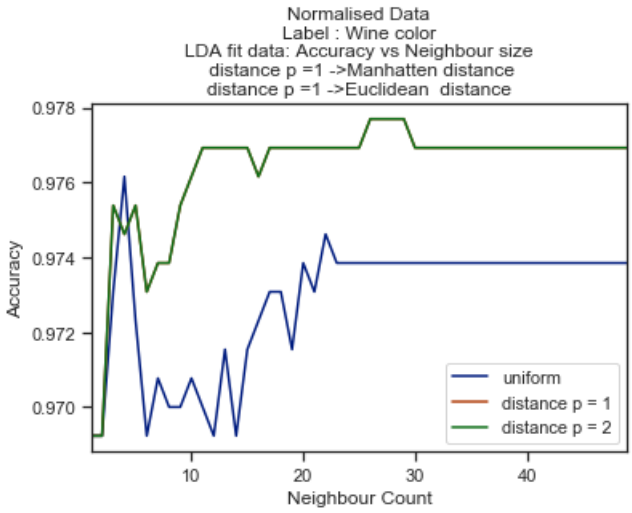
In [47]:

```
lda = LinearDiscriminantAnalysis(n_components=None)
lda.fit(X_train_norm,y_train)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train_norm)
X_test_lda = lda.transform(X_test_norm)

#calling function to caluclate accuracies
acc_lda = cal_accuracy(X_train_lda,y_train,X_test_lda,y_test)
# print(type(acc_lda))
acc_lda[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nLDA fit data: Accuracy vs Neighbour size\n distanc
e p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
# acc_lda.describe()
```

```
# acc_lda.describe()
# acc_lda.head()
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[47]:

```
Text(0, 0.5, 'Accuracy')
```




***Even This scheme did not out perform Z-norm and Manhatten Scheme with all features dataset***

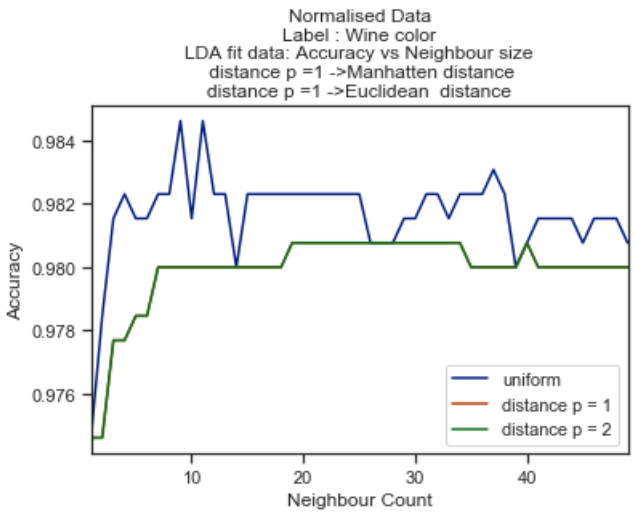
***Trying The following order : normalization ,PCA and then LDA for the dataset with full features***

In [48]:

```
X = wine[D].values
y = np.ravel(wine[[C]])

# deviding data set in train and test sets
ran = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)

#normalization
scaler = StandardScaler().fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)

#applying PCA model
pca = PCA(n_components=5)
pca.fit(X_train_norm)

#transforming both train and test data using fitted model
X_train_pca = pca.transform(X_train_norm)
X_test_pca = pca.transform(X_test_norm)

#then applying LDA
lda = LinearDiscriminantAnalysis(n_components=None)
lda.fit(X_train_pca,y_train)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train_pca)
X_test_lda = lda.transform(X_test_pca)

#calling function to caluclate accuracies
acc_lda = cal_accuracy(X_train_lda,y_train,X_test_lda,y_test)
acc_lda[1:].plot.line()
plt.title("Normalised Data\nLabel : Wine color\nLDA fit data: Accuracy vs Neighbour size\n distanc
e p =1 ->Manhatten distance\ndistance p =1 ->Euclidean  distance")
plt.xlabel("Neighbour Count")
plt.ylabel("Accuracy")
```

Out[48]:

```
Text(0, 0.5, 'Accuracy')
```




***Even This scheme did not out perform Z-norm and Manhatten Scheme with all features dataset***

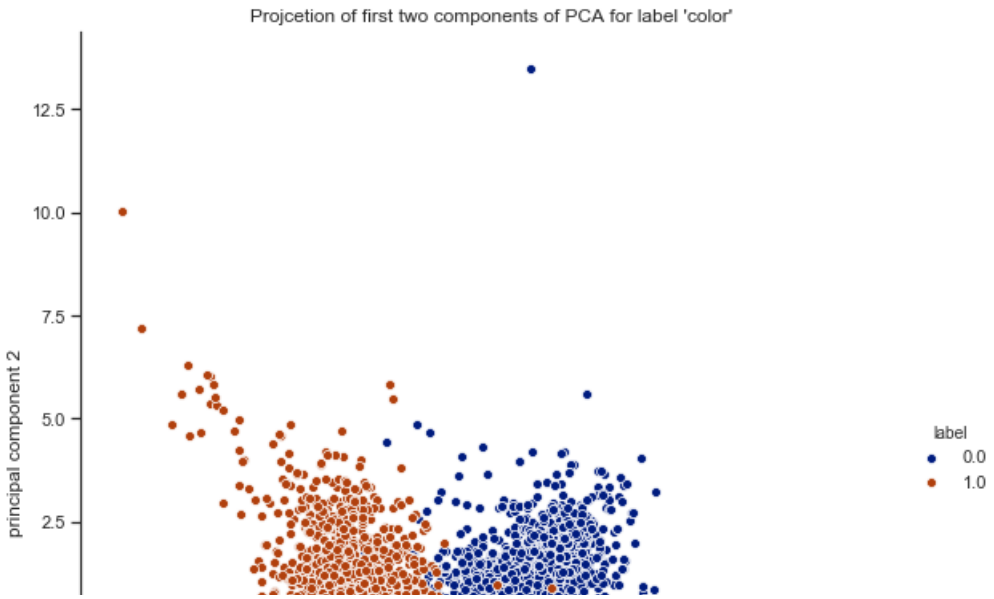
## Projecting first two components of PCA

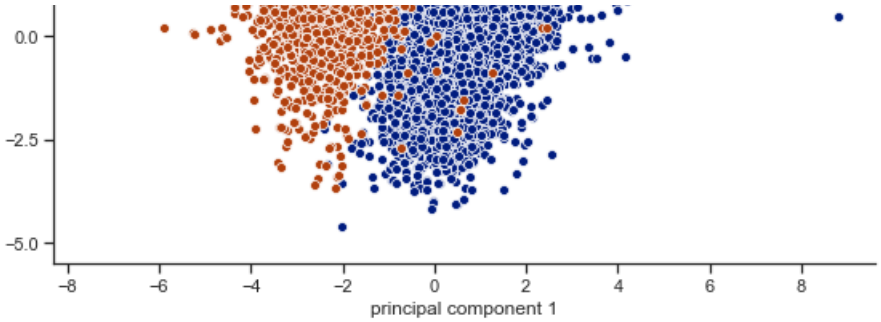
In [49]:

```
df = wine[D].values
c = np.ravel(wine[[C]])
l = np.ravel(wine[[L]])

scaler = StandardScaler().fit(df)
df_norm = scaler.transform(df)
# X_test_norm = scaler.transform(X_test)

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df_norm)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf['label'] = wine[C].values
# print(principalDf.shape)
sns.set_palette("dark")
sns.pairplot(principalDf,hue='label',x_vars=['principal component 1'],y_vars=['principal component
2'],height=8).set(title ="Projcetion of first two components of PCA for label 'color'")
```

Out[49]:

```
<seaborn.axisgrid.PairGrid at 0x1b19ef21ec8>
```





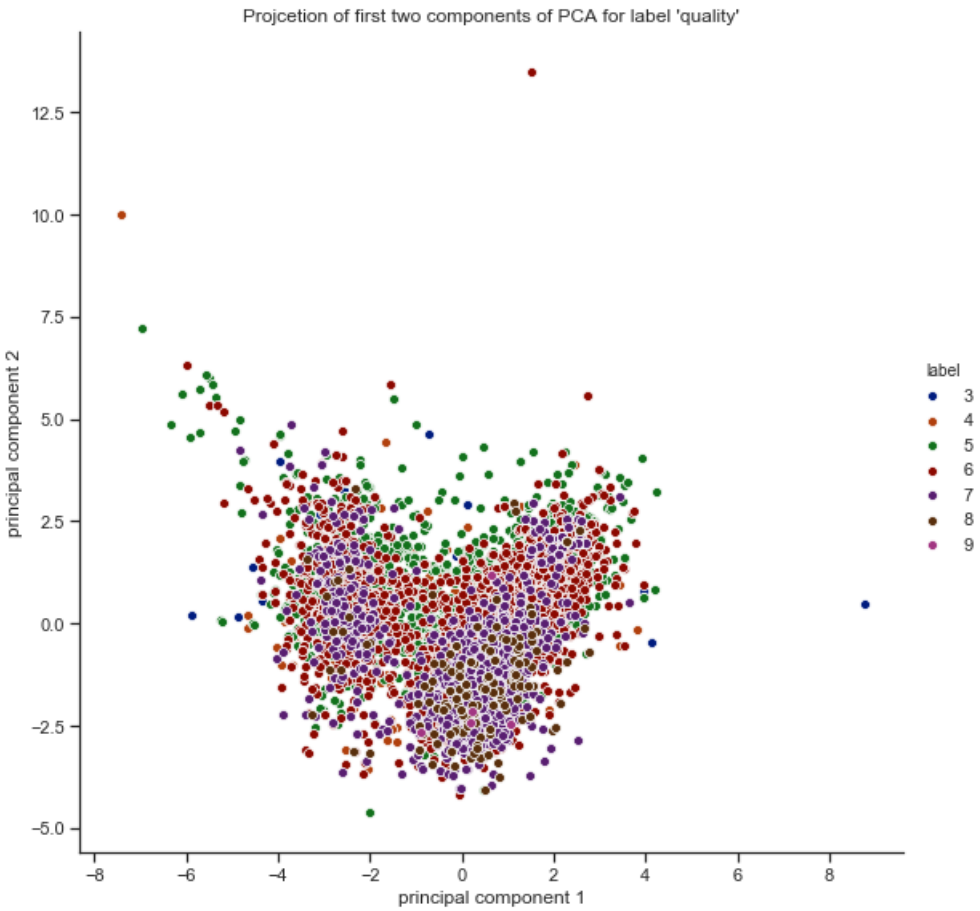

In [50]:

```
principalDf['label'] = wine[L].values
print(type(principalDf))
sns.set_palette("dark")
sns.pairplot(principalDf,hue='label',x_vars=['principal component 1'],y_vars=['principal component
2'],height=8).set(title ="Projcetion of first two components of PCA for label 'quality'")
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[50]:

```
<seaborn.axisgrid.PairGrid at 0x1b19f0e4a08>
```




**Projecting first two components of LDA**

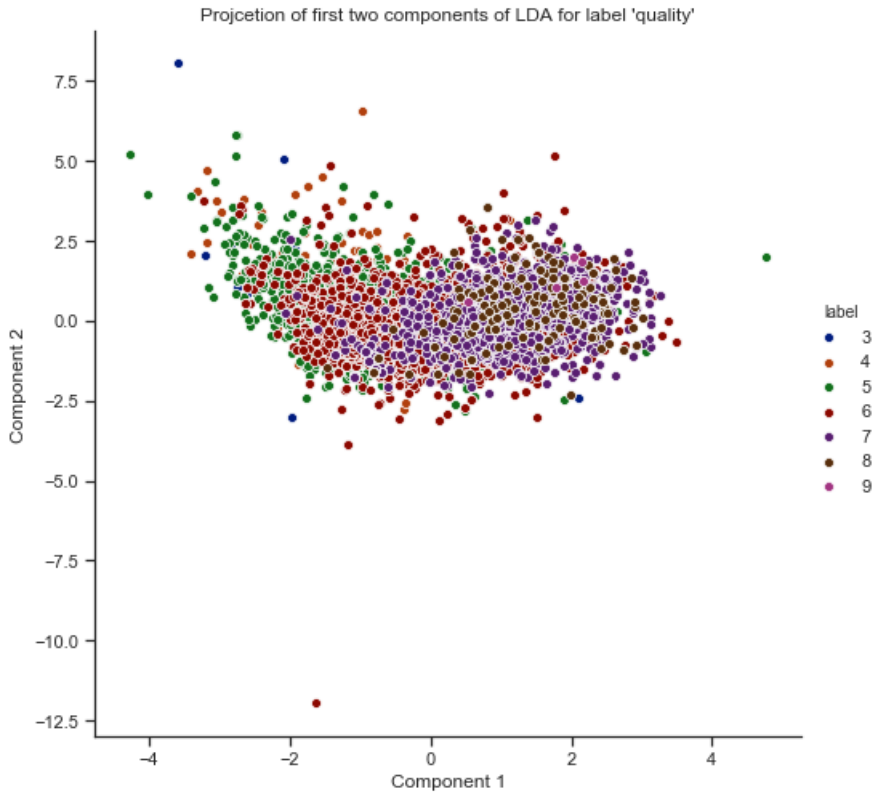
In [51]:

```
X = wine[D].values
y = np.ravel(wine[[L]])

# deviding data set in train and test sets
# ran = 42
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = ran)
```

```
scaler = StandardScaler().fit(X)
X_train_norm = scaler.transform(X)


#fitting LinearDiscriminantAnalysis model only for train set
lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(X_train_norm,y)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train_norm)
# X_test_lda = lda.transform(X_test_norm)
# print(X_train_lda.shape)
# print(type(X_train_lda))
X_train_lda = pd.DataFrame(data = X_train_lda, columns = ['Component 1', 'Component 2'])
# print(type(X_train_lda))
# print(X_train_lda.shape)
# print(wine[L].shape)
X_train_lda['label']=wine[L].values
sns.set_palette("dark")
sns.pairplot(X_train_lda,hue='label',x_vars=['Component 1'],y_vars=['Component 2'],height=7).set(ti
tle ="Projcetion of first two components of LDA for label 'quality'")
# sns_p.set(title ="Projcetion of first two components of LDA")
```

Out[51]:

```
<seaborn.axisgrid.PairGrid at 0x1b19ef3f8c8>
```

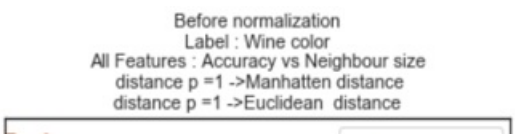


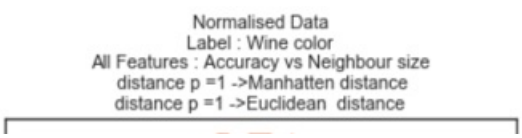

# Analysis and Discussion

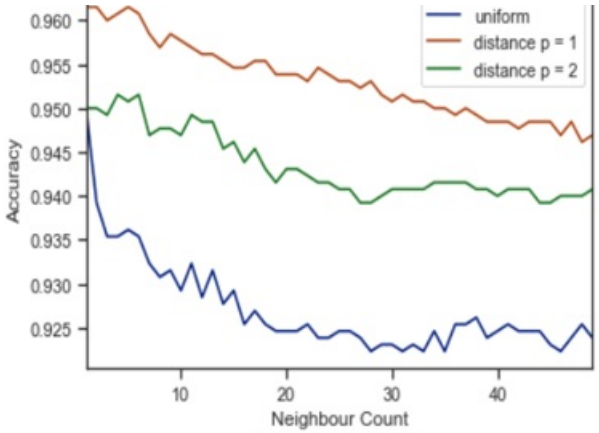
### k Plots:

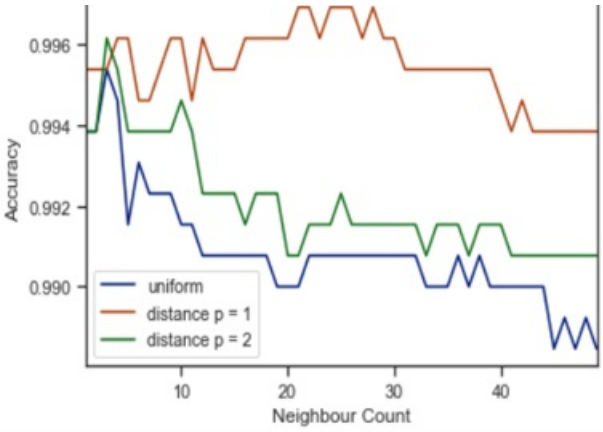
***The below plots are for different cases indicated by the plot titles***

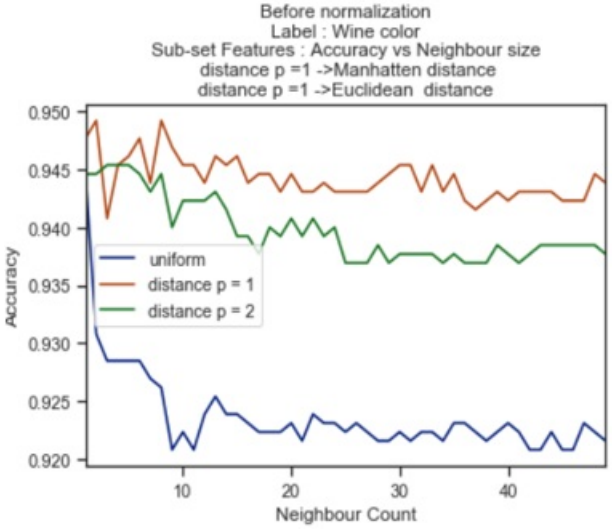
The selected sub-set of features are 'total sulfur dioxide','free sulfur dioxide','residual sugar','fixed acidity'

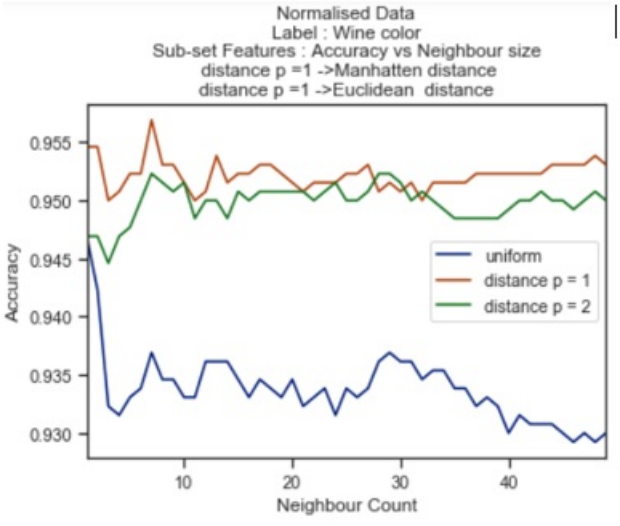


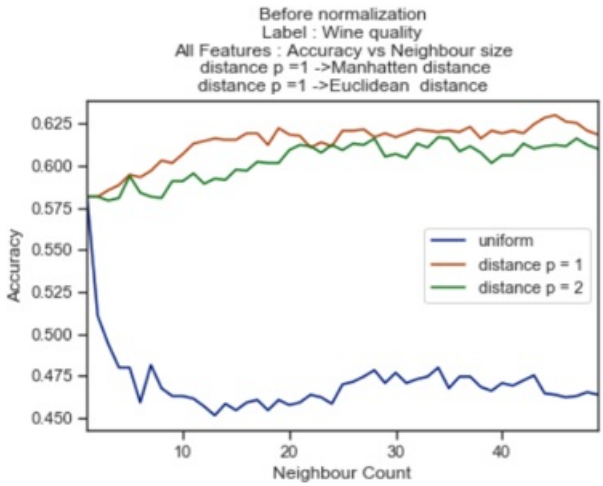



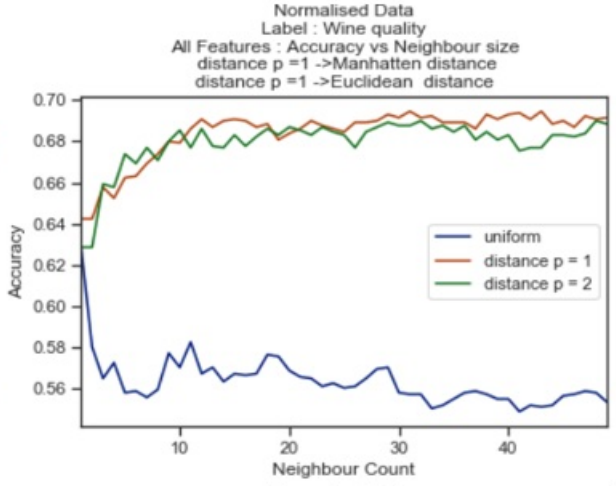


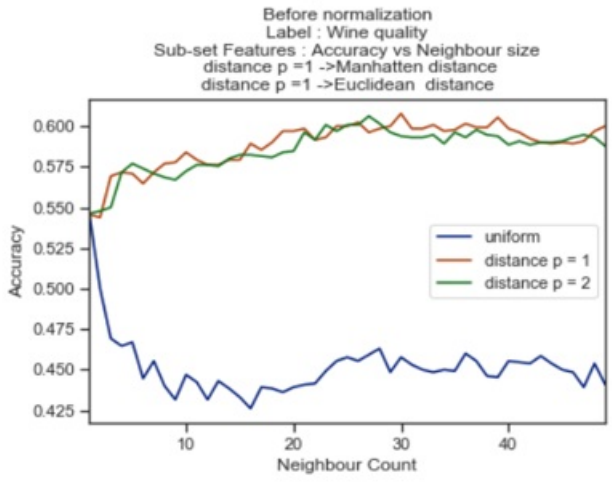



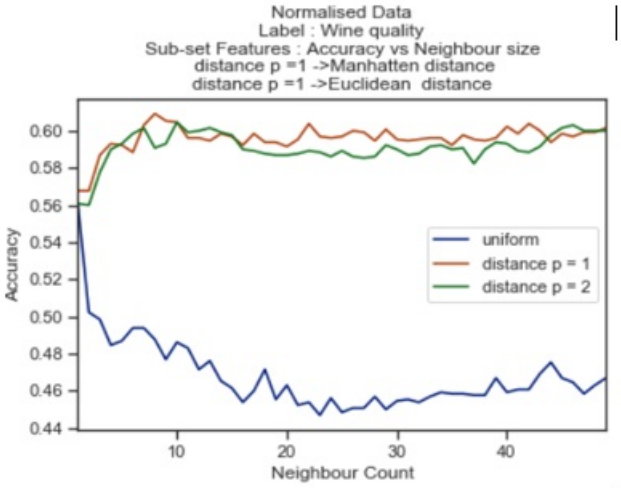




***From the above graphs we can observe that :***

- The acuuracy of the perdiction is always greater when the data is normalized than that of teh accuracies obtained without normalization
- The highest accuracy obtained is 99.6923 % when the data is normalized using Z-norm and when the point weighting schem is using "distance based weight with manhattan distance"
- The accuaracy rates when the lable or target is 'wine color' is very high compared to label 'wine quality' as the umber of classes in color are only two where are in quality has a huge number.
- For Z-norm and Manhatten scheme ,KNN ha better performance when the neighbour Count is in the range of (15-30).
- Distance based weighing schemes always out performed the normal weighing schemes.
- Manhatten Scheme seems to perform better than Euclidean scheme most of teh time.

## Features and Selected Features:

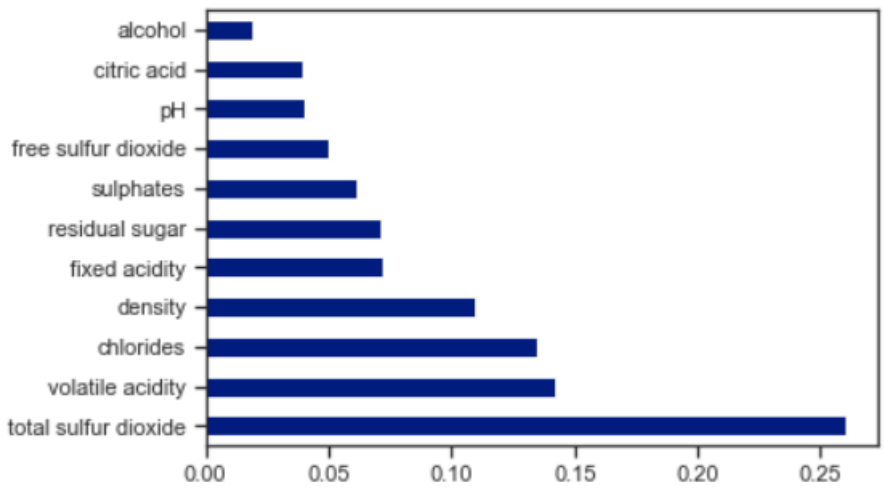
***Analysis using univariate selection***

| | Specs | Score |
|---|---|---|
| 6 | total sulfur dioxide | 87946.248732 |
| 5 | free sulfur dioxide | 14913.554060 |
| 3 | residual sugar | 3287.056863 |
| 0 | fixed acidity | 358.497091 |
| 1 | volatile acidity | 221.064880 |
| 9 | sulphates | 64.271565 |
| 4 | chlorides | 37.398554 |
| 2 | citric acid | 15.118752 |
| 8 | pH | 5.652320 |
| 7 | density | 0.008961 |

- From the above scores we can observe the features which has strong relationship with the variable.This has been performed on non -normalized data.
- We can observe that 'total sulfur dioxide','free sulfur dioxide','residual sugar' has scores far greater than the scores obtained by the remaining features.
- From the graphs of KNN accuracies ,it can be observed that even when the considered features are 'total sulfur dioxide','free sulfur dioxide','residual sugar' and 'fixed acidity' the accuracy of the model did not drop much.In case when you consider reducing teh computational costs of the algorithm ,these sub- set of features can be selected to provide good accuracy and good computational efficiencies too. ###### - For this Dataset ,selecting Sub-set of features did not improve the accuracy of the model when compared to considering all features.

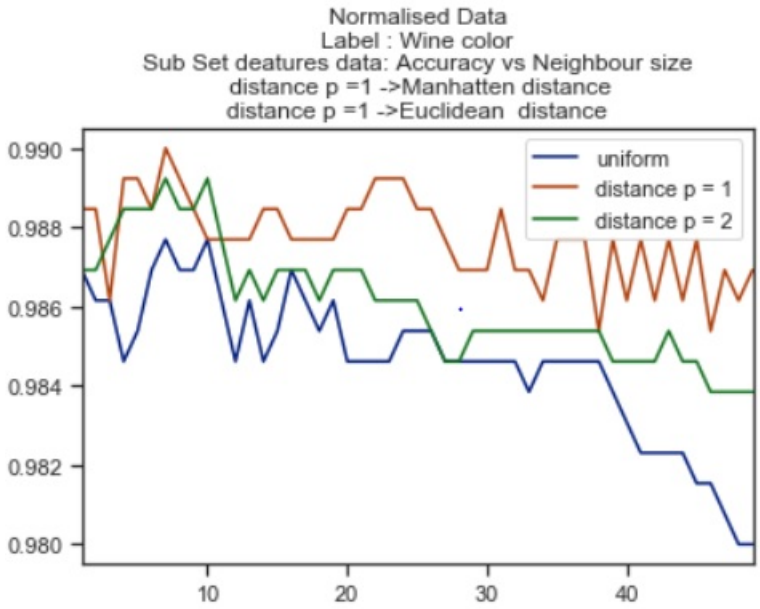
***Feature Importance Selection Method***

Feature importance gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable




- The above selection has been made for the normalized data

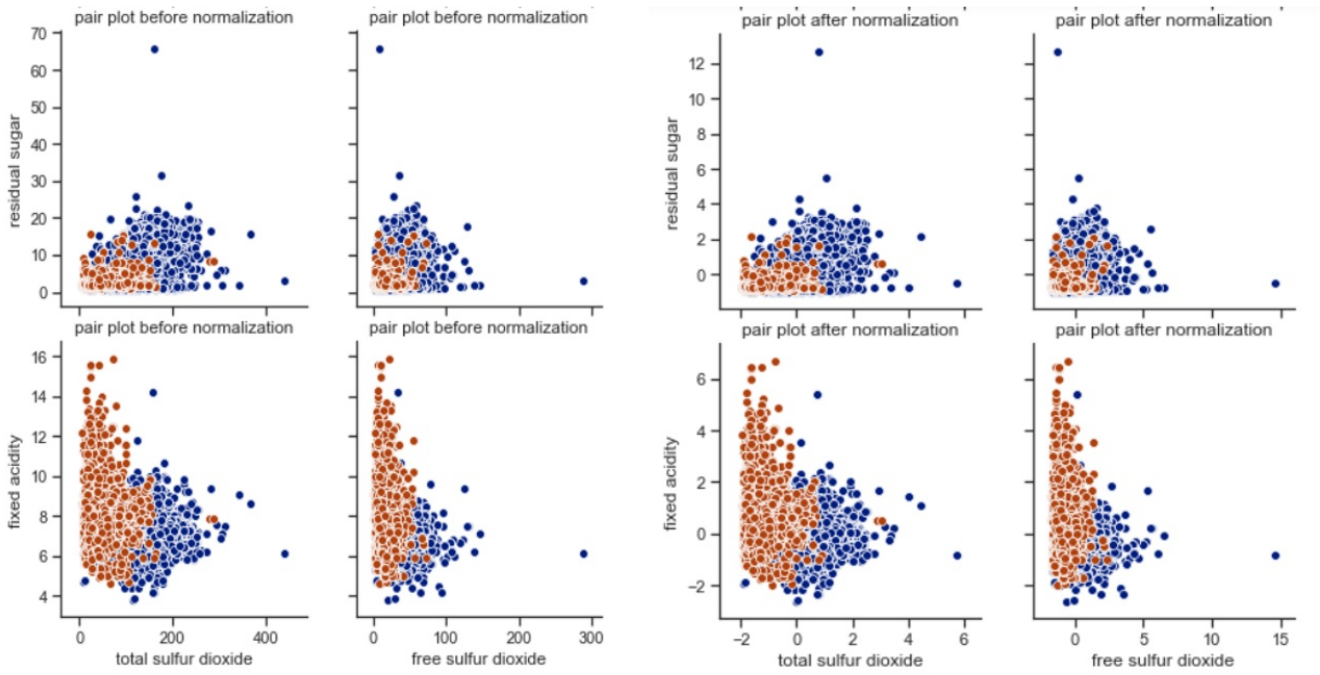
- The above selection has been made for the normalized data
- From the above graph,we can observe that 'total sulfur dioxide','volatile acidity','chlorides' and 'density' has more relavence to the output that the other features.




Even with the selection of only 4 feature , the accuracy of the model did not drop much.And this normalised data feature selection performed better than the non-normalized data deature selection

***Selection of sub set of features did not outperform PCA or LDA.But the accuray drop is less than 1% when compared to both the cases***

**Pair Plots comparison:**




Considering only 4 plots as it is diffcult to compare 11 * 11 plots. From the above graphs before and after normalization ,we can observe :

- We can notice Z-score normalization 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 means of the features means.
- Z-score normalization is a strategy of normalizing data that avoids this outlier issue.

## PCA vs. LDA:

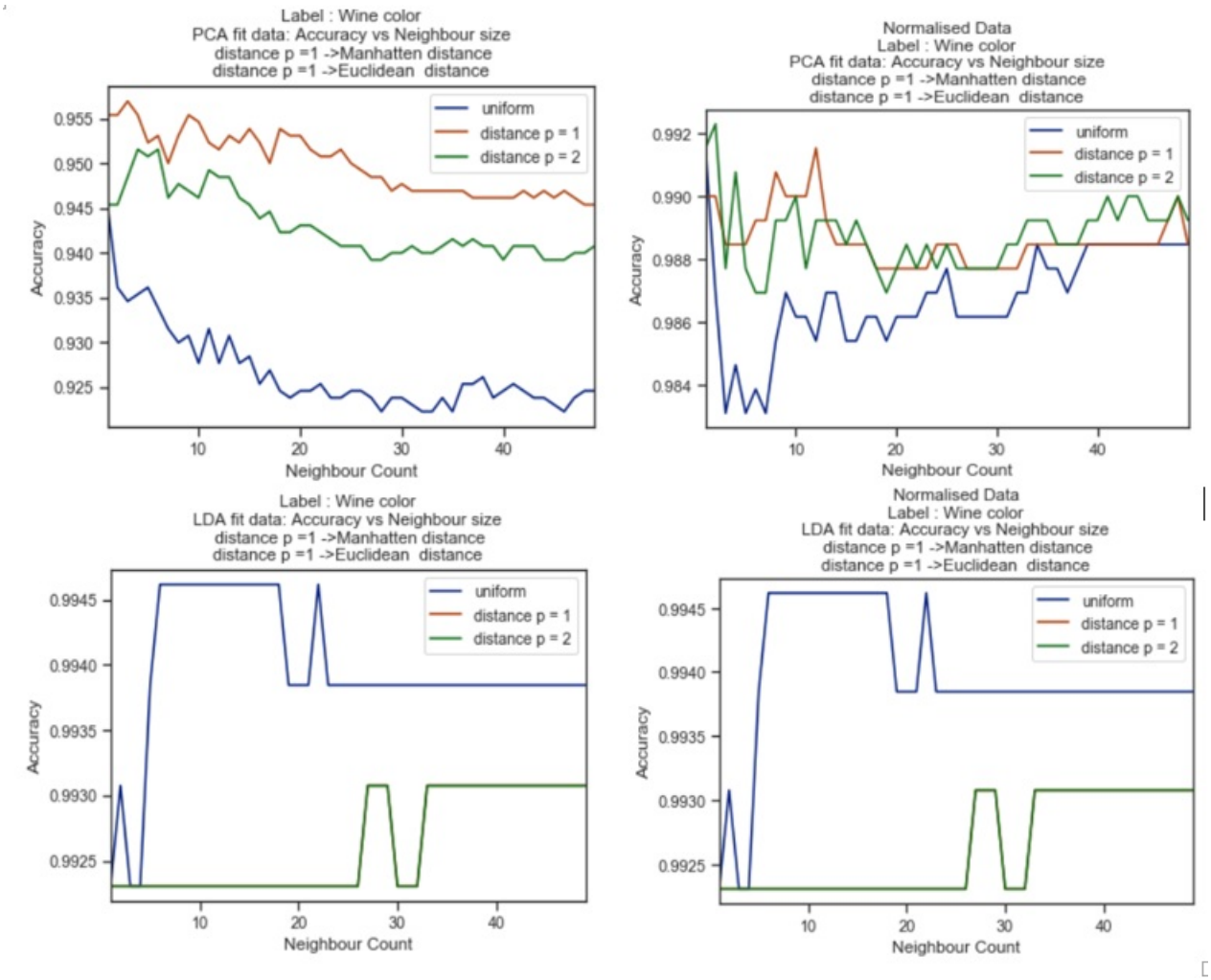
***Did either of these methods help in this situation?Which worked better for the task?***

- Neither PCA nor LDA has out performed the Z-norm - Manhatten scheme in terms of accuracy.But the accuracy drop is very less i.e. around 0.5%.
- For this data set they did not provide much improvement in the model.

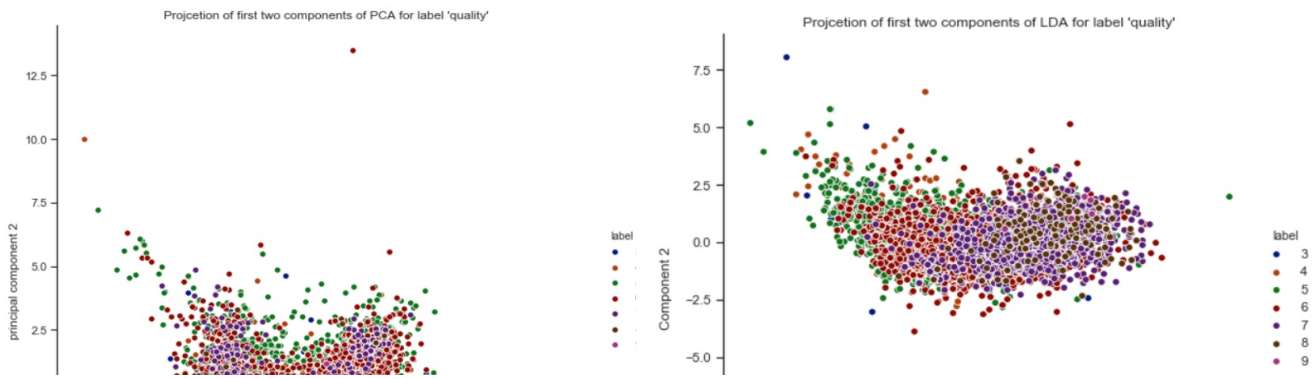
***Did normalization impact the performance of either of them?***

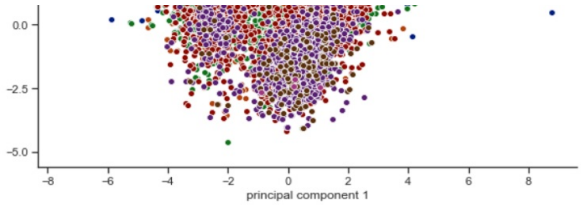
- Normalization has impacted the performance of PCA but not LDA.
- The reults obtained for LDA before and afer normalization are same.

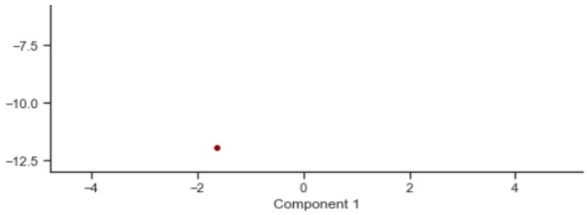
  Reasons learned from : https://stats.stackexchange.com/questions/109071/standardizing-features-when-using-lda-as-a-pre-processing-step




***Plot project he data on the first two components for PCA and LDA and colour the points by the two labels ('quality','color') so four plots in total. How does this compare or inform your understanding of the data from the pairplots or other results?***

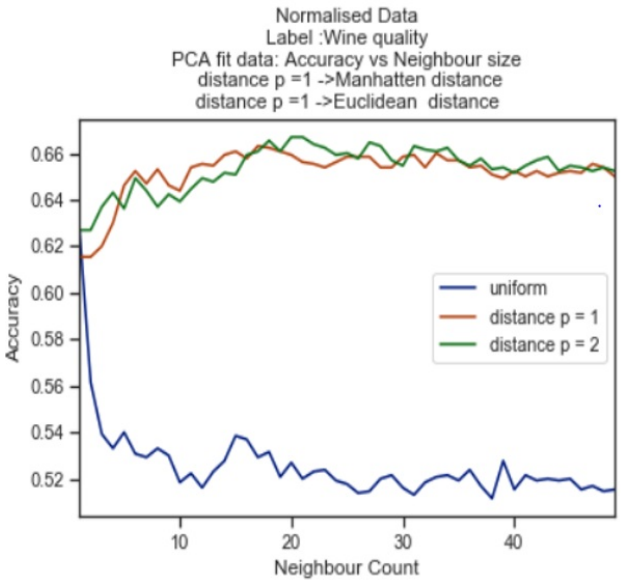


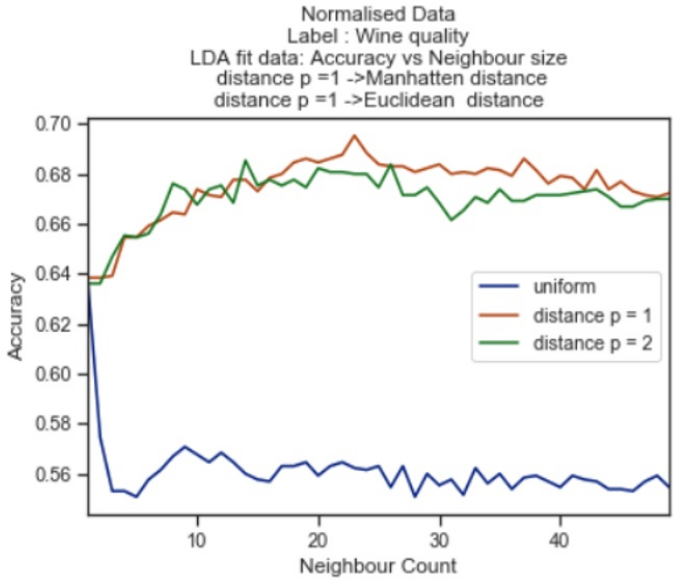



From the above graphs of projects of first two components of PCA and LDA for labels 'quality' we can observe that the LDA seems to cluster the data little bit better than that od PCA.More clear clustering can be seen in LDA ,whereas PCA doesnt tend to do that better.

As per the perious analysis, for normalised data the accuracy given by LDA for label 'quality' is better than that of PCA as shown in below comparision.As LDA clusters are more clearer than PCA ,this accuracy can be thus explained.




Refernces :


- https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- https://towardsdatascience.com/dive-into-pca-principal-component-analysis-with-python-43ded13ead21
- https://kite.com/python/examples/370/numpy-compute-the-eigenvalues-of-a-matrix
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html


# Question 2:

## 2.2 Principal Component Analysis (PCA)

### 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.**

In [113]:

```
# libraries
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import scipy.linalg as la
import chart_studio.plotly as py
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.patheffects as PathEffects
```

```
import time
```

In [114]:

```
# loading dataset
dataset = pd.read_csv("DataB.csv")

X = dataset.iloc[:,1:785]
Y = dataset.iloc[:,-1]

X_std = StandardScaler().fit_transform(X)
```

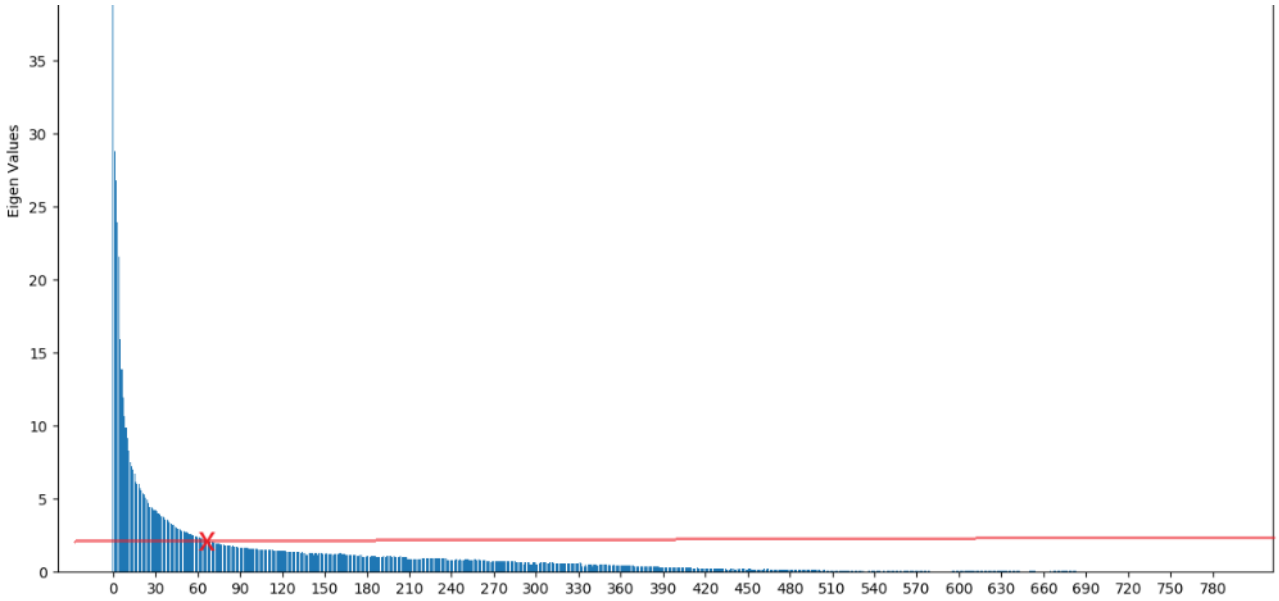
In [124]:

```
X_std = StandardScaler().fit_transform(X)
#caluclating covariance matrix using numpy cov function
cov_matrix = np.cov(X_std.T)
# print(cov_mat)
print(cov_matrix.shape)

#decomposing the covariance matrix into eigen values and eigen vectore using numpy eig method
eigen_values, eigen_vectors = np.linalg.eig(cov_matrix)
# print(eigen_vectors.shape)
# print(eigen_vectors[:].shape)
# print(eigen_vectors[:20].shape)
# print(eig_vecs)
# print(eig_vals)

#reating an eigen pair using eigen values and eigen vectors
eigen_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i]) for i in range(len(eigen_values))]

#sorting the eigen pairs using eigen values and making them sorted in descending order
eigen_pairs.sort()
eigen_pairs.reverse()

print('Eigenvalues in desc order:')
for i in eig_pairs:
    print(i[0])

# creating labels for principal components
labels = ['PC %s' %i for i in range(1,785)]
y_pos = np.arange(len(labels))
# print(eigen_values)

#plotting the graph
plt.figure(figsize=(15,10))
# plt.bar(y_pos, eigen_values, align='center')
# plt.xticks(y_pos, objects)
# as 784 pc names cannot be labeled using steps of 30 to indicate their number.
plt.xticks(np.arange(0, 785, step=30))
plt.yticks(np.arange(0, 60, step=5))
plt.ylabel('Eigen Values')
plt.title('Scree Plot')
plt.show()
```

```
(784, 784)
```

***Descision of cutoff:***

A scree plot shows the the eigenvalues in a downward curve, ranging from the largest to the smallest. According to the scree test, the graph's "elbow" where the individual values tend to be levelling off is identified and variables or components to the left of this point can be retained as significant.




Soure for selection descision : https://en.wikipedia.org/wiki/Scree_plot

***Total variance explained by slecting 65 principal components as choosen above is : 56.01394993613066***

In [119]:

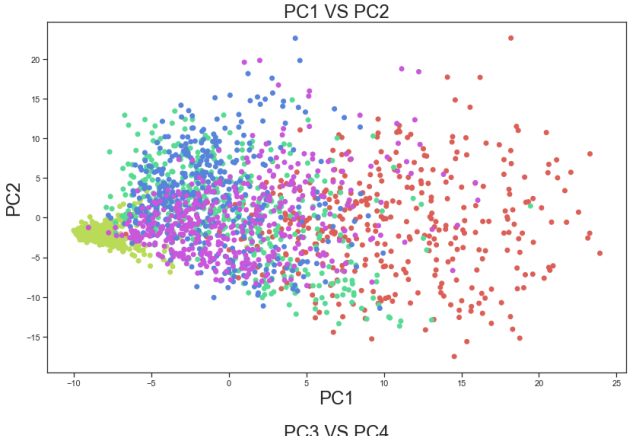
```
tot = sum(eigen_values)
var_explained = [(i / tot)*100 for i in sorted(eigen_values, reverse=True)]
print("Explained Variance for top 20 eigen vectors {0}".format(sum(var_explained[:65])))
```

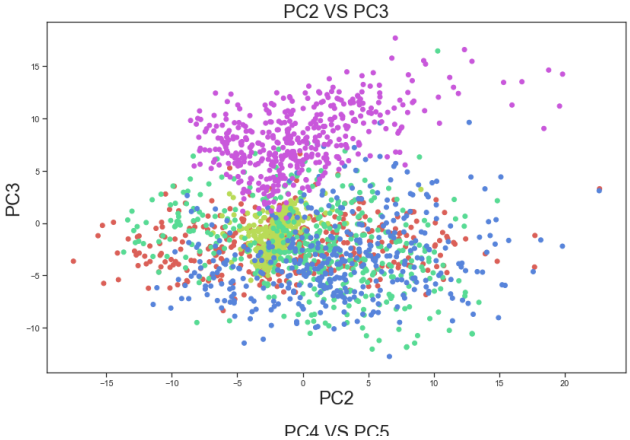
```
Explained Variance for top 20 eigen vectors 56.01394993613066
```

**2. Using subplot in python matplotlib, plot the scatter plot of the projected data with the top 20 eigenvalues (although PCA does not use labels but use colors and legend to show the class instances). Is there a clear point where you could cut off the dimensions? Compare your analysis with the analysis from previous section.**

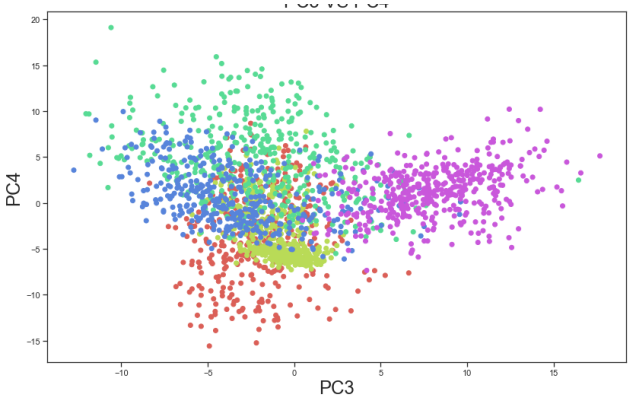
In [120]:

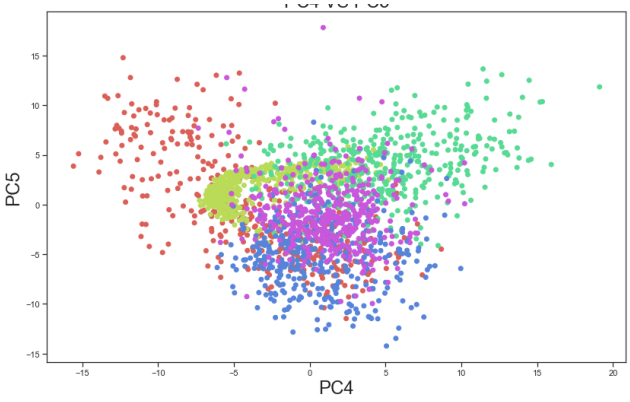
```
pca_model = PCA(n_components=20).fit(X_std)
pca = pca_model.transform(X_std)
# print(pca_model.explained_variance_ratio_)
# print(pca.shape)
num_classes = len(np.unique(Y))
palette = np.array(sns.color_palette("hls", num_classes))
x=0;
fig = plt.figure(figsize=(30, 100))
for i in range(1,20):
    plt.subplot(10, 2, i)
    if x<19 :
        plt.scatter(pca[:, [x]],pca[:, [x+1]],c=palette[Y.astype(np.int)])
        plt.title("PC"+str(x+1)+" VS "+"PC"+str(x+2),fontsize=24)
        plt.xlabel("PC"+str(x+1),fontsize=24)
        plt.ylabel("PC"+str(x+2),fontsize=24)
        x= x +1;
```



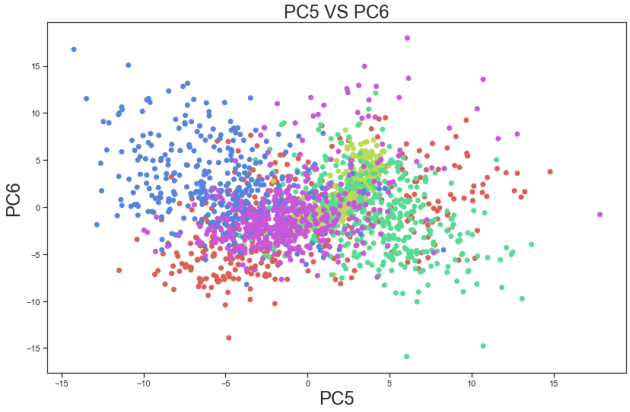




PC3 VS PC4 PC4 VS PC5

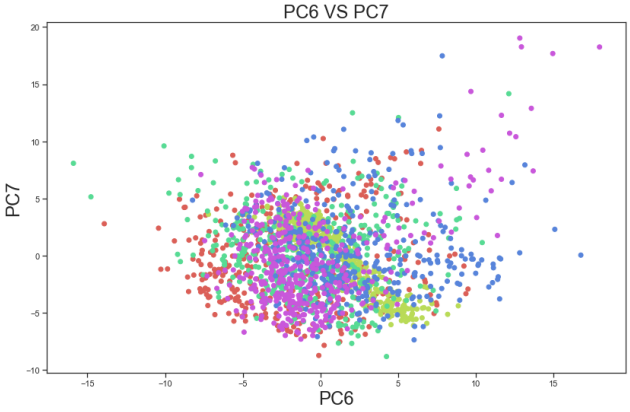


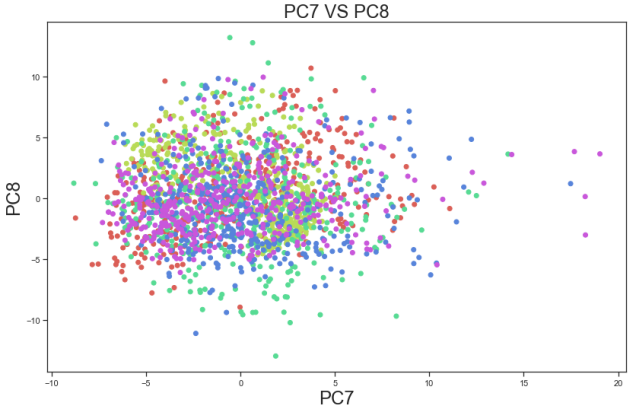

PC5 VS PC6



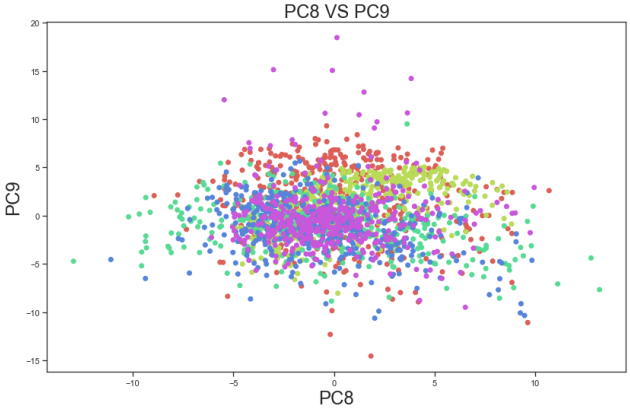

PC6 VS PC7



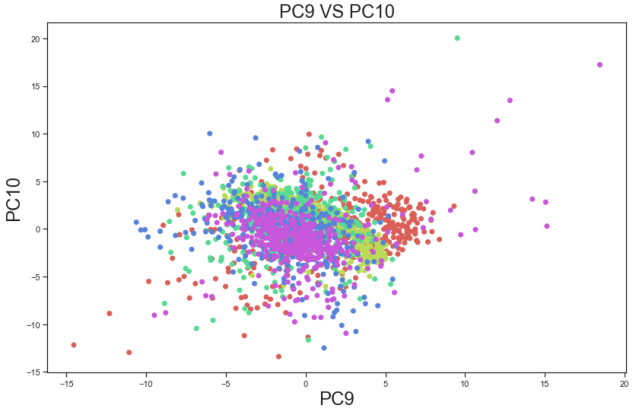

PC7 VS PC8



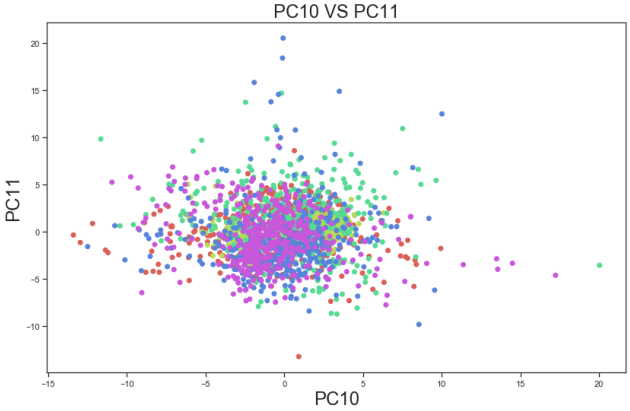

PC8 VS PC9



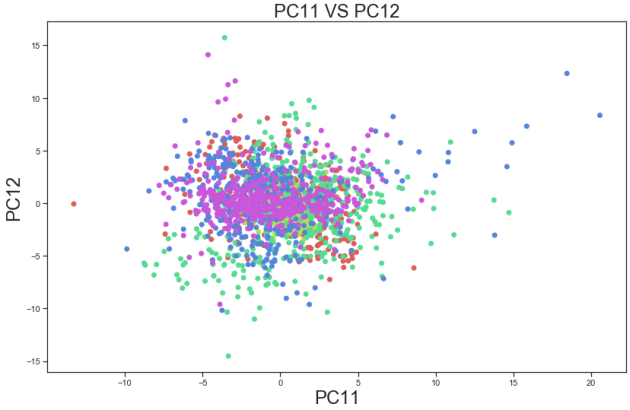

PC9 VS PC10



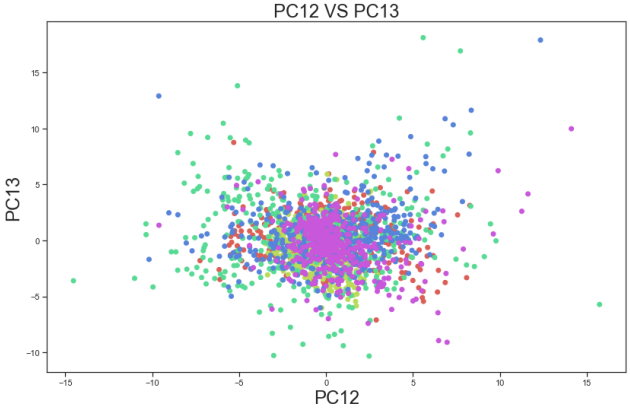

PC10 VS PC11

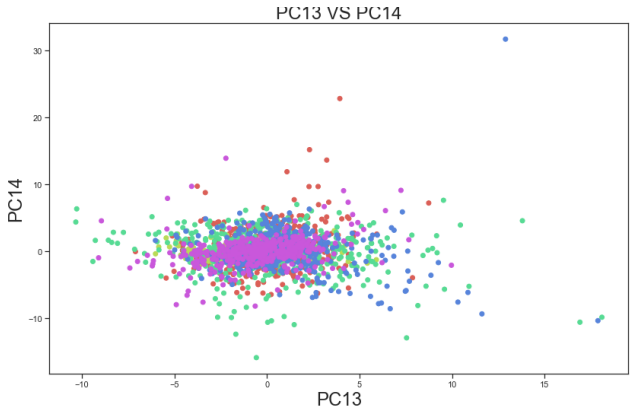


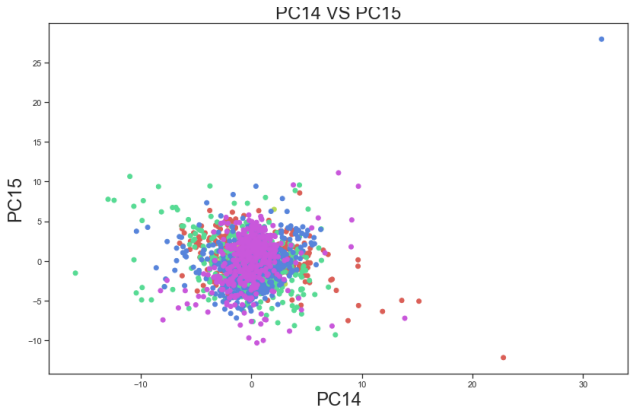

PC11 VS PC12

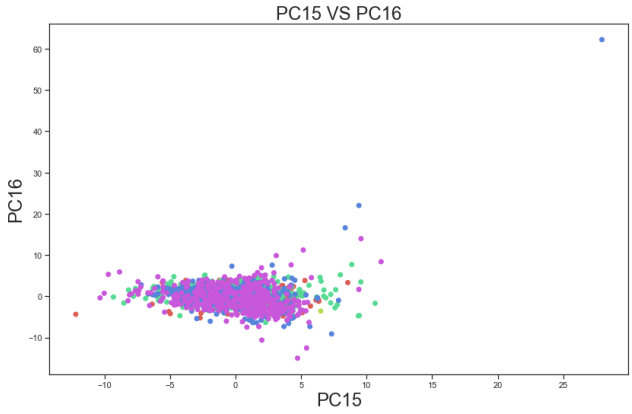


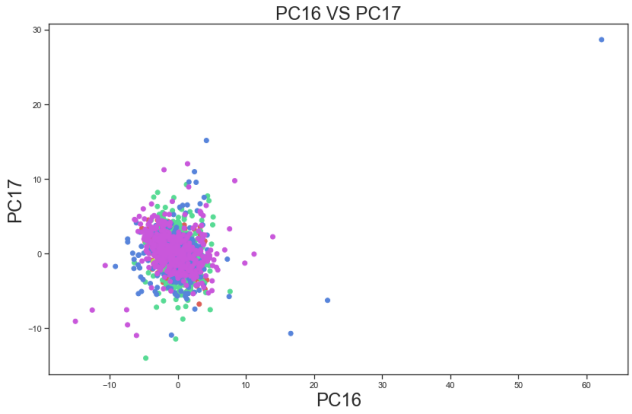

PC12 VS PC13

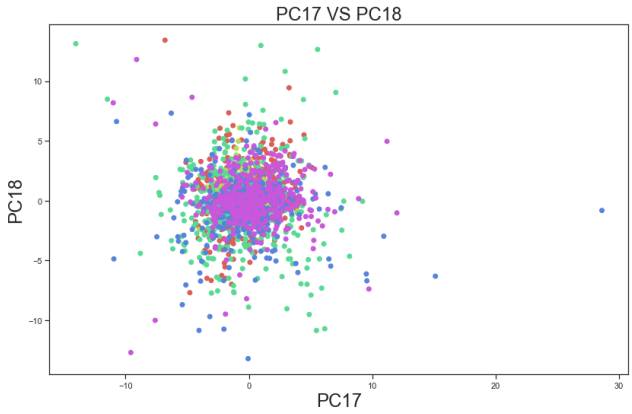


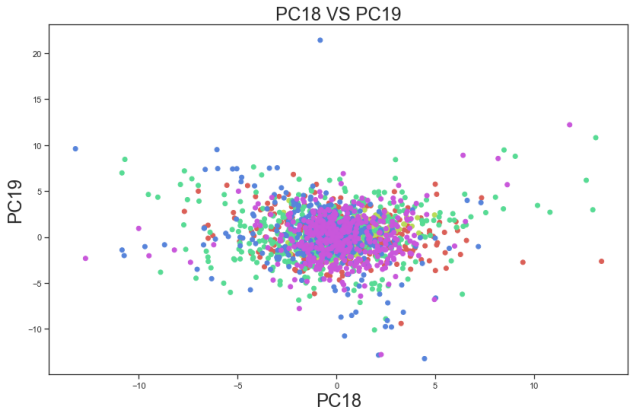


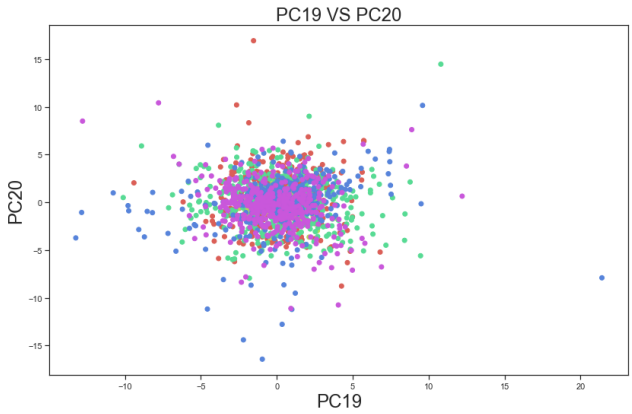




In [121]:

```
tot = sum(eigen_values)
var_explained = [(i / tot)*100 for i in sorted(eigen_values, reverse=True)]
print("Explained Variance for top 20 eigen vectors {0}".format(sum(var_explained[:20])))

pca_model = PCA(.90).fit(X_std)
print("In order to get a expained variance ratio of 90 we need to select {0} PC components".format
(len(pca_model.explained_variance_ratio_)))
```

```
Explained Variance for top 20 eigen vectors 36.300406688477395
In order to get a expained variance ratio of 90 we need to select 297 PC components
```

***Is there a clear point where you could cut off the dimensions? Compare your analysis with the analysis from previous section.***

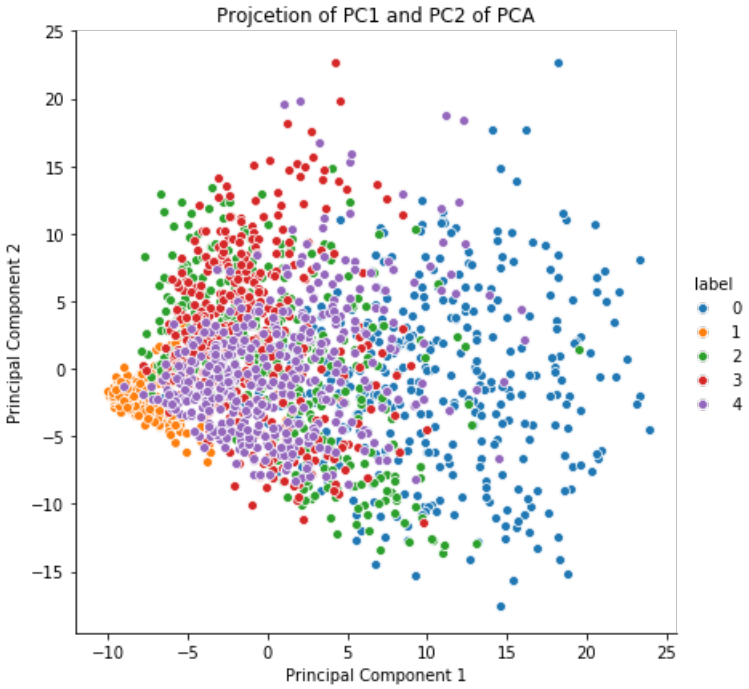
- A clear cut off cannot be indetified using the scatter plots.
- We can see that the ecplained variance ratio when top 20 eigen values are choosen is 36.300406688477395,which is not good enough for good classification of the data.
- As shown above selection of 65 top eigen values we ten to achieve a explained variance of 56.01394993613066.
- In order to achive a explained variance ration of 90 we need to choose about 297 PC

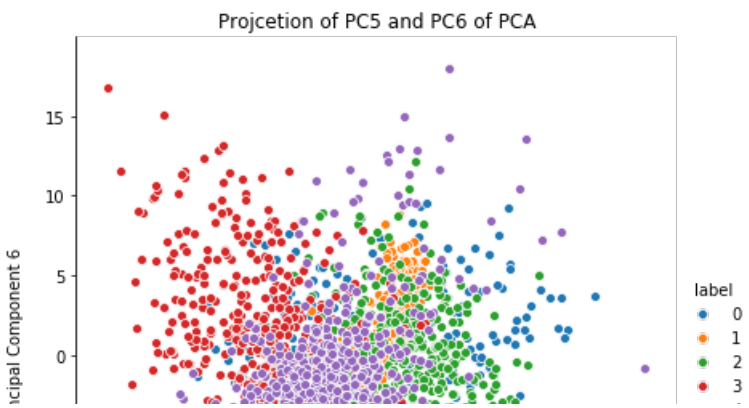
**3. Plot two 2-dimensional representations of the data points based on the first vs second principal components and 5th vs 6th displaying the data points of each class with a different color (you will need to project the data). Explain the results versus the known classes and compare between the two plots.**

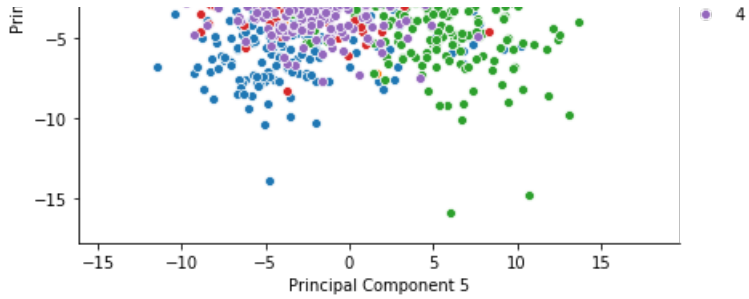
In [13]:

```
x=0
j=1
# X_train_lda = pd.DataFrame(data = X_train_lda, columns = ['Component 1', 'Component 2'])
fig = plt.figure(figsize=(30, 10))
for i in range(1,20):
    pca_data = pd.DataFrame(data = pca[:,[x,x+1]], columns = ['Principal Component '+str(x+1), 'Pri
ncipal Component '+str(x+2)])
    pca_data['label'] = Y
    if x==0 or x==4 :
        sns.pairplot(pca_data,hue='label',x_vars=['Principal Component
'+str(x+1)],y_vars=['Principal Component '+str(x+2)],height=6).set(title ="Projcetion of PC"+str(x+
1)+" and PC"+str(x+2)+" of PCA")
    x= x +1;
```

```
<Figure size 2160x720 with 0 Axes>
```





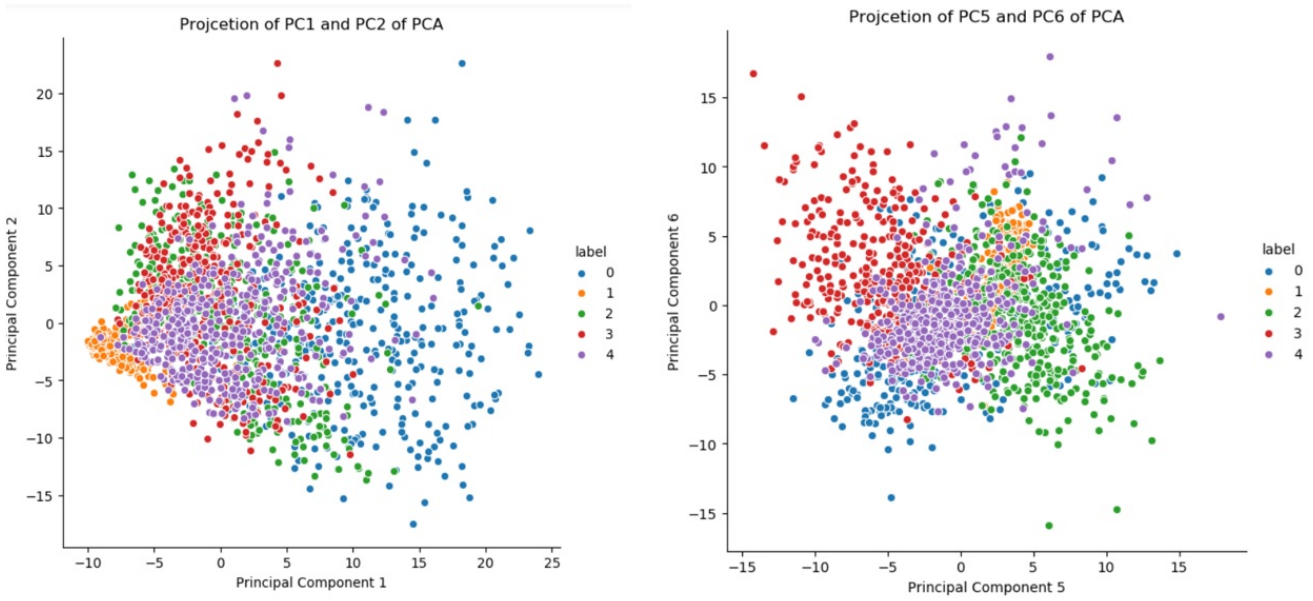

***Understanding Plot PC1 VS PC2***

- From the plot we can observe that class1 has been clustered together to the left center of the plot.
- Class0 is highly dispersed across the right most part the graph ,along with the noise datapoints of other classes.
- Class2,Class3,Class4 are sparesly clustered together with no specific structures ,one lying under the other along with the noise datapoints spreading all over the graph.

***Understanding Plot PC5 VS PC6***

- From this plot we can notice that Class1 is densely clustered but has been overlapped by other class data points.
- Class0 is very sparsed accros the graph with no specific cluster.

***COmparision of Plots (PC1 vs PC2) as "Plot1" and (PC5 vs PC6) as "Plot2"***




- From the above comparison graph we can see that in Plot1 PCA does well to cluster Class1 from other clusters than that of Plot2 where the cluster of Class1 is overlapped by all other clusters.
- In Plot2 Class2 cluster is nostly concentrated to the right lower corner of the graph which is better than the Plot1 Class1 cluster ,which is spread all over the graph.
- For Class3 both the plots tend to perform the same.
- Class0 cluster of Plot1 is very sparesly cluster, where it is bit densily clustered in Plot but which is overlapped by other clusters.
- Cluster of Class4 in Plot 2 is bit densily clustered but is overlapped by almost all the clusters of other classes ,where as the overlapping is less in Plot1.

**4. Implement (1) PCA and (2) dual PCA with singular value decomposition.Save the time of computations and compare the times. Analyze your comparison.**

***PCA with SVD left matrix :***

In [14]:

```
# X_std = StandardScaler().fit_transform(X)
#caluclating covariance matrix using numpy cov function
X_new = X

start_time1= time.time()
meanPoint =X_new.mean(axis = 0)
```

```
# print(meanPoint)

# subtract mean point
X_new -= meanPoint
# n, m = X_std.shape
# assert np.allclose(X_std.mean(axis=0), np.zeros(m))

#Using left SVD matrix which is X.T * X in our case
# cov_matrix = np.cov(X_new.T)
cov_matrix= np.dot(X_new.T, X_new)
# cov_matrix = np.dot(X_std.T,X_std) / (n-1)
# print(cov_mat)
# print(cov_mat.shape)

#decomposing the covariance matrix into eigen values and eigen vectore using numpy eig method
eigen_values, eigen_vectors = np.linalg.eig(cov_matrix)

pca_svd = np.dot(X_new,eigen_vectors)

print("time taken for PCA {0} sec".format(time.time() - start_time1))
print(pca_svd.shape)
```

```
time taken for PCA 2.098554849624634 sec
(2066, 784)
```

***dual PCA with SVD right matrix***

In [15]:

```
start_time1= time.time()
meanPoint =X_new.mean(axis = 0)
X_new -= meanPoint
cov_mat= np.dot(X_new, X_new.T)
eigen_values, V = np.linalg.eig(cov_mat)
V_sqare_root = np.sqrt(eigen_values)
sigma = np.zeros((784,2066))
np.fill_diagonal(sigma,V_sqare_root)
dual_pca = V.T.dot(sigma.T)
print("time taken for dual PCA {0} sec".format(time.time() - start_time1))
print(dual_pca.shape)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\index_tricks.py:861: ComplexWarning: Casting
complex values to real discards the imaginary part
  a.flat[:end:step] = val
```

```
time taken for dual PCA 9.238569259643555 sec
(2066, 784)
```

***Save the time of computations and compare the times. Analyze your comparison.***

- The time taken by PCA is 0.596637487411499 sec
- The time taken by dual PCA is 5.307797193527222 sec

***The time taken by dual PCA is almost times that of PCA.***

Analysis:

- For PCA with SVD the complexity of the algotithm is mostly due to the decomposition of eigenvalues and vectors which can be roughly O(d^3)(big O of d cube).
- For dual LDA with SVD ,along with eigen decomposition generating sigma matrix has a complexity of O(dn^2) (big O of d*(n sqaure)).
- So for this data set number of samples n is thrice the number of dimensions, the time taken by dual PCA tends to be more than that of PCA
- Hence dual PCA can be used when the number of features are very greater than samples.

## 2.2.2 Theoretical Question

**Prove that PCA is the best linear method for reconstruction (with orthonormal bases). Hint: write down the optimization**

problem and solve it.

(8) Prove that PCA is the best linear method for reconstruction.

Assumptions ⇒

Sample size = $n$

$x_i \in R^d$ where $\{x_i\}_{i=1}^n$ is the input data

$y_i \in R^1$ where $\{y_i\}_{i=1}^n$ are the observations.

$\Rightarrow R^{d \times n} \ni X := [x_1, \ldots, x_n]$

$\Rightarrow R^{1 \times n} \ni Y := [y_1, \ldots, y_n]$

→ Assume we have a datapoint $x \in R^d$ and we want to project this data point onto a vector subspace spanned by $p$ vectors $\{u_1, \ldots, u_p\}$ where each vector is $d$-dimensional and $p \ll d$.

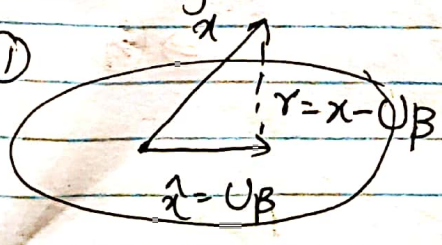
→ We stack these vectors columnwise in Matrix $U$

$$U = [u_1, \ldots, u_p] \in R^{d \times p}.$$

→ In other words we want to project $x$ onto the column space of $U$.

→ The projection of $x \in R^d$ onto $\text{Col}(U) \in R^p$ and then its representation in the $R^d$ (its reconstruction) can be seen as a linear system of equations.

$$R^d \ni \hat{x} : U\beta \quad \text{—} \quad (1)$$




→ In case if subspace and original space are not equal then we have a residual

$$r = x - \hat{x} = x - U\beta. \Rightarrow \text{which should be small}$$

↳ (2)

→ The smallest residual vector is orthogonal to $U$ (5)

$\therefore \quad x - U\beta \perp U \Rightarrow U^T(x - U\beta) = 0$

$\Rightarrow \beta = (U^TU)^{-1}U^Tx \longrightarrow (3)$

(3) in (1) gives.

$$\hat{x} = U(U^TU)^{-1}U^Tx \quad \text{—— (4)}$$

assume $R^{d \times d} \ni \Pi := U(U^TU)^{-1}U^T$ —— (5)

↓ hat matrix. as projection matrix

If vectors $\{u_1, \ldots, u_p\}$ are orthonormal (matrix $U$ is orthogonal) the $U^T = U^{-1}$ and thus $U^TU = I$

$\Pi = UU^T$

$\Rightarrow$ (4) becomes

$$\hat{x} = \Pi x = UU^Tx \longrightarrow (6)$$

If there exists $n$ training data points ie $\{x_i\}_{i=1}^n$, the projection of a training data point $x$ is:

$$R^p \ni \tilde{x} := U^T\breve{x} \longrightarrow (7)$$

where $R^d \ni \breve{x} = x - \mu_x$ i.e Centered data point $\longrightarrow (8)$

$R^d \ni \mu_x : \frac{1}{n}\sum_{i=1}^{n} x_i$ $\longrightarrow (9)$ mean of traing data points

→ The reconstruction of a training data point $x$

after projection onto the PCA sub-space is:

$$R^d \Rightarrow \hat{x} := UU^T\tilde{x} + \mu_x = U\tilde{x} + \mu_x \quad \text{— (10)}$$

## Minimising Reconstruction Error :-

if we centre data

$$r = \tilde{x} - \hat{x}$$

then from eq (2), (8) and (10) we have

$$r = x - \hat{x} = \tilde{x} + \mu_x - UU^T\tilde{x} - \mu_x$$

$$r = \tilde{x} - UU^T\tilde{x}. \quad \text{— (11)}$$

for n data points

$$R: X - \hat{X} = \tilde{X} + \mu_x - UU^T\tilde{x} - \mu_x$$

$$R := \tilde{X} - UU^T\tilde{X} \quad \text{— (12)}$$

$R^{d \times n} \Rightarrow R = [r_1, \ldots r_n]$ matrix of residual

∴ we have to minimize R

$$\text{minimize } \|\tilde{X} - UU^T\tilde{X}\|_F^2$$

$$\text{subject to } U^TU = I \text{ (Identity matrix)}$$

The objective function can be simplified as

$$\|\tilde{X} - UU^T\tilde{X}\|^2$$

$\|\hat{X} - \hat{X}\|_F$

$$= tr((\hat{X} - UU^T\hat{X})^T(\hat{X} - UU^T\hat{X}))$$

$$= tr((\hat{X}^T - \hat{X}U^TU^T)(\hat{X} - UU^T\hat{X}))$$

$$= tr(\hat{X}^T\hat{X} - 2\hat{X}^TUU^T\hat{X} + \hat{X}^TUU^TUU^T\hat{X})$$

$U^TU = I$

$\Rightarrow UU^TUU^T$

$UIU^T = UU^T$

$$= tr(\hat{X}^T\hat{X} - 2\hat{X}^TUU^T\hat{X} + \hat{X}^TUU^T\hat{X})$$

$$= tr(\hat{X}^T\hat{X} - \hat{X}^TUU^T\hat{X})$$

$$= tr(\hat{X}^T\hat{X}) - tr(\hat{X}^TUU^T\hat{X})$$

→ By properties of trace.

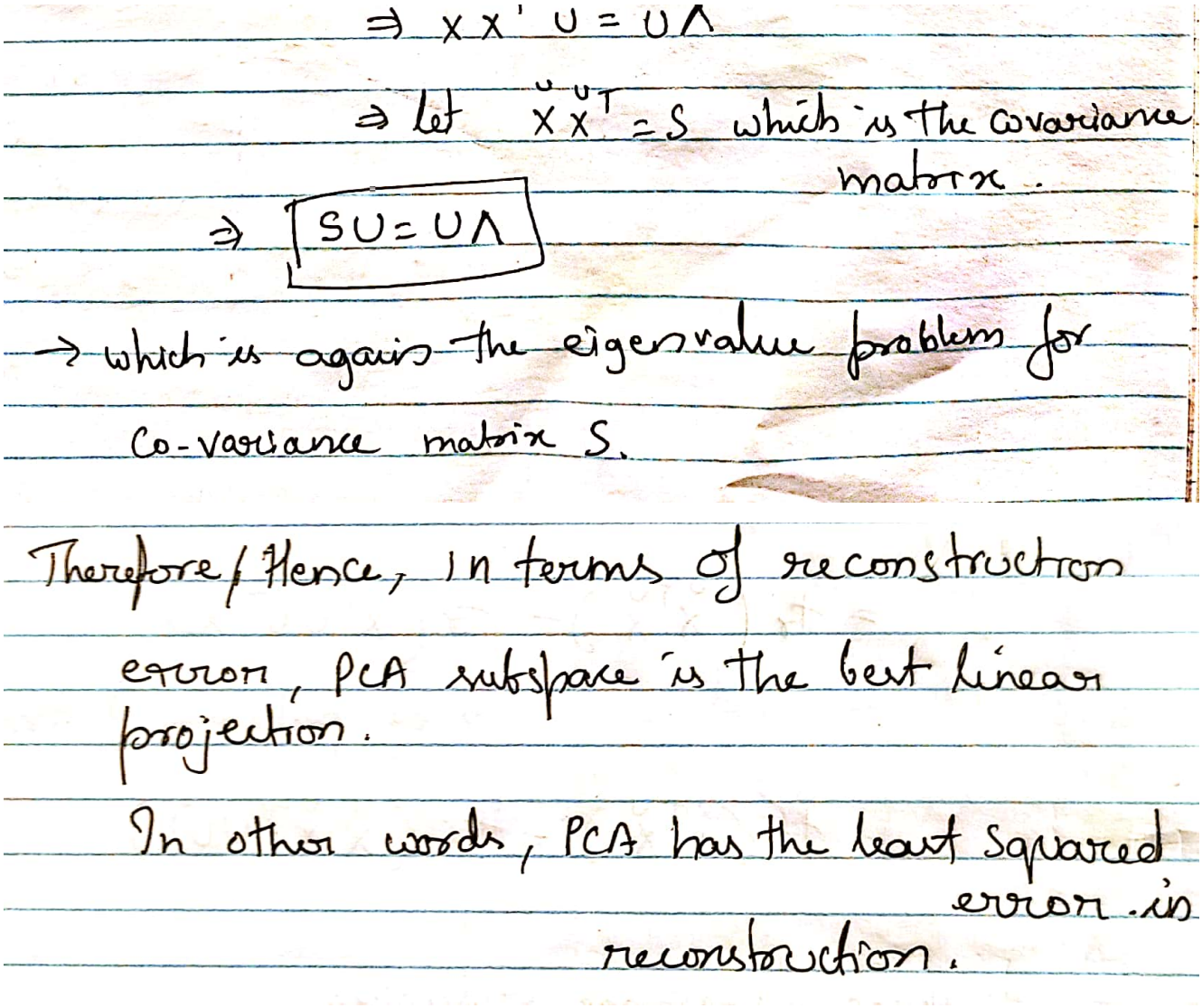
$$= tr(\hat{X}^T\hat{X}) - tr(\hat{X}\hat{X}^TUU^T)$$

→ Using lagrange multipliers (13)

$$\mathcal{L} = tr(\hat{X}^T\hat{X}) - tr(\hat{X}\hat{X}^TUU^T) - tr(\Lambda^T(U^TU - I))$$

where $\Lambda \in R^{P \times P}$ is a diagonal matrix $diag([\lambda_1, \ldots, \lambda_p]^T)$ containing Lagrange multipliers. To minimize equation (13) we can derivate it and equal it to zero

$$R^{d \times p} \ni \frac{\partial \mathcal{L}}{\partial U} = 2\hat{X}\hat{X}^TU - 2U\Lambda \overset{set}{=} 0$$




## 2.3 Fisher Discriminant Analysis (FDA)

### 2.3.1 Practical Questions

**1. As the class labels are already known, you can use the FDA or LDA to reduce the dimensionality. Using any implementation of FDA or LDA you wish, and subplot in python matplotlib, plot the scatter plot of the projected data with the top 20 eigenvalues (use colors and legends for classes).**

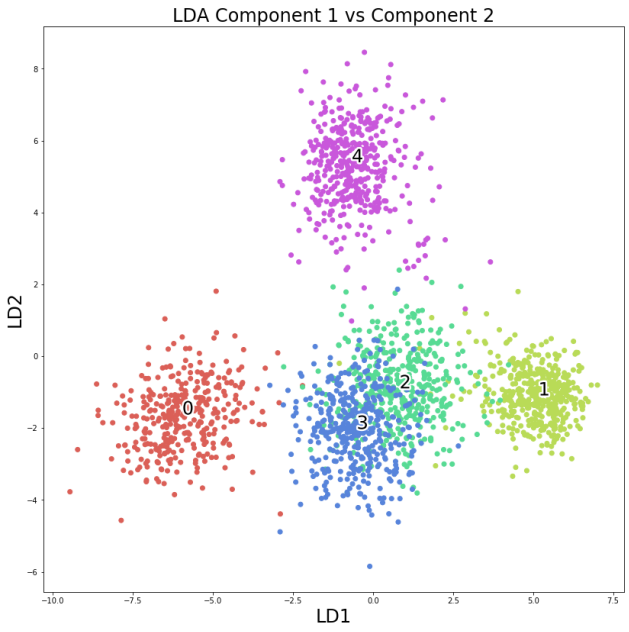
In [17]:

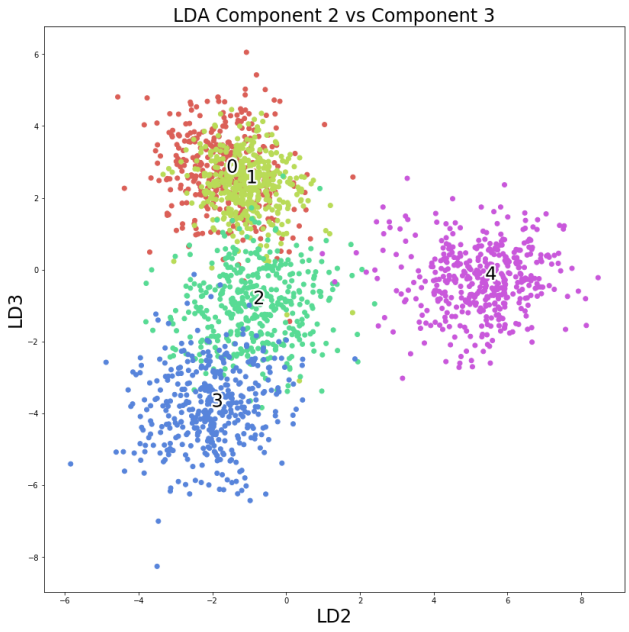
```
def scatter_plot(x, colors,method,ax,xlab,ylab):
    # choose a color palette with seaborn.
    num_classes = len(np.unique(colors))
    palette = np.array(sns.color_palette("hls", num_classes))

    # create a scatter plot.
    sc = ax.scatter(x[:,0], x[:,1], c=palette[colors.astype(np.int)])
    ax.set_title(method,fontsize=24)
    plt.xlabel(xlab,fontsize=24)
    plt.ylabel(ylab,fontsize=24)
    # add the labels for each digit corresponding to the label

    for i in range(num_classes):

        # Position of each label at median of data points.

        xtext, ytext = np.median(x[colors == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
```

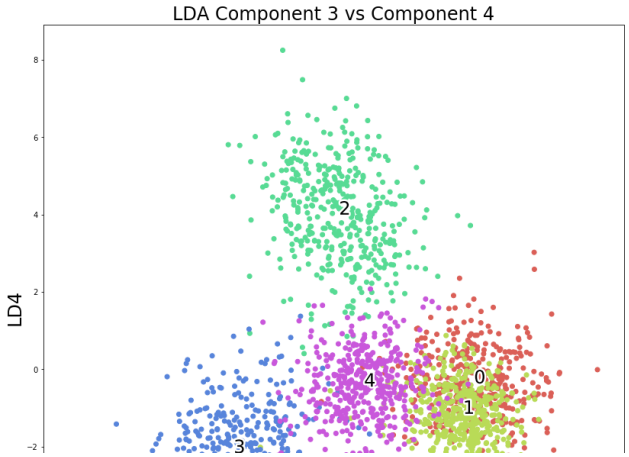
In [18]:

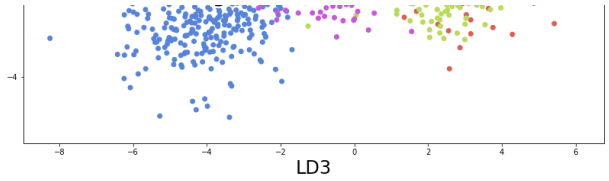
```
X = dataset.iloc[:,1:785]
Y = dataset.iloc[:,-1]

scaler = StandardScaler().fit(X)
X_train_norm = scaler.transform(X)

#fitting LinearDiscriminantAnalysis model only for train set
lda = LinearDiscriminantAnalysis(n_components=None)
lda.fit(X_train_norm,Y)

#transforming both train and test data using fitted model
X_train_lda = lda.transform(X_train_norm)

print(X_train_lda.shape)



# f = plt.figure(figsize=(8, 8))
fig = plt.figure(figsize=(30, 30))
for i in range(0,3):
    ax = plt.subplot(2,2,i+1)
    scatter_plot(X_train_lda[:,[i,i+1]],Y,"LDA Component "+str(i+1)+" vs Component "+str(i+2),ax,"L
D"+str(i+1),"LD"+str(i+2))
# X_train_lda = pd.DataFrame(data = X_train_lda, columns = ['Component 1', 'Component
2','Component 3', 'Component 4'])
# X_train_lda['label']=Y
# sns.set_palette("dark")
# sns.pairplot(X_train_lda,hue='label',height=7).set(title ="Projcetion of first two components of
LDA for label 'quality'")
```

```
(2066, 4)
```






***Explain the results obtained in terms of the known classes. Which LDA directions separate which classes better (which LDA directions are responsible for separating which classes)?***

Let Comp1 vs Comp2 = Plot1

Let Comp2 vs Comp3 = Plot2

Let Comp3 vs Comp4 = Plot3

- From Plot1 we can observe that LD1 direction can separates classes '0' and '1' quite well ,but fails to add any valuble information about classes '2','3' and '4'.
- From Plot1 and Plot2 we can observe that LD2 separates class '4' very well ,but does very bad for the remaining classes.
- From Plot2 and Plot3 we can observe that LD3 separates class '3' but with some noise datapoints for other classes.
- From Plot3 we can notice that class '2' can be separated using LD4.

Directions and thier responsibilty to separate classes:

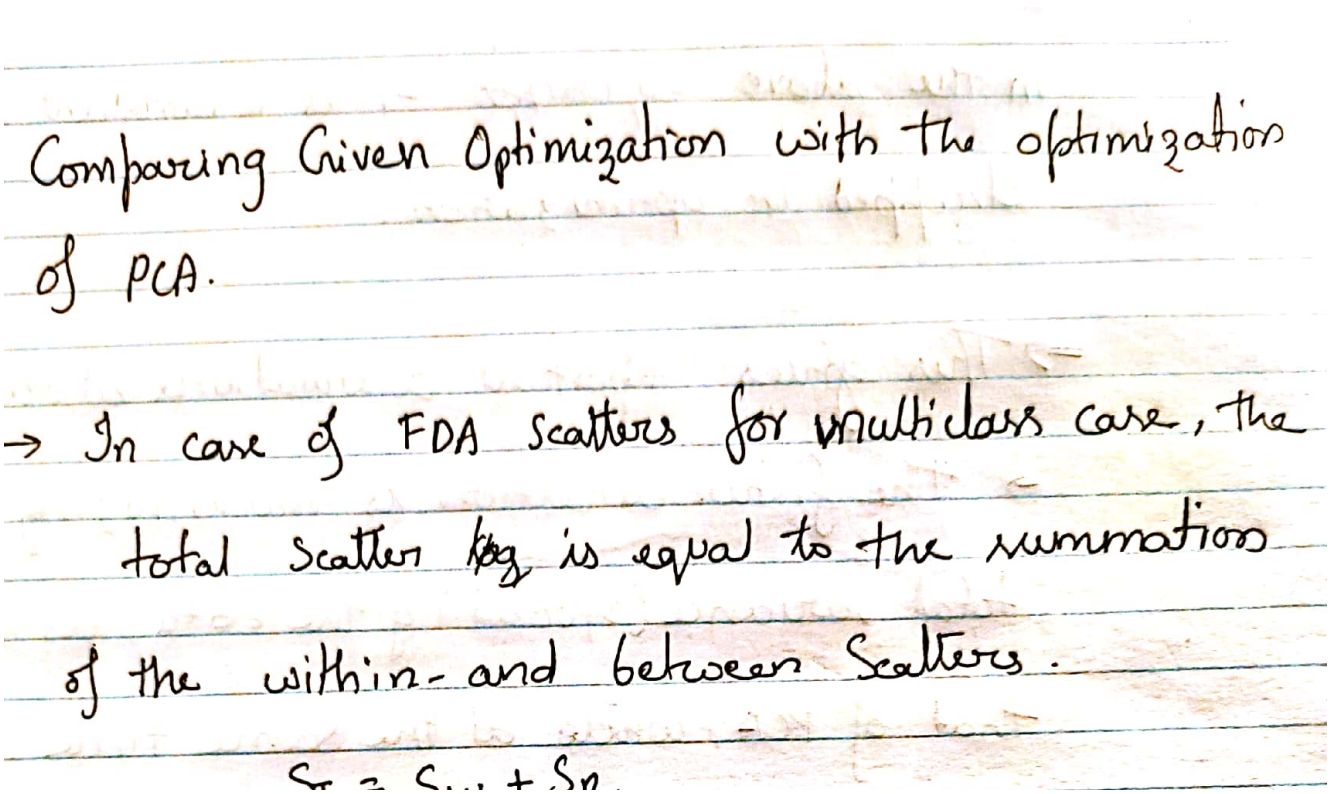
- Direction 1(LD1) -> class '0' and class '1'
- Direction 2(LD2) -> class '4'
- Direction 3(LD3) -> class '3'
- Direction 4(LD4) -> class '2'

**2. Compare the results of the LDA with the results obtained by using PCA.**

- We can observe that from the graph clustering can be visualised well with the plots of LDA rather than that of PCA
- Both LDA and PCA are linear transformation techniques: LDA is a supervised whereas PCA is unsupervised – PCA ignores class labels.So LDA identifies the clusters better by using the class labels.
- PCA is a technique that finds the directions of maximal variance,where as in contrast,LDA attempts to find a feature subspace that maximizes class separability.Hence the class separability in LDA plots has good boundaries.

### 2.3.2 Theoretical Question

**Compare this with the optimization in PCA. Explain and analyze your comparison. After this analysis, compare your theoretical comparison (this question) and the practical comparison (question 2 in practical questions).**



$S_T = S_W + S_B$.

$\therefore$ Between Scatter $= S_B = S_T - S_W$

When this $S_B$ is put into fisher criterion :-

$$f(u) = \frac{u^T S_B u}{u^T S_W u} = \frac{u^T (S_T - S_W) u}{u^T S_W u}$$

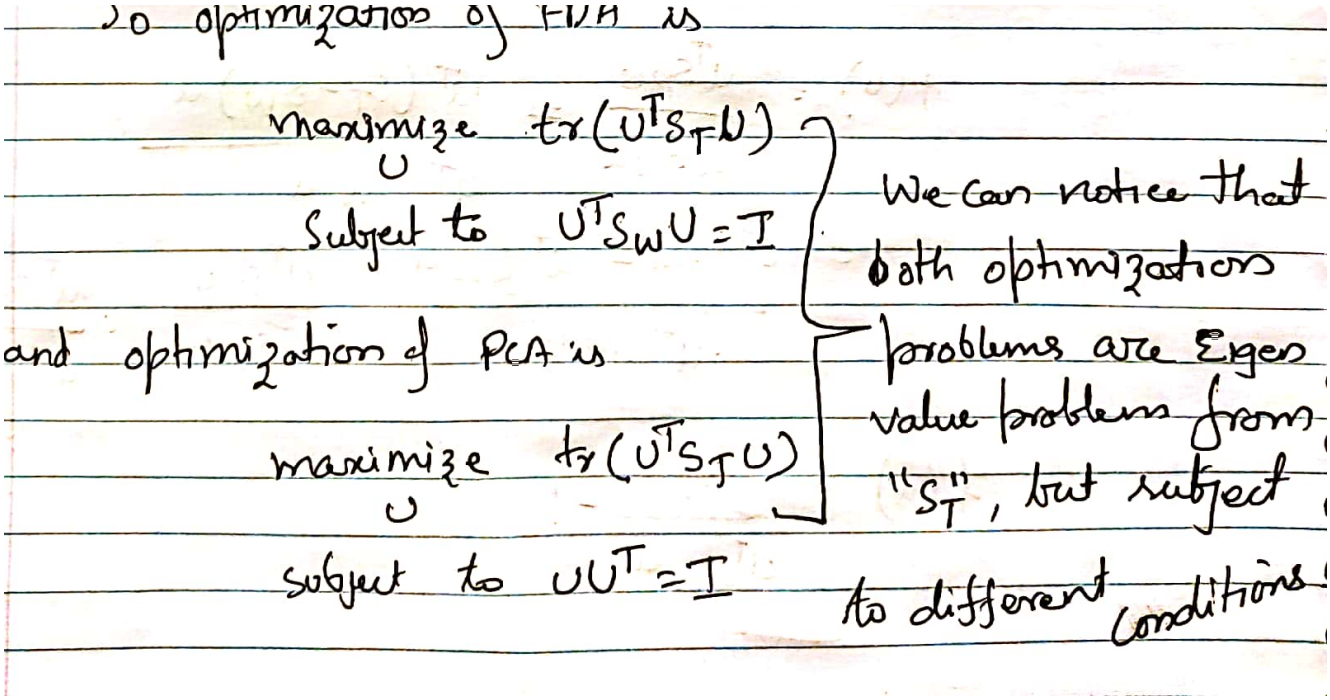
$$= \frac{u^T S_T u - u^T S_W u}{u^T S_W u} = \frac{u^T S_T u}{u^T S_W u} - \frac{u^T S_W u}{u^T S_W u}.$$

$$f(u) = \frac{u^T S_T u}{u^T S_W u} - 1$$

in the above equation $-1$ is a constant and is dropped in optimization.

$\Rightarrow$ This gives important observations about FDA.

→ The fisher direction is maximizing the total variance (spread) of the data, as same as that of PCA, while at the same time, Fisher direction minizies the within-scatters of classes.




- From the above analysis we can note that,Fisher directions are maximizing the total variance(spread) of the data just like the PCA.So we can observe that both PCA and LDA has almmost similar area of scatter.
- At the same time Fisher directions tend to minimize the sactters within the classes.So we can notice that the clusters of FDA are more denser than that of PCA along with good separation between clusters.As this tries to minimize teh within scatters , the boundaries between clusters are more visible.
- Each Fisher directions tries to minimise the scatter of some class along its direction.Hence we can observe from the scatter plots FDA, that each direction clusters one class along its direction.

References :


- Building PCA model : https://www.jeremyjordan.me/principal-components-analysis/
- PCA Scree Plot descision : https://en.wikipedia.org/wiki/Scree_plot
- LDA Analysis : https://sebastianraschka.com/Articles/2014_python_lda.html
- Subplots : https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html
- Scatter Plot : https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html


# Question 3:

In [19]:

```
# libraries
import numpy as np
import pandas as pd
import seaborn as sns
import time
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import matplotlib.patheffects as PathEffects
from sklearn.decomposition import PCA
import scipy.linalg as la

from sklearn.manifold import Isomap
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.manifold import SpectralEmbedding
from sklearn.manifold import TSNE
from sklearn.decomposition import KernelPCA

sns.set_context("notebook", font_scale=1.5,
                rc={"lines.linewidth": 2.5})
```

In [20]:

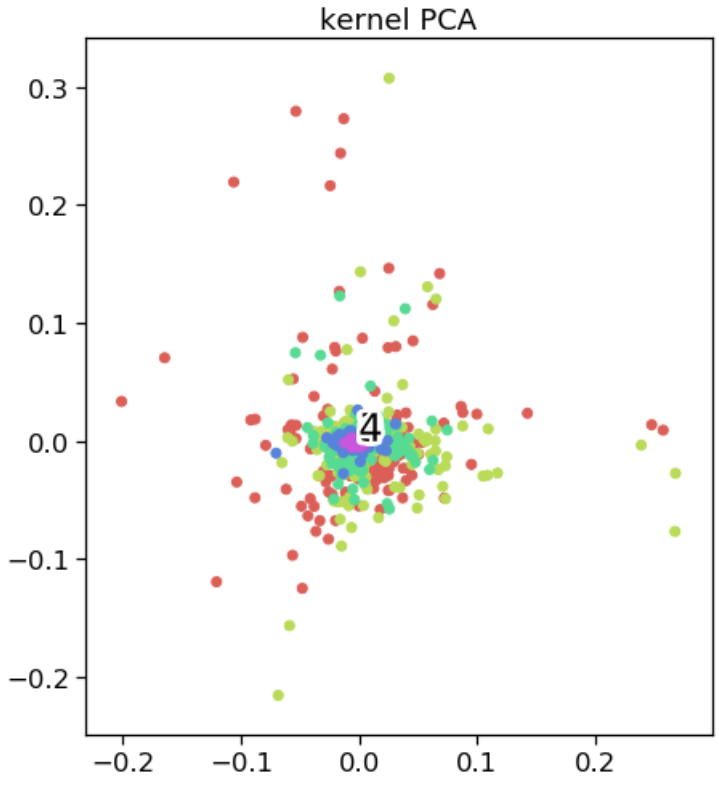
```
#loading dataset
```

```
dataset = pd.read_csv("DataB.csv")

#dividing data set into labels and target
X_train = dataset.iloc[:,1:-1]
Y_train = dataset.iloc[:,-1]
# X_train.shape
# print(X_train)
# Y_train.shape
# print(Y_train)
# Y_train.shape
taken_t= []

#function to plot the scatter plot of the models used
def scatter_plot(x, colors,method):
    # choose a color palette with seaborn.
    num_classes = len(np.unique(colors))
    palette = np.array(sns.color_palette("hls", num_classes))

    # create a scatter plot.
    f = plt.figure(figsize=(8, 8))
    ax = plt.subplot(aspect='equal')
    sc = ax.scatter(x[:,0], x[:,1], c=palette[colors.astype(np.int)])
    ax.set_title(method)
    # add the labels for each digit corresponding to the label

    for i in range(num_classes):

        # Position of each label at median of data points.

        xtext, ytext = np.median(x[colors == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
```

**kernel PCA**

In [21]:

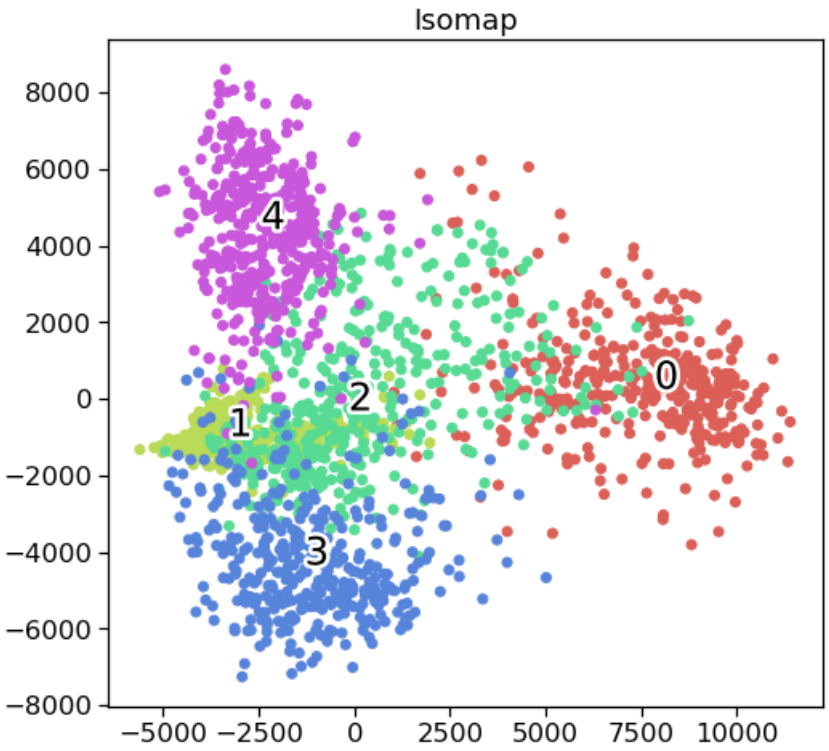
```
time_start = time.time()
kernel_Data = KernelPCA(n_components=2,kernel="rbf",random_state=42).fit_transform(X_train)
taken_t.append(["Kernel PCA",time.time()-time_start])
scatter_plot(kernel_Data, Y_train,"kernel PCA")
```



**Isomap**

In [22]:

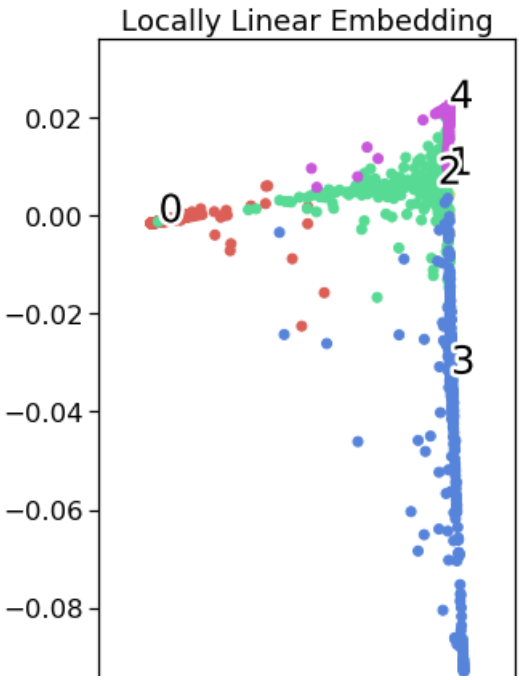
```
time_start = time.time()
iso_data = Isomap(n_components=2).fit_transform(X_train)
taken_t.append(["Isomap",time.time()-time_start])
scatter_plot(iso_data, Y_train,"Isomap")
```

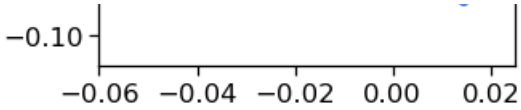



**Locally Linear Embedding (LLE)**

In [23]:

```
time_start = time.time()
lle_Data = LocallyLinearEmbedding(n_components=2,random_state=42).fit_transform(X_train)
taken_t.append(["LLE",time.time()-time_start])
scatter_plot(lle_Data, Y_train,"Locally Linear Embedding")
```

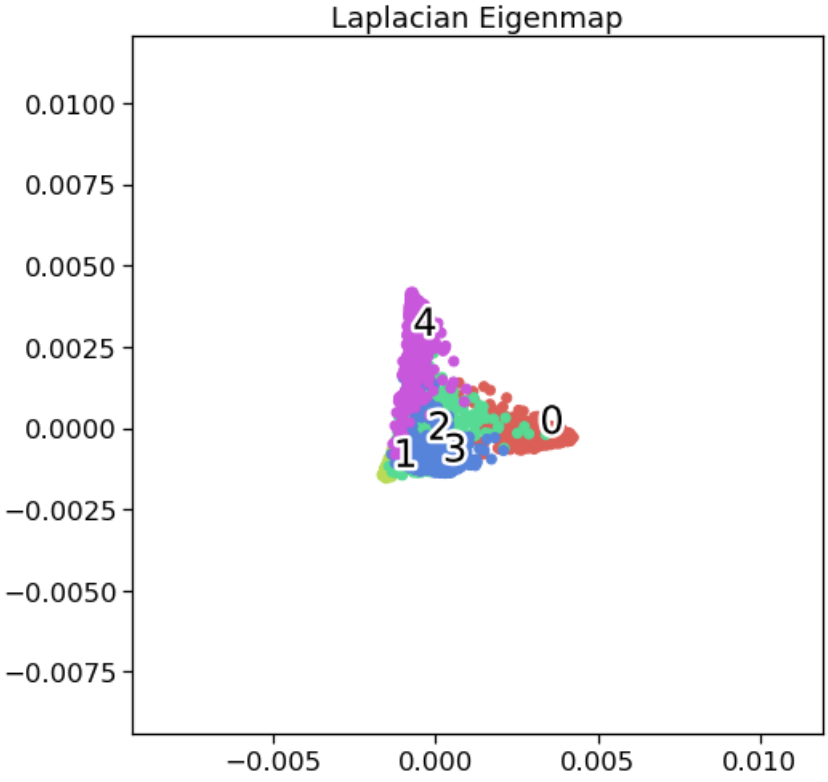





**Laplacian Eigenmap (sklearn.manifold.SpectralEmbedding)**

In [24]:

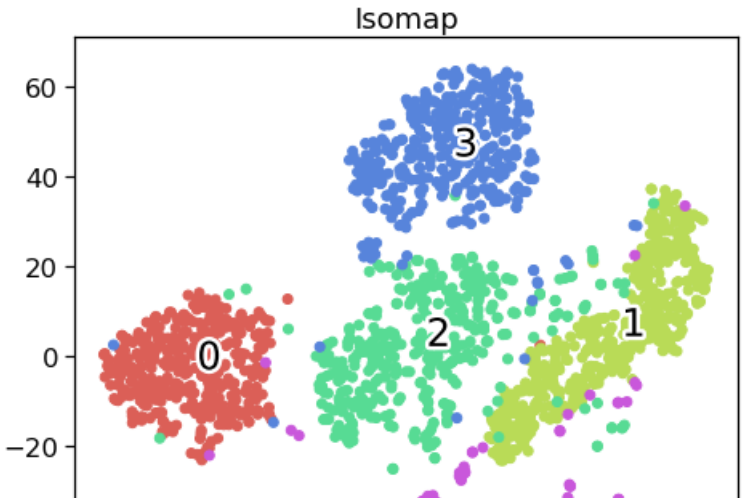
```
time_start = time.time()
se_Data = SpectralEmbedding(n_components=2,random_state=42).fit_transform(X_train)
taken_t.append(["Laplacian Eigenmap",time.time()-time_start])
scatter_plot(se_Data, Y_train,"Laplacian Eigenmap")
```

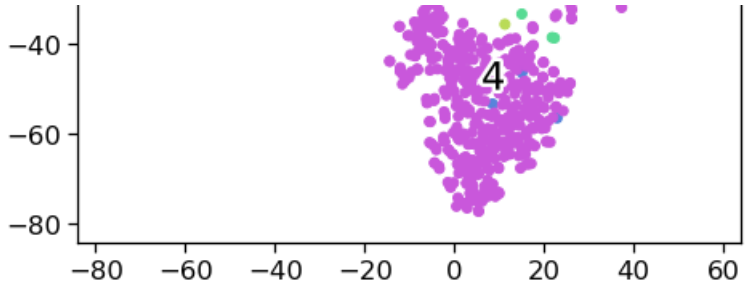



**t-SNE**

In [25]:

```
time_start = time.time()
sne_data = TSNE(n_components=2,random_state=42).fit_transform(X_train)
taken_t.append(["t-SNE",time.time()-time_start])
scatter_plot(sne_data, Y_train,"Isomap")
```






In [26]:

```
#different method for graph
# tsne_results = TSNE(n_components=2,random_state=42).fit_transform(X_train)
# df_tsne = pd.DataFrame(tsne_results, columns=['comp1', 'comp2'])
# df_tsne['label'] = Y_train
# sns.lmplot(x='comp1', y='comp2', data=df_tsne, hue='label', fit_reg=False)

#printing time taken by different models
for k in taken_t:
    print("Time taken by {0} is {1} sec".format(k[0],k[1]))
```

```
Time taken by Kernel PCA is 2.813565969467163 sec
Time taken by Isomap is 20.885336875915527 sec
Time taken by LLE is 16.572259187698364 sec
Time taken by Laplacian Eigenmap is 18.03773856163025 sec
Time taken by t-SNE is 48.875895977020264 sec
```

***Which methods do better on which parts of the data?***

- kernel PCA -> Kernal PCA doesnt tend to perform well for this dataset.All clusters are emmbedded at the same location ,with lot of noise data surrounding them.The need for visualising data in clusters in 2D plane cannot be sufficed by this method.
- Isomap ->This method classified the clusters into 5 groups as expected and seems to perform better for clusters of 3 and 4,but the other clusters doesnt tend to have good boundaries and this methods shows the effectively the nosie data points present.
- Locally Linear Embedding (LLE) ->This method tends to preform better on clusters 0,3 and 4 where as it barely disingushed cluster 1 from 2.
- Laplacian Eigenmap (sklearn.manifold.SpectralEmbedding) -> Laplacian Eigenmap tends to classify the data into 5 clusters with high density ,but no specific boundaries between them.
- t-SNE ->t-SNE performed well in clustering the data into 5 clusters with good boundaries along with some noise data points.

***Give at least three clear performance differences between a pair of methods that you can explain based on the nature of methods and the data.***

We can consider methods Isomap and t-SNE for comparision and stating the performance differences:

- Computaional performance of Isomap is far better than that off the t-SNE.
- As caluclated above ,the time taken by t-SNE for fitting the model is "48.4825 sec" where as Isomap has taken only "19.7505 sec".So Isomap is as twice fast as t-SNE.
- The ability to reatin local structure of data: t-SNE model is based on Local approaches,which tends to retain local geometric structure in the lower dimensional space,in contrast Isomap is based on global approaches.The choice has to be made considering whether one has to retain the local sturcture or the global one.
- Clustering representation:When considered the scatter plot of t-SNE ,we can observe that the clusters are more clear with good boundaries,where as in Isomap ,the clusters are overlapped with no perfect boundaries.In case if clear clustering is needed one should choose t-SNE for this dataset.

***General Observations with Isomap and t-SNE:***

- As isomap operates on nearest neighbours approch ,we don’t take enough nearest neighbors, then the clustering of the graph varies.
- Isomap is capable of retaining the local structure of the data while also revealing some important global structure (such as clusters at multiple scales)
- Often different runs with the same hyperparameters will produce different results in t-SNE, so several plots must be observed before any evaluation with t-SNE is carried out.
- Naturally, t-SNE extends large clusters and shrinks those of spares ones.So teh comparision of sizes of clusters in t-SNE graph might mean nothing.
- Distance between the clusters in t-SNE might mean nothing as this approch is based on t-based distribution probablity of nearest data points

data points.

***What tradeoffs might need to be considered in order to decide which method is 'best' to use for this dataset?***

As the given dataset has only 2066 observations and the required result is the dimensionality reduction of data for visualization , we can think of the following tradeoffs:

- Performance/Computational efficiency:
  - The time taken by different reduction methods/algorithms doesn't vary much as the dataset has less observations.So performance of the models can be traded off for the required quality like clustering cleanliness.If the data has around million observations the time taken by different algorithms might have large differences, in such senario we can cosider models which has better computational efficiency.
- Clustering Boundaries:
  - As discussed above,as the computational efficency can be traded off,we can consider the model which clusters the dataset into classes with definite boundaries and which has non-overlapping clusters.

```
Time taken by Kernel PCA is 0.9163684844970703 sec
Time taken by Isomap is 19.60268759727478 sec
Time taken by LLE is 16.24790859222412 sec
Time taken by Laplacian Eigenmap is 17.89239263534546 sec
Time taken by t-SNE is 48.248770236968994 sec
```

We can notice that the time taken by PCA is under 1 sec,but the clustering obtained by using kernal PCA is not useful to represent different clutesrs of the given dataset.We also can notice that Isomap,LLE and Laplacian Eigenmap has taken huge time when compared to KernelPCA(20 times) and half the time as t-SNE(approx).

***We can observe that t-SNE has provied best clusters for this dataset which has good boundaries without getting rid of noise data.So the computational expense of t-SNE can be traded off for its ability to form proper clusters for this dataset.***

References:


- https://distill.pub/2016/misread-tsne/
- https://scikit-learn.org/stable/modules/manifold.html
- http://www.cs.toronto.edu/~hinton/absps/tsne.pdf
- google tech talk on Visualizing Data Using t-SNE -> https://www.youtube.com/watch?v=RJVL80Gg3lA
- A Global Geometric Framework for Nonlinear Dimensionality Reduction->http://web.mit.edu/cocosci/Papers/sci_reprint.pdf
- An Introduction to Locally Linear Embedding -> https://cs.nyu.edu/~roweis/lle/papers/lleintro.pdf


In [ ]: