In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns; sns.set(color_codes=True)
```

In [3]:

```
one = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-Mo
unted Accelerometer/1.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleration"
,"Labels"])
one.drop('Sequence', axis = 1, inplace = True)

two = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-Mo
unted Accelerometer/2.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleration"
,"Labels"])
two.drop('Sequence', axis = 1, inplace = True)

three = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-
Mounted Accelerometer/3.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleratio
n","Labels"])
three.drop('Sequence', axis = 1, inplace = True)

four = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-M
ounted Accelerometer/4.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleration
","Labels"])
four.drop('Sequence', axis = 1, inplace = True)

five = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-M
ounted Accelerometer/5.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleration
","Labels"])
five.drop('Sequence', axis = 1, inplace = True)

six = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-Mo
unted Accelerometer/6.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleration"
,"Labels"])
six.drop('Sequence', axis = 1, inplace = True)

seven = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-
Mounted Accelerometer/7.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleratio
n","Labels"])
seven.drop('Sequence', axis = 1, inplace = True)

eight = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-
Mounted Accelerometer/8.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleratio
n","Labels"])
eight.drop('Sequence', axis = 1, inplace = True)

nine = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-M
ounted Accelerometer/9.csv', names=["Sequence", "x_acceleration", "y_acceleration", "z_acceleration
","Labels"])
nine.drop('Sequence', axis = 1, inplace = True)

ten = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Chest-Mo
unted Accelerometer/10.csv', names=["Sequence", "x_acceleration", "y_acceleration",
"z_acceleration","Labels"])
ten.drop('Sequence', axis = 1, inplace = True)

eleven = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single
Chest-Mounted Accelerometer/11.csv', names=["Sequence", "x_acceleration", "y_acceleration",
"z_acceleration","Labels"])
eleven.drop('Sequence', axis = 1, inplace = True)

twelve = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single
Chest-Mounted Accelerometer/12.csv', names=["Sequence", "x_acceleration", "y_acceleration",
"z_acceleration","Labels"])
twelve.drop('Sequence', axis = 1, inplace = True)

thirteen = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Che
st-Mounted Accelerometer/13.csv', names=["Sequence", "x_acceleration", "y_acceleration",
"z_acceleration","Labels"])
thirteen.drop('Sequence', axis = 1, inplace = True)
```

```
fourteen = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Che
st-Mounted Accelerometer/14.csv', names=["Sequence", "x_acceleration", "y_acceleration",
"z_acceleration","Labels"])
fourteen.drop('Sequence', axis = 1, inplace = True)

fifteen = pd.read_csv('C:/Users/Pujachouhan/OneDrive/Desktop/Activity Recognition from Single Ches
t-Mounted Accelerometer/15.csv', names=["Sequence", "x_acceleration", "y_acceleration",
"z_acceleration","Labels"])
fifteen.drop('Sequence', axis = 1, inplace = True)
```

In [4]:

```
one['Person'] = 1
two['Person'] = 2
three['Person'] = 3
four['Person'] = 4
five['Person'] = 5
six['Person'] = 6
seven['Person'] = 7
eight['Person'] = 8
nine['Person'] = 9
ten['Person'] = 10
eleven['Person'] = 11
twelve['Person'] = 12
thirteen['Person'] = 13
fourteen['Person'] = 14
fifteen['Person'] = 15
```

In [5]:

```
frames = [one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve
,thirteen,fourteen , fifteen]

df= pd.concat(frames)

df.shape
```

Out[5]:

```
(1926896, 5)
```

In [6]:

```
df_rnd = df.sample(n=100000, random_state=6758)
```

Data Preparation

In [7]:

```
df_rnd.dtypes
```

Out[7]:

```
x_acceleration    int64
y_acceleration    int64
z_acceleration    int64
Labels            int64
Person            int64
dtype: object
```

In [8]:

```
df_rnd.isna().sum()
```

Out[8]:

```
x_acceleration    0
y_acceleration    0
z_acceleration    0
Labels            0
```

```
Labels          0
Person          0
dtype: int64
```

In [9]:

```
df_rnd = df_rnd[df_rnd.Labels != 0] #Removing rows which had label zero
```

In [10]:

```
df_rnd['Labels'].value_counts()
```

Out[10]:

```
1    31627
7    30770
4    18590
3    11178
5     2785
2     2498
6     2371
Name: Labels, dtype: int64
```

Data Exploration

In [11]:

```
df_rnd.describe()
```

Out[11]:

	x_acceleration	y_acceleration	z_acceleration	Labels	Person
count	99819.000000	99819.000000	99819.000000	99819.000000	99819.000000
mean	1987.307336	2381.967471	1970.829041	3.887617	7.518589
std	111.369988	100.211564	94.573223	2.438922	4.185780
min	483.000000	74.000000	1157.000000	1.000000	1.000000
25%	1904.000000	2337.000000	1918.000000	1.000000	4.000000
50%	1991.000000	2366.000000	1988.000000	4.000000	7.000000
75%	2076.000000	2411.000000	2033.000000	7.000000	11.000000
max	2895.000000	2968.000000	4095.000000	7.000000	15.000000

In [12]:

```
means = pd.DataFrame(columns = ['x_acceleration_mean','y_acceleration_mean','z_acceleration_mean','
Labels'])
grouped = df.groupby(df.Labels)


lst = []
lst2 = []
lst3 = []
lst4 = []
for val in range(1,8):
    label = grouped.get_group(val)
    lst.append(label['x_acceleration'].mean())
    lst2.append(label['y_acceleration'].mean())
    lst3.append(label['z_acceleration'].mean())
    lst4.append(val)

means['x_acceleration_mean'] = lst
means['y_acceleration_mean'] = lst2
means['z_acceleration_mean'] = lst3
means['Labels'] = lst4

means
```

Out[12]:

	x_acceleration_mean	y_acceleration_mean	z_acceleration_mean	Labels
0	1977.689653	2376.558532	1966.415593	1
1	1969.489431	2371.051965	1940.448703	2
2	1996.272755	2378.303095	1965.729391	3
3	1976.819111	2386.292905	1978.708646	4
4	2000.554449	2385.493844	1997.001573	5
5	2027.107076	2374.075277	1952.189366	6
6	1997.845983	2388.535898	1973.053026	7

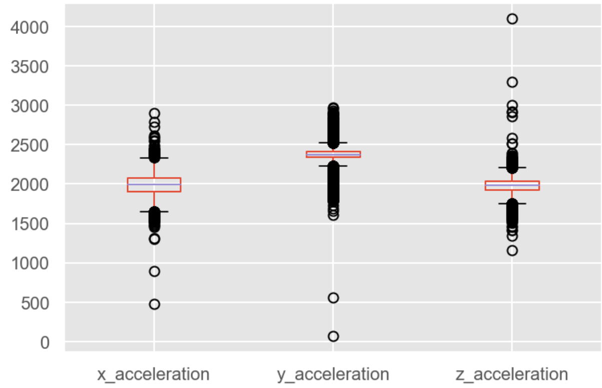
Graphical representation

In [13]:

```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.style.use("ggplot")
```

In [14]:

```
df_rnd.boxplot(column=['x_acceleration','y_acceleration','z_acceleration']);
```

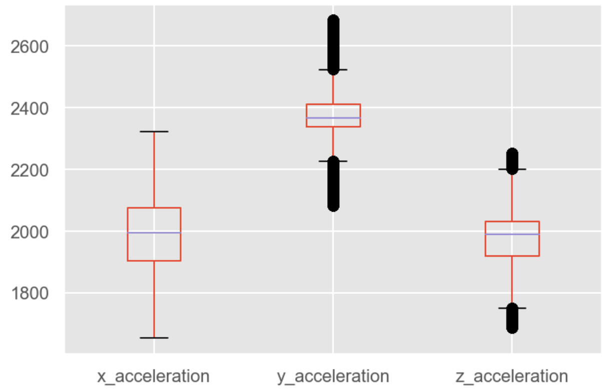


In [15]:

```
from scipy import stats
temp = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
```

In [16]:

```
temp.boxplot(column=['x_acceleration','y_acceleration','z_acceleration']);
```



In [17]:

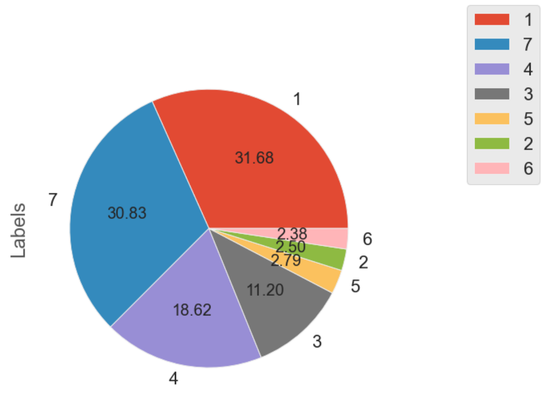
```
means = pd.DataFrame(columns = ['x_acceleration_mean','y_acceleration_mean','z_acceleration_mean','
Labels'])
grouped = temp.groupby(temp.Labels)


lst = []
lst2 = []
lst3 = []
lst4 = []
for val in range(1,8):
    label = grouped.get_group(val)
    lst.append(label['x_acceleration'].mean())
    lst2.append(label['y_acceleration'].mean())
    lst3.append(label['z_acceleration'].mean())
    lst4.append(val)

means['x_acceleration_mean'] = lst
means['y_acceleration_mean'] = lst2
means['z_acceleration_mean'] = lst3
means['Labels'] = lst4

means
```

Out[17]:

	x_acceleration_mean	y_acceleration_mean	z_acceleration_mean	Labels
0	1977.124868	2380.634526	1969.635385	1
1	1968.976390	2370.137015	1940.871289	2
2	1996.980548	2376.779558	1965.878496	3
3	1977.721802	2382.146119	1980.369508	4
4	2000.629027	2382.777215	1996.469772	5
5	2026.990742	2373.108891	1951.914391	6
6	2001.113811	2393.558041	1972.517494	7

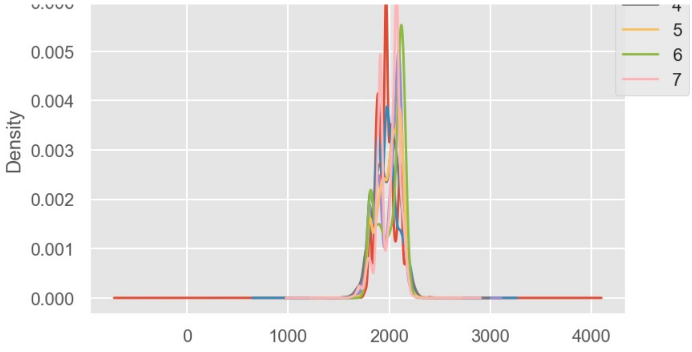
In [18]:

```
df_rnd['Labels'].value_counts().plot(kind='pie',autopct='%.2f')
plt.figlegend()
plt.show()
```



In [19]:

```
df_rnd.groupby('Labels')['x_acceleration'].plot.kde();
plt.autoscale(enable=True, axis= 'both',tight=None)
plt.figlegend();
```

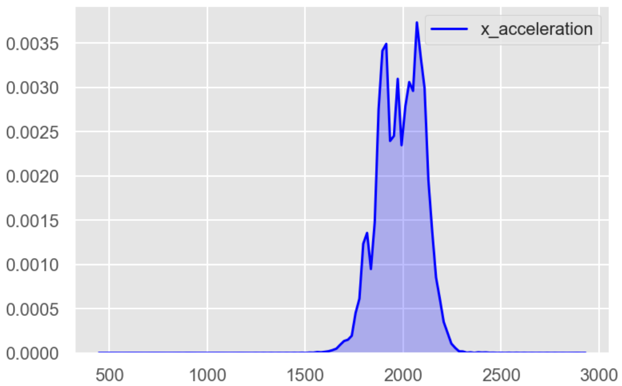


In [21]:

```
sns.kdeplot(df_rnd['x_acceleration'],shade = True, color = 'blue')
```

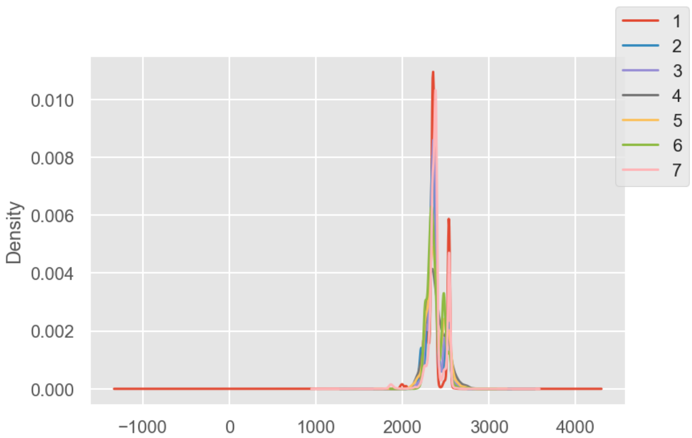
Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19505305d90>
```



In [22]:

```
df_rnd.groupby('Labels')['y_acceleration'].plot.kde();
plt.figlegend();
```

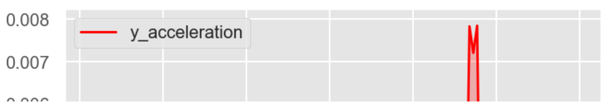


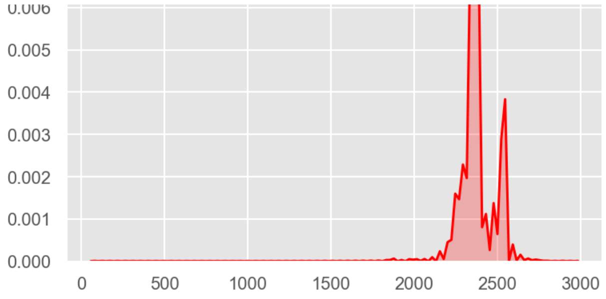
In [23]:

```
sns.kdeplot(df_rnd['y_acceleration'],shade = True, color = 'red')
```

Out[23]:

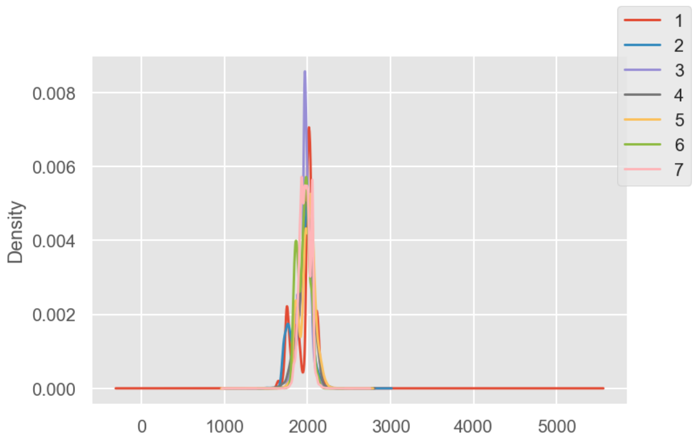
```
<matplotlib.axes._subplots.AxesSubplot at 0x195053bceb0>
```





In [24]:

```
df_rnd.groupby('Labels')['z_acceleration'].plot.kde();
plt.autoscale(enable=True, axis= 'both',tight=None)
plt.figlegend();
```

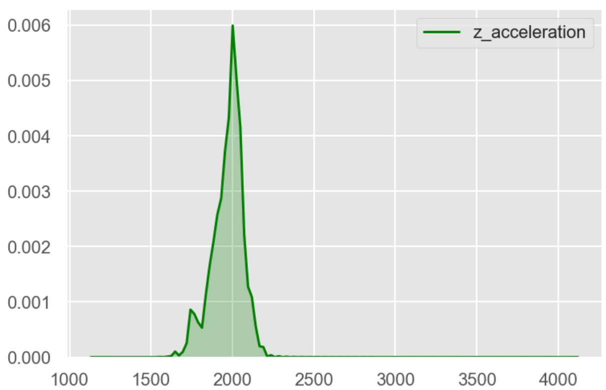


In [25]:

```
sns.kdeplot(df_rnd['z_acceleration'],shade = True, color = 'green')
```

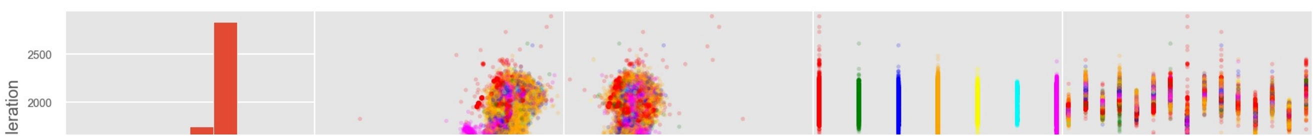
Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1950565d430>
```



In [26]:

```
from pandas.plotting import scatter_matrix
colors_palette = {1: 'red', 2:'green',3:'blue',4:'orange',5:'yellow',6:'cyan',7:'magenta'}
colors = [colors_palette[c] for c in df_rnd['Labels']]
scatter_matrix(df_rnd, alpha = 0.2, figsize = (16,16), diagonal = 'hist',c=colors)
plt.show()
```



In [27]:

```
df_rnd.corr(method='pearson')
```

Out[27]:

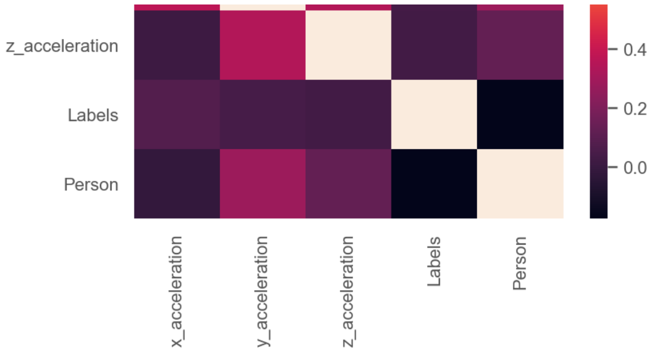
	x_acceleration	y_acceleration	z_acceleration	Labels	Person
x_acceleration	1.000000	0.371044	0.015191	0.079724	-0.018510
y_acceleration	0.371044	1.000000	0.340963	0.044100	0.284180
z_acceleration	0.015191	0.340963	1.000000	0.030303	0.127364
Labels	0.079724	0.044100	0.030303	1.000000	-0.174606
Person	-0.018510	0.284180	0.127364	-0.174606	1.000000

In [28]:

```
sns.heatmap(df_rnd.corr(method='pearson'))
```

Out[28]:

<matplotlib.axes._subplots.AxesSubplot at 0x19507fce2b0>



Data Modeling

The sampled dataset which is present right now is a numerical descriptive feature. So thereby scaling descriptive features, we can then train them on a classification model.

In [29]:

```
df['Labels'].value_counts()
```

Out[29]:

```
1    608667
7    593563
4    357064
3    216737
5     51498
2     47878
6     47770
0      3719
Name: Labels, dtype: int64
```

In [30]:

```
Data = df_rnd.drop(columns = ['Person','Labels']).values
target = df_rnd['Labels'].values
```

In [31]:

```
from sklearn import preprocessing

target =  preprocessing.LabelEncoder().fit_transform(target)
```

In [32]:

```
np.unique(target, return_counts=True)
```

Out[32]:

```
(array([0, 1, 2, 3, 4, 5, 6], dtype=int64),
 array([31627,  2498, 11178, 18590,  2785,  2371, 30770], dtype=int64))
```

Scalling Features

In [33]:

```
#Using standard scaling techniques

Data = preprocessing.StandardScaler().fit_transform(Data)
```

Scaling each of these descriptive feature is done by : scaled_value= value - mean/std. dev.

In [34]:

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

```
D_train, D_test, t_train, t_test = train_test_split(Data,
                                                    target,
                                                    test_size = 0.3,
                                                    random_state = 6758)
```

K-Nearest Neighbor classifier

In [36]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier()
knn_classifier.fit(D_train, t_train)
kNN = knn_classifier.score(D_test, t_test)
print('KNN Classifier : ', kNN)
```

```
KNN Classifier :  0.7180257797368597
```

Decision Tree Classifier

In [37]:

```
from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state= 6758)
dt_classifier.fit(D_train, t_train)
dT = dt_classifier.score(D_test, t_test)
print('DecisionTreeClassifier : ', dT)
```

```
DecisionTreeClassifier :  0.51829960595739
```

We can see that the accuracy is very less for both the cases without feeding in any parameters. Let us fine tune our parameters to see if we can fair better.

Hyper- Parameter Tuning

k-Nearest Neighbor classifier

In [38]:

```
from sklearn.model_selection import RepeatedStratifiedKFold

cv_method = RepeatedStratifiedKFold(n_splits=5,
                                    n_repeats=3,
                                    random_state= 6758)
```

In [39]:

```
import numpy as np
params_KNN = {'n_neighbors': [1,3, 5, 7, 11, 15, 20, 25],
              'p': [1, 2, 5]}
```

In [40]:

```
from sklearn.model_selection import GridSearchCV

gs_KNN = GridSearchCV(estimator=KNeighborsClassifier(),
                      param_grid=params_KNN,
                      cv=cv_method,
                      verbose=1,  # verbose: the higher, the more messages
                      scoring='accuracy',
                      n_jobs= -2,
                      return_train_score=True)
```

In [41]:

```
gs_KNN.fit(Data, target);
```

```
Fitting 15 folds for each of 24 candidates, totalling 360 fits
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Done  44 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-2)]: Done 194 tasks      | elapsed:  8.5min
[Parallel(n_jobs=-2)]: Done 360 out of 360 | elapsed: 18.9min finished
```

In [42]:

```
gs_KNN.best_params_
```

Out[42]:

```
{'n_neighbors': 20, 'p': 2}
```

In [43]:

```
gs_KNN.best_score_
```

Out[43]:

```
0.7417492378598932
```

In [44]:

```
gs_KNN.cv_results_['mean_test_score']
```

Out[44]:

```
array([0.65973411, 0.6598777 , 0.65959386, 0.69766612, 0.69818707,
       0.69775293, 0.72025701, 0.72141911, 0.72120205, 0.72988443,
       0.73029851, 0.7299412 , 0.73725778, 0.73805255, 0.73776536,
       0.73982575, 0.74071737, 0.74036005, 0.74103127, 0.74174924,
       0.74106801, 0.7411782 , 0.74143868, 0.74102126])
```

In [45]:

```
import pandas as pd

results_KNN = pd.DataFrame(gs_KNN.cv_results_['params'])
```

In [46]:

```
results_KNN['test_score'] = gs_KNN.cv_results_['mean_test_score']
```

In [47]:

```
results_KNN['metric'] = results_KNN['p'].replace([1,2,5], ["Manhattan", "Euclidean", "Minkowski"])
results_KNN
```

Out[47]:

	n_neighbors	p	test_score	metric
0	1	1	0.659734	Manhattan
1	1	2	0.659878	Euclidean
2	1	5	0.659594	Minkowski
3	3	1	0.697666	Manhattan
4	3	2	0.698187	Euclidean
5	3	5	0.697753	Minkowski

	n_neighbors	p	test_score	metric
6	5	1	0.720257	Manhattan
7	5	2	0.721419	Euclidean
8	5	5	0.721202	Minkowski
9	7	1	0.729884	Manhattan
10	7	2	0.730299	Euclidean
11	7	5	0.729941	Minkowski
12	11	1	0.737258	Manhattan
13	11	2	0.738053	Euclidean
14	11	5	0.737765	Minkowski
15	15	1	0.739826	Manhattan
16	15	2	0.740717	Euclidean
17	15	5	0.740360	Minkowski
18	20	1	0.741031	Manhattan
19	20	2	0.741749	Euclidean
20	20	5	0.741068	Minkowski
21	25	1	0.741178	Manhattan
22	25	2	0.741439	Euclidean
23	25	5	0.741021	Minkowski

In [50]:

```
import altair as alt
alt.renderers.enable('html')

alt.Chart(results_KNN,
          title='KNN Performance Comparison'
         ).mark_line(point=True).encode(
    alt.X('n_neighbors', title='Number of Neighbors'),
    alt.Y('test_score', title='Mean CV Score', scale=alt.Scale(zero=False)),
    color='metric'
).interactive()
```

Out[50]:

In [49]:

```
pip install altair
```

```
Collecting altairNote: you may need to restart the kernel to use updated packages.

  Downloading altair-4.1.0-py3-none-any.whl (727 kB)
Requirement already satisfied: entrypoints in c:\users\pujachouhan\anaconda3\lib\site-packages
(from altair) (0.3)
Requirement already satisfied: jsonschema in c:\users\pujachouhan\anaconda3\lib\site-packages
(from altair) (3.2.0)
Requirement already satisfied: toolz in c:\users\pujachouhan\anaconda3\lib\site-packages (from
altair) (0.10.0)
Requirement already satisfied: numpy in c:\users\pujachouhan\anaconda3\lib\site-packages (from
altair) (1.18.5)
Requirement already satisfied: pandas>=0.18 in c:\users\pujachouhan\anaconda3\lib\site-packages
(from altair) (1.0.5)
Requirement already satisfied: jinja2 in c:\users\pujachouhan\anaconda3\lib\site-packages (from
altair) (2.11.2)
Requirement already satisfied: attrs>=17.4.0 in c:\users\pujachouhan\anaconda3\lib\site-packages
(from jsonschema->altair) (19.3.0)
Requirement already satisfied: pyrsistent>=0.14.0 in c:\users\pujachouhan\anaconda3\lib\site-
packages (from jsonschema->altair) (0.16.0)
Requirement already satisfied: setuptools in c:\users\pujachouhan\anaconda3\lib\site-packages
(from jsonschema->altair) (49.2.0.post20200714)
Requirement already satisfied: six>=1.11.0 in c:\users\pujachouhan\anaconda3\lib\site-packages
(from jsonschema->altair) (1.15.0)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\pujachouhan\anaconda3\lib\site-p
ackages (from pandas>=0.18->altair) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\pujachouhan\anaconda3\lib\site-packages
```

```
(from pandas>=0.18->altair) (2020.1)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\pujachouhan\anaconda3\lib\site-packages (from jinja2->altair) (1.1.1)
Installing collected packages: altair
Successfully installed altair-4.1.0
```

We can see that the mean CV score always increases with respect to number of neighbors. This can lead to overfitting which should be avoided.

Decision Tree classifier

In [51]:

```
from sklearn.tree import DecisionTreeClassifier

df_classifier = DecisionTreeClassifier(random_state=6758)

params_DT = {'criterion': ['gini', 'entropy'],
             'max_depth': [5, 6, 7, 8, 10 , 12, 15, 17],
             'min_samples_split': [2, 3]}

gs_DT = GridSearchCV(estimator=df_classifier,
                     param_grid=params_DT,
                     cv=cv_method,
                     verbose=1,
                     n_jobs=-2,
                     scoring='accuracy')

gs_DT.fit(Data, target);
```

```
Fitting 15 folds for each of 32 candidates, totalling 480 fits
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Done  44 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-2)]: Done 194 tasks      | elapsed:   16.6s
[Parallel(n_jobs=-2)]: Done 444 tasks      | elapsed:   45.0s
[Parallel(n_jobs=-2)]: Done 480 out of 480 | elapsed:   51.8s finished
```

In [52]:

```
gs_DT.best_params_
```

Out[52]:

```
{'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 2}
```

In [53]:

```
gs_DT.best_score_
```

Out[53]:

```
0.7141225790520183
```

In [54]:

```
results_DT = pd.DataFrame(gs_DT.cv_results_['params'])
results_DT['test_score'] = gs_DT.cv_results_['mean_test_score']
results_DT.columns
```

Out[54]:

```
Index(['criterion', 'max_depth', 'min_samples_split', 'test_score'], dtype='object')
```

In [55]:

```
alt.Chart(results_DT,
          title='DT Performance Comparison'
         ).mark_line(point=True).encode(
```

```
    alt.X('max_depth', title='Maximum Depth'),
    alt.Y('test_score', title='Mean CV Score', aggregate='average', scale=alt.Scale(zero=False)),
    color='criterion'
).interactive()
```

Out[55]:

Predicting

Since we could conclude that the accuracy for KNN is much higher than Decision Tree, predicting the values with KNN is much more efficient. But however we will predict using both to compare both the scores.

In [56]:

```
t_pred_knn = gs_KNN.predict(D_test)
```

In [57]:

```
from sklearn import metrics
print('KNN Predicted accuracy score: ',metrics.accuracy_score(t_test, t_pred_knn))
```

```
KNN Predicted accuracy score:  0.7560609096373472
```

In [58]:

```
t_pred_dt = gs_DT.predict(D_test)
```

In [59]:

```
print('DecisionTree accuracy score : ',metrics.accuracy_score(t_test,t_pred_dt))
```

```
DecisionTree accuracy score :  0.7503172376945167
```

In [60]:

```
# Classification report for KNN

print(metrics.classification_report(t_test, t_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      9411
           1       0.59      0.15      0.24       747
           2       0.62      0.50      0.55      3438
           3       0.66      0.74      0.70      5515
           4       0.41      0.10      0.17       836
           5       0.56      0.19      0.28       750
           6       0.78      0.85      0.81      9249

    accuracy                           0.76     29946
   macro avg       0.64      0.49      0.52     29946
weighted avg       0.74      0.76      0.74     29946
```

In [61]:

```
print(metrics.confusion_matrix(t_test, t_pred_knn))
```

```
[[8674   39   98  224    6    4  366]
 [ 320  114   41  139    4    1  128]
 [ 223   10 1712  709   63   22  699]
 [ 470   13  294 4063   13   16  646]
 [  96    0  158  353   87    5  137]
 [  43    3  107  138   22  141  296]
 [ 419   15  352  534   18   61 7850]]
```

Looking at the F1-scores, we can see the model is not very good when it comes to label 1,4,5.While it is doing extremely well for the other labels

In [62]:

```
#Classification report for Decision tree

print(metrics.classification_report(t_test, t_pred_dt, labels=np.unique(t_pred_dt)))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      9411
           1       0.67      0.19      0.30       747
           2       0.60      0.48      0.54      3438
           3       0.67      0.74      0.70      5515
           4       0.46      0.12      0.20       836
           5       0.60      0.17      0.26       750
           6       0.74      0.85      0.79      9249

    accuracy                           0.75     29946
   macro avg       0.66      0.49      0.52     29946
weighted avg       0.74      0.75      0.73     29946
```

In [63]:

```
print(metrics.confusion_matrix(t_test, t_pred_dt))
```

```
[[8461   40  123  256    5    2  524]
 [ 252  144   28  135    1    0  187]
 [ 213    7 1653  651   65   25  824]
 [ 389    8  289 4092   11    8  718]
 [  80    3  152  318  104    6  173]
 [  30    4  103  143   26  125  319]
 [ 381    9  390  520   16   43 7890]]
```

The F1 score for DT in label 4, 5 has a lower F1 score. Whilst the model is performing fairly well when looking at the other labels.