Análisis exploratorio de datos

In [ ]:
```
# Importamos librerías:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [ ]:
```
# Importamos los datos:
data_train = pd.read_csv('./train_motion_data.csv')
data_test = pd.read_csv('./test_motion_data.csv')
```

In [ ]:
```
# Revisamos el set de entrenamiento:
data_train.head()
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	NORMAL	3581629
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	NORMAL	3581630
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	NORMAL	3581630
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	NORMAL	3581631
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	NORMAL	3581631

In [ ]:
```
data_train.sort_values(["Timestamp"])
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	NORMAL	3581629
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	NORMAL	3581630
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	NORMAL	3581630
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	NORMAL	3581631
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	NORMAL	3581631
...	...	...	...	...	...	...	...	...
3639	0.915688	-2.017489	1.687505	0.450360	0.384845	-1.236468	SLOW	3583789
3640	-1.934203	0.914925	-0.096013	0.321468	0.649350	-0.477162	SLOW	3583790
3641	-0.222845	0.747304	-0.887430	0.361174	-0.406836	0.054291	SLOW	3583790
3642	-0.349423	0.067261	0.394368	-0.132405	0.020159	-0.004963	SLOW	3583791
3643	-0.402428	0.406218	-0.423009	-0.053603	-0.006720	0.001145	SLOW	3583791

3644 rows × 8 columns

In [ ]:
```
data_train.sort_values(["Timestamp"]).loc[data_train.Class == "NORMAL"]
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	NORMAL	3581629
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	NORMAL	3581630
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	NORMAL	3581630
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	NORMAL	3581631
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	NORMAL	3581631
...	...	...	...	...	...	...	...	...
1195	-0.820672	2.556599	-0.617599	-0.031612	-0.865596	-0.962189	NORMAL	3582274
1197	2.488864	-1.001262	0.432143	0.140041	-0.080023	0.051847	NORMAL	3582275
1196	-0.016871	1.171574	0.603792	0.253662	0.692110	0.537485	NORMAL	3582275
1198	-0.680338	-0.048300	-3.298533	-0.169057	-0.383012	0.217392	NORMAL	3582276
1199	0.563353	1.234707	-0.729512	0.226784	-1.290758	-0.309174	NORMAL	3582276

1200 rows × 8 columns

In [ ]:

```
# Revisamos el set de prueba:
data_test.head()
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.758194	-0.217791	0.457263	0.000000	0.000000	0.000000	AGGRESSIVE	818922
1	0.667560	-0.038610	0.231416	-0.054367	-0.007712	0.225257	AGGRESSIVE	818923
2	2.724449	-7.584121	2.390926	0.023824	0.013668	-0.038026	AGGRESSIVE	818923
3	2.330950	-7.621754	2.529024	0.056810	-0.180587	-0.052076	AGGRESSIVE	818924
4	2.847215	-6.755621	2.224640	-0.031765	-0.035201	0.035277	AGGRESSIVE	818924

In [ ]:

```
# Lógicamente, ambos sets describen el mismo fenómeno y cuentan con las mism
# Imprimimos las columnas (o variables) involucradas:
data_train.columns
```

Out[ ]:

```
Index(['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ', 'Class',
       'Timestamp'],
      dtype='object')
```

In [ ]:

```
# Observamos si hay datos faltantes (en el caso del set de entrenamiento):
pd.isna(data_train).sum()
```

Out[ ]:

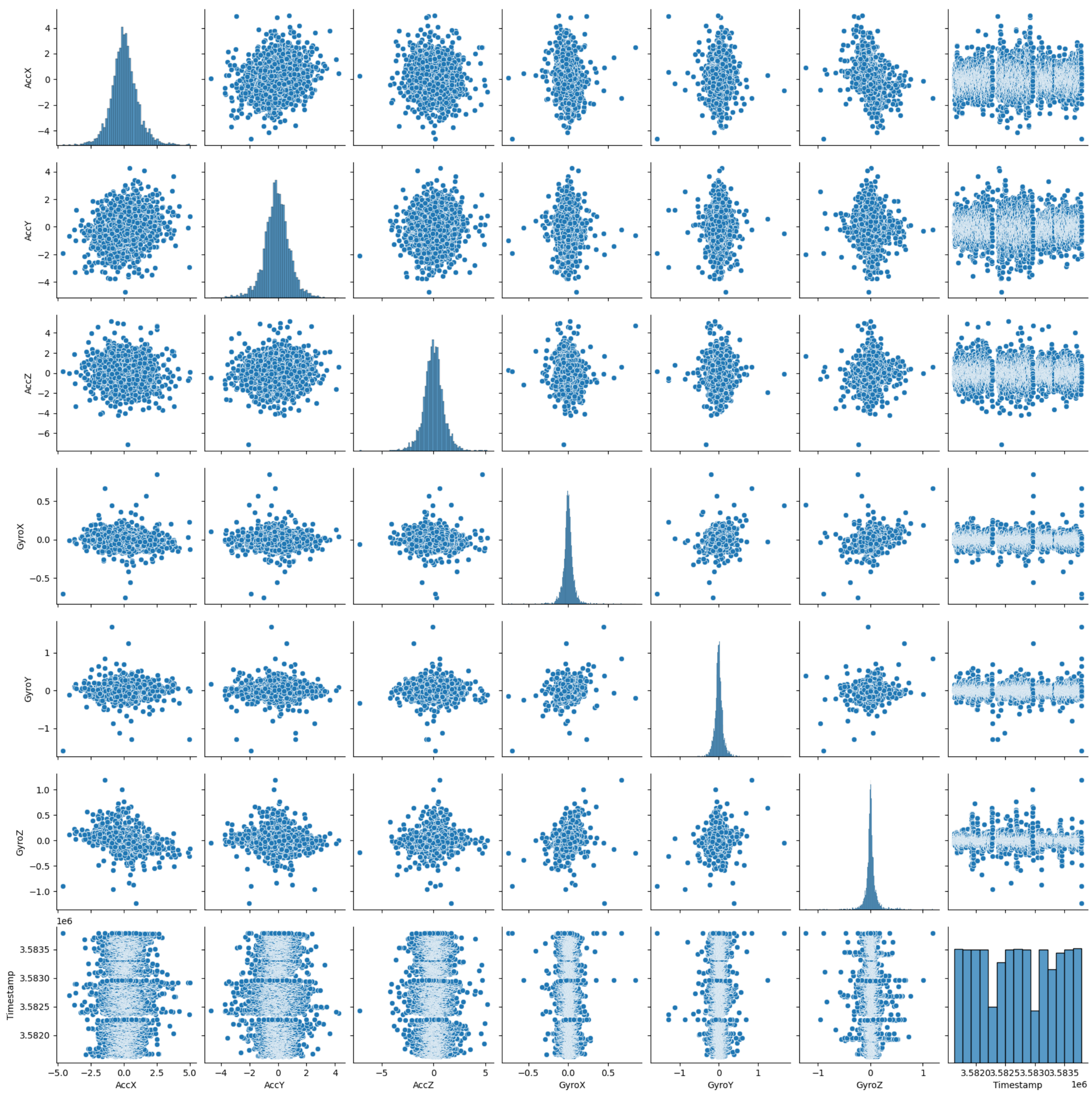
```
AccX         0
AccY         0
AccZ         0
GyroX        0
GyroY        0
GyroZ        0
Class        0
Timestamp    0
dtype: int64
```

In [ ]:
```
# Observamos si hay datos faltantes (en el caso del set de prueba):
pd.isna(data_test).sum()
```

Out[ ]:
```
AccX         0
AccY         0
AccZ         0
GyroX        0
GyroY        0
GyroZ        0
Class        0
Timestamp    0
dtype: int64
```

In [ ]:
```
# No habrán de imputarse los datos en ningún momento del análisis.

# Observamos de manera gráfica un primer vistazo de la distribución de los c

# Realizamos ploteos entre pares:
sns.pairplot(data_train, vars=["AccX","AccY","AccZ","GyroX", "GyroY", "GyroZ
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f36375af130>



In [ ]:
```
# Se observan diversos comportamientos entre variables, existiendo relacione

# Computamos e imprimimos estadísticas basadas en las clases que nos present

# Categorías
print( data_train['Class'].unique() )

# Conteo por categoría
print( data_train['Class'].value_counts() )

# Proporciones de datos existentes por categoría
Nc1      = len(data_train[data_train['Class']=='SLOW'])
Nc2      = len(data_train[data_train['Class']=='NORMAL'])
Nc3      = len(data_train[data_train['Class']=='AGGRESSIVE'])
Ntotal   = Nc1 + Nc2 + Nc3

pct_Nc1  = Nc1 / Ntotal
pct_Nc2  =  Nc2 / Ntotal
pct_Nc3  = Nc3 / Ntotal
print("Proporción de SLOW       es {0:0.1f}%".format(pct_Nc1*100))
print("Proporción de NORMAL     es {0:0.1f}%".format(pct_Nc2*100))
print("Proporción of AGGRESSIVE es {0:0.1f}%".format(pct_Nc3*100))
```

```
['NORMAL' 'AGGRESSIVE' 'SLOW']
SLOW          1331
NORMAL        1200
AGGRESSIVE    1113
Name: Class, dtype: int64
Proporción de SLOW       es 36.5%
Proporción de NORMAL     es 32.9%
Proporción of AGGRESSIVE es 30.5%
```

In [ ]:
```
# Observamos que los datos no se encuentran balanceados, por lo que habrá qu

# Codificamos la etiqueta de cada clase en todas las ocurrencias del conjunt
data_train.Class = [2 if i == 'NORMAL' else 3 if i == 'AGGRESSIVE' else 1 fo

# Visualización del conjunto de datos con la clase codificada
data_train
```

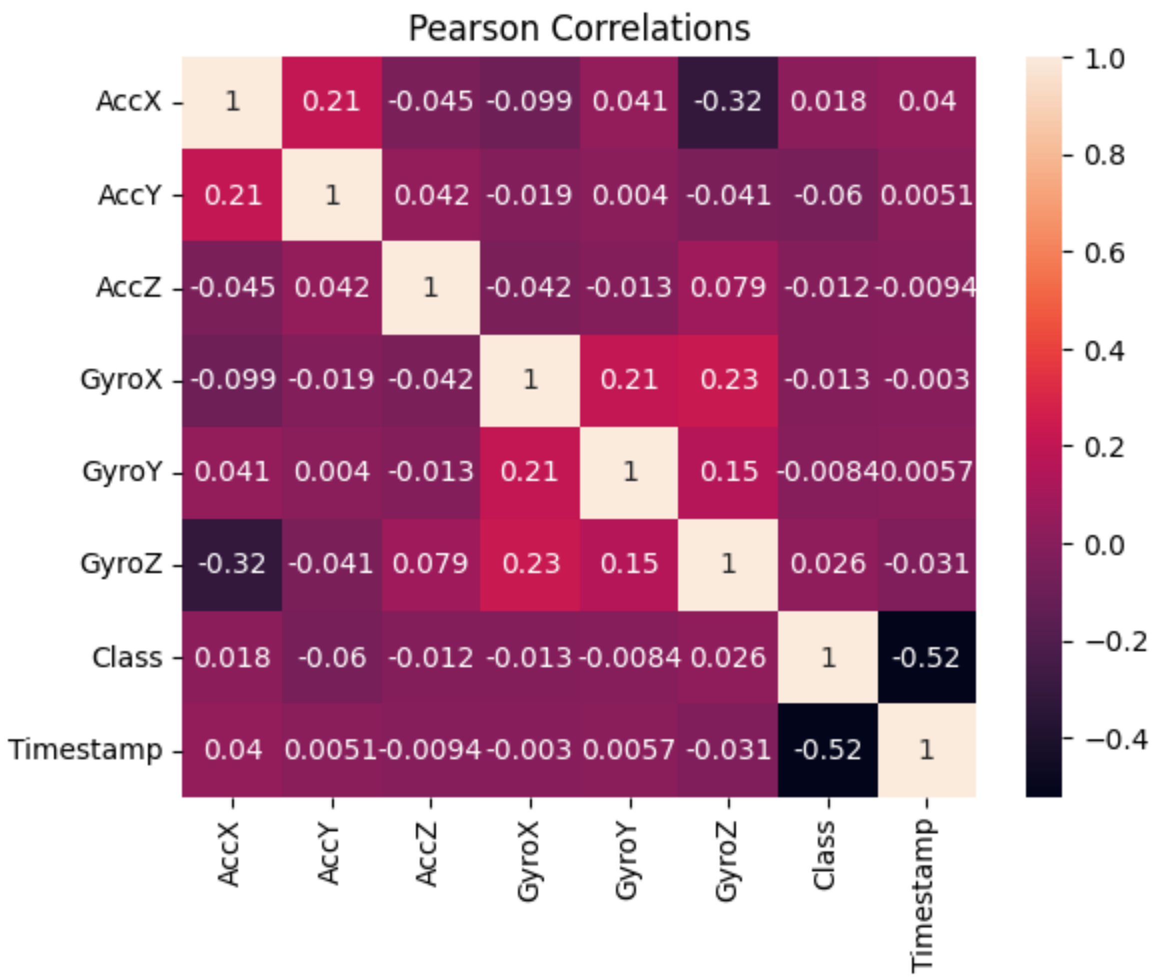
Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	2	3581629
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	2	3581630
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	2	3581630
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	2	3581631
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	2	3581631
...	...	...	...	...	...	...	...	...
3639	0.915688	-2.017489	1.687505	0.450360	0.384845	-1.236468	1	3583789
3640	-1.934203	0.914925	-0.096013	0.321468	0.649350	-0.477162	1	3583790
3641	-0.222845	0.747304	-0.887430	0.361174	-0.406836	0.054291	1	3583790
3642	-0.349423	0.067261	0.394368	-0.132405	0.020159	-0.004963	1	3583791
3643	-0.402428	0.406218	-0.423009	-0.053603	-0.006720	0.001145	1	3583791

3644 rows × 8 columns

In [ ]:

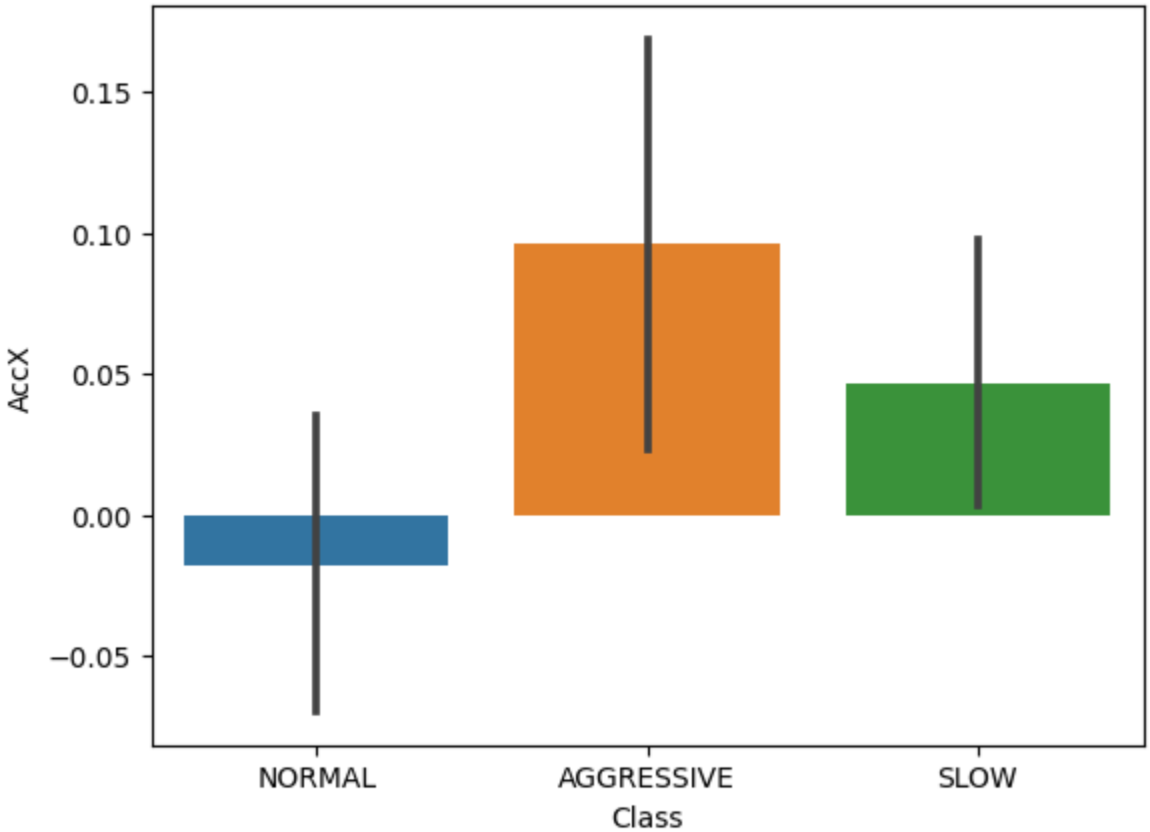
```
# Calculamos la correlación entre variables:
ax = sns.heatmap(data_train.corr(), annot=True).set(title='Pearson Correlati
```



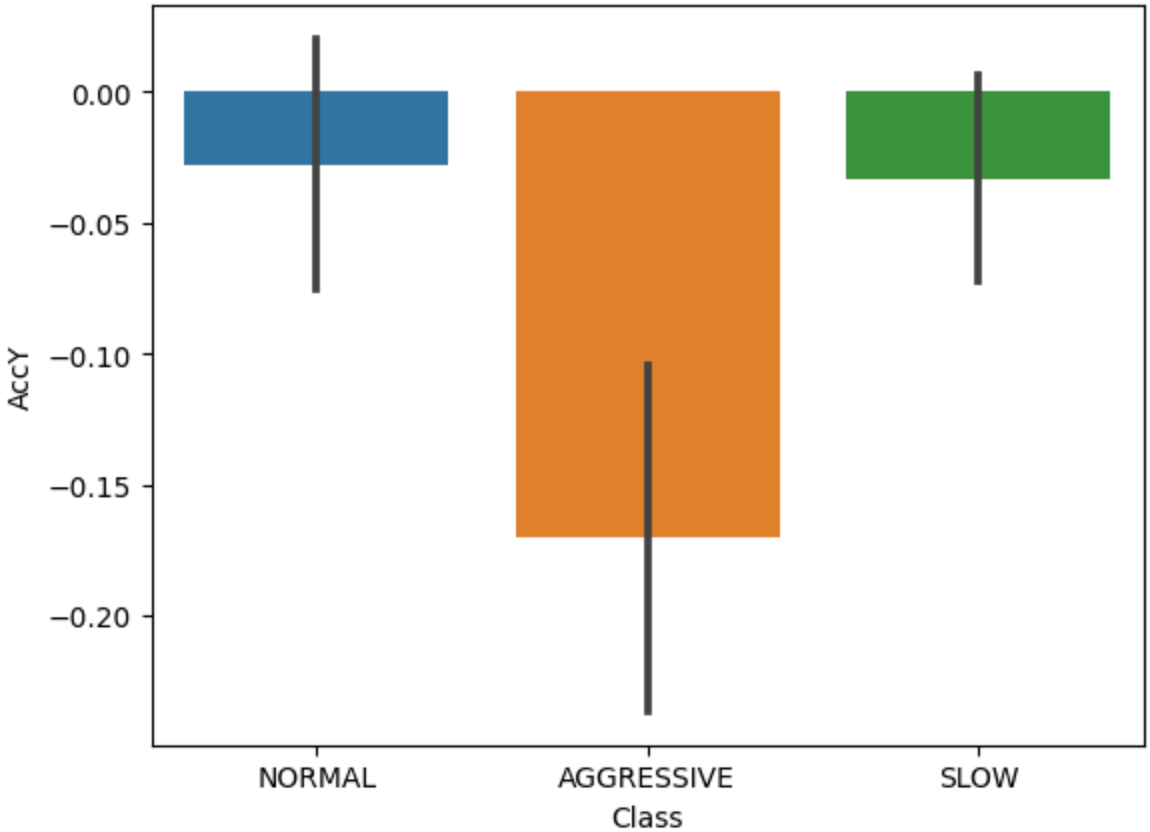
In [ ]:
```
# Graficando aceleración en X y giroscopio en Z, clasificados en función de

accXSLOW = data_train.loc[data_train.Class == 1]
accXNORMAL = data_train.loc[data_train.Class == 2]
accXAGGRESSIVE = data_train.loc[data_train.Class == 3]
```

In [ ]:
```
data_train2 = pd.read_csv('./train_motion_data.csv')
```

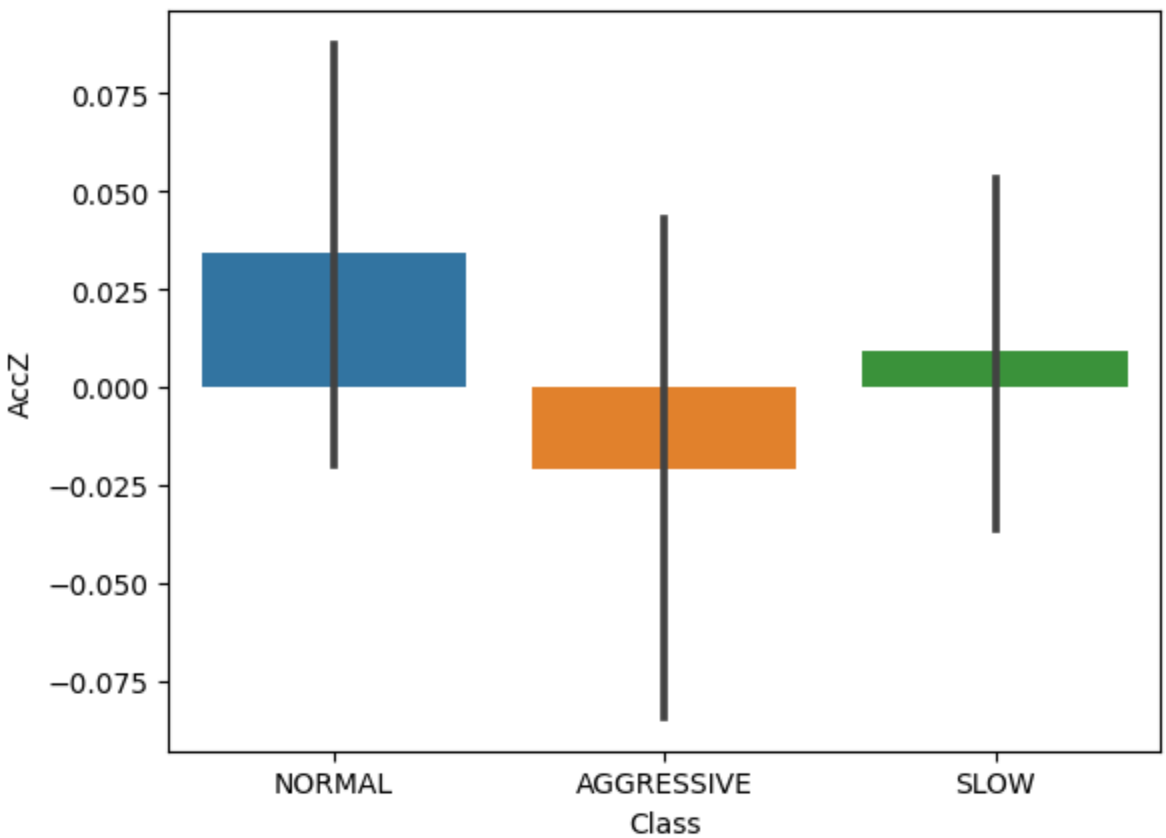
In [ ]:
```
ax = sns.barplot(x="Class", y="AccX", data=data_train2)
```



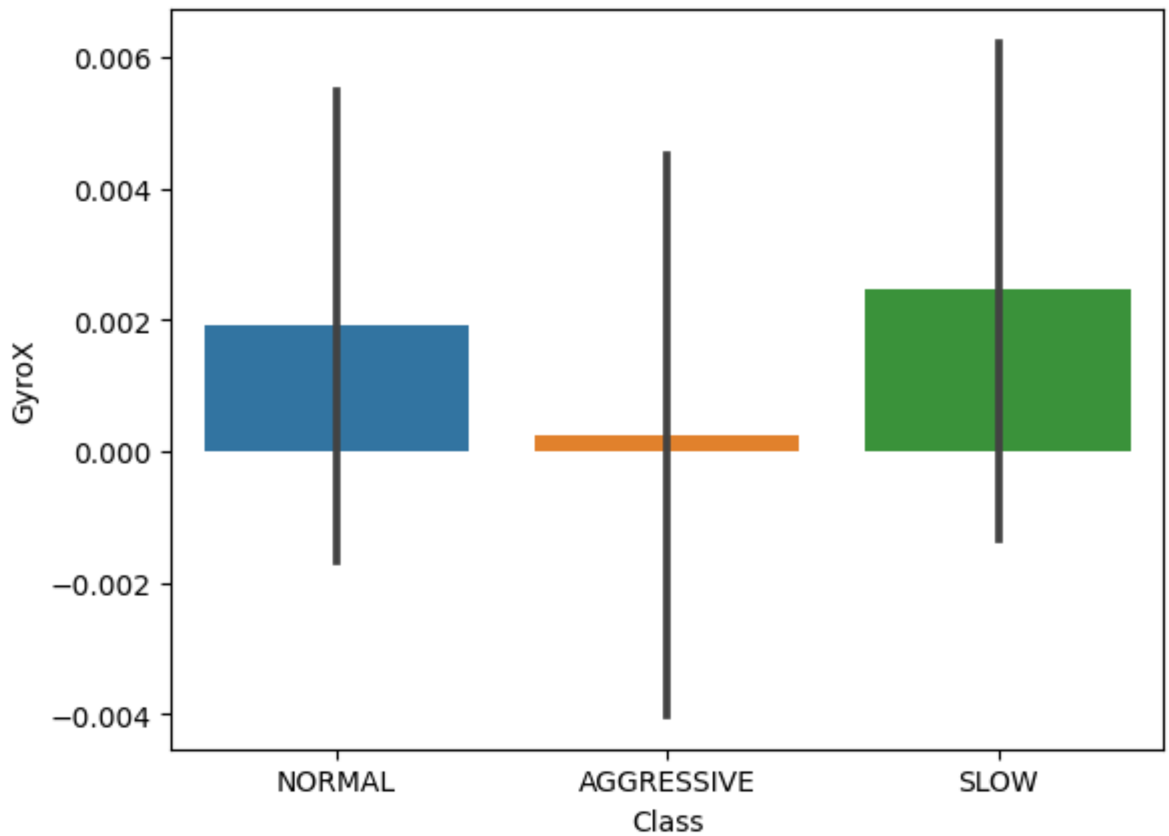
In [ ]:
```
ax3 = sns.barplot(x="Class", y="AccY", data=data_train2)
```



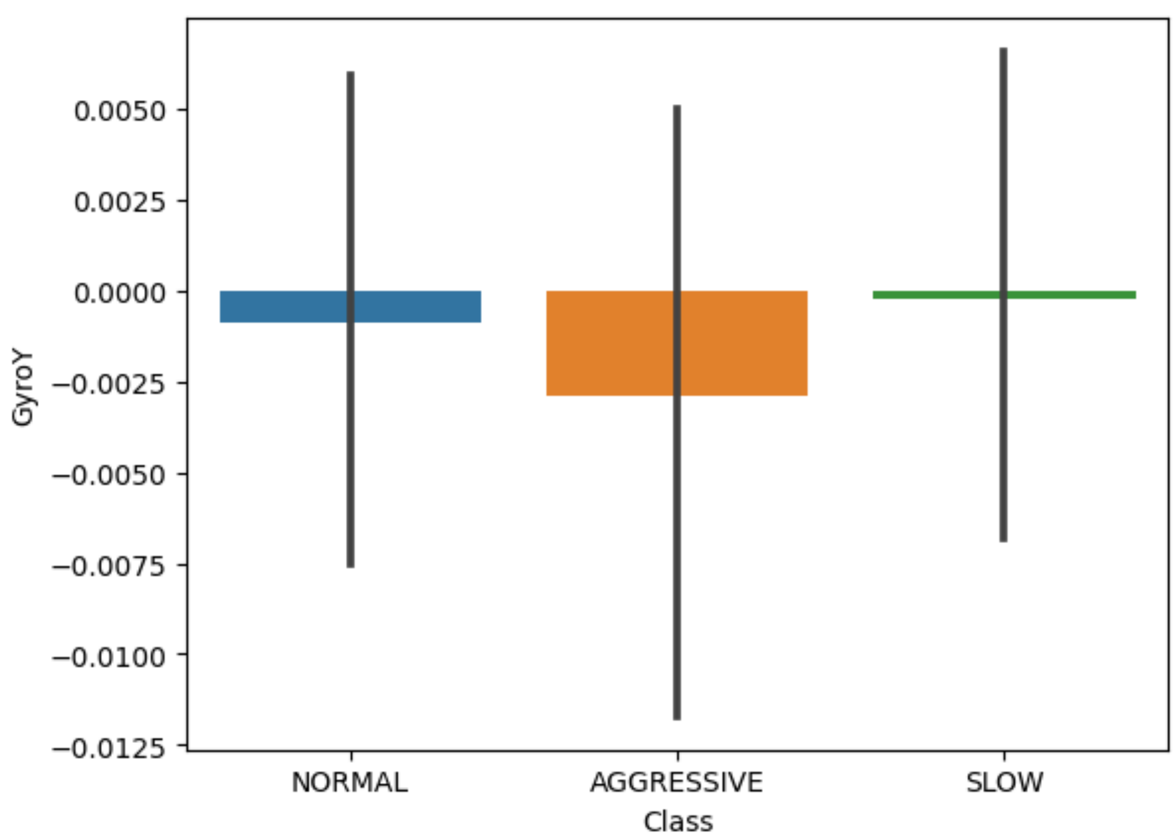
In [ ]:
```
ax3 = sns.barplot(x="Class", y="AccZ", data=data_train2)
```



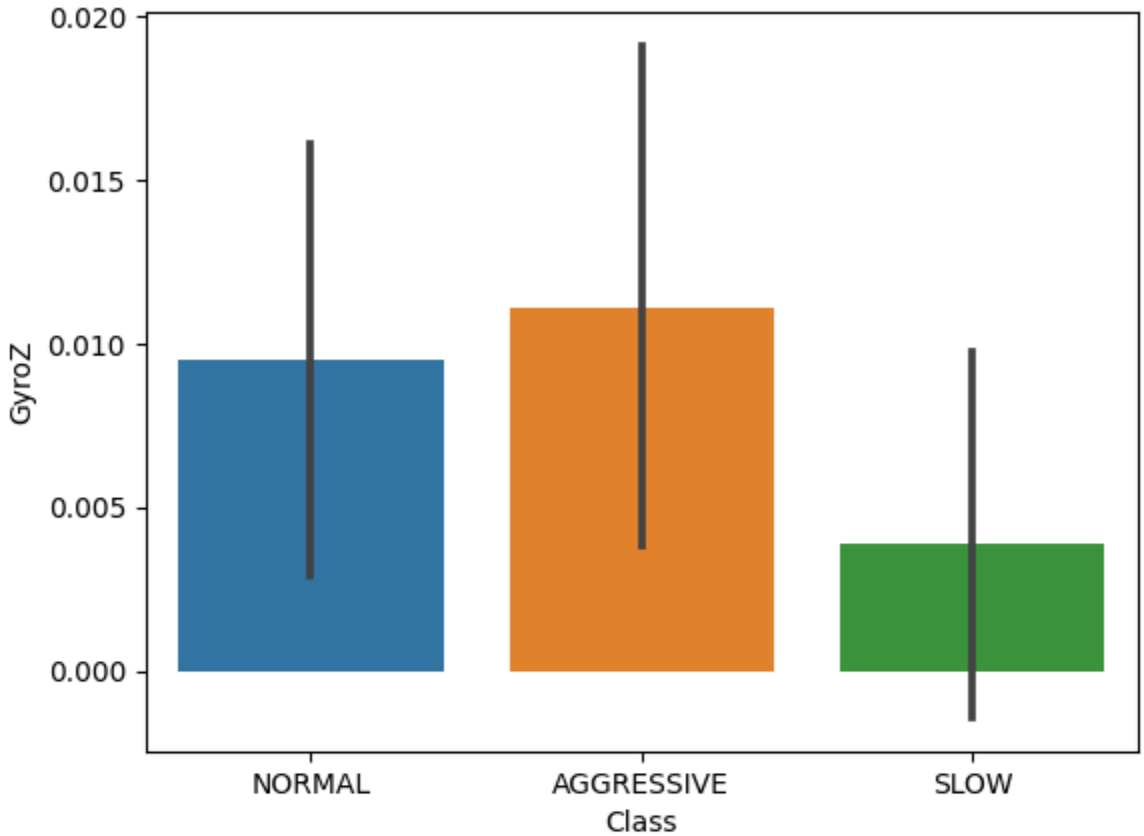
In [ ]:
```
ax3 = sns.barplot(x="Class", y="GyroX", data=data_train2)
```



In [ ]:
```
ax3 = sns.barplot(x="Class", y="GyroY", data=data_train2)
```



In [ ]:
```
ax2 = sns.barplot(x="Class", y="GyroZ", data=data_train2)
```



Implementación del modelo de Random Forest

Tres clases: SLOW, NORMAL y AGGRESSIVE

In [ ]:
```
# AccX distingue entre manejo etiquetado como normal (valores negativos) los
# AccZ distingue entre manejo etiquetado como agresivo (valores negativos) y

# Implementamos Random Forest
X = data_train[['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ']]
X
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902
...	...	...	...	...	...	...
3639	0.915688	-2.017489	1.687505	0.450360	0.384845	-1.236468
3640	-1.934203	0.914925	-0.096013	0.321468	0.649350	-0.477162
3641	-0.222845	0.747304	-0.887430	0.361174	-0.406836	0.054291
3642	-0.349423	0.067261	0.394368	-0.132405	0.020159	-0.004963
3643	-0.402428	0.406218	-0.423009	-0.053603	-0.006720	0.001145

3644 rows × 6 columns

In [ ]:

```
y = data_train['Class']  # Labels
y
```

Out[ ]:

```
0       2
1       2
2       2
3       2
4       2
       ..
3639    1
3640    1
3641    1
3642    1
3643    1
Name: Class, Length: 3644, dtype: int64
```

In [ ]:

```
X_test = data_test[['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ']]
X_test
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ
0	0.758194	-0.217791	0.457263	0.000000	0.000000	0.000000
1	0.667560	-0.038610	0.231416	-0.054367	-0.007712	0.225257
2	2.724449	-7.584121	2.390926	0.023824	0.013668	-0.038026
3	2.330950	-7.621754	2.529024	0.056810	-0.180587	-0.052076
4	2.847215	-6.755621	2.224640	-0.031765	-0.035201	0.035277
...	...	...	...	...	...	...
3079	-0.713858	-0.652975	-0.164015	-0.147829	-1.309466	0.517250
3080	1.514261	0.330070	1.020714	1.321302	1.707598	-0.674548
3081	1.280216	-1.735172	-2.332695	0.583376	0.690507	-0.468075
3082	0.912313	0.583314	-0.965622	0.235794	0.512745	0.406073
3083	1.462172	0.190287	0.019377	-0.254731	-0.279547	0.076205

3084 rows × 6 columns

In [ ]:

```
# Codificamos la etiqueta de cada clase en todas las ocurrencias del conjunt
data_test.Class = [2 if i == 'NORMAL' else 3 if i == 'AGGRESSIVE' else 1 for

y_test = data_test['Class']
y_test
```

Out[ ]:

```
0       3
1       3
2       3
3       3
4       3
       ..
3079    1
3080    1
3081    1
3082    1
3083    1
Name: Class, Length: 3084, dtype: int64
```

In [ ]:
```
# Normalizamos los datos
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
norm_train_data_X = pd.DataFrame(sc.fit_transform(X), columns = X.columns)
norm_test_data_X = pd.DataFrame(sc.fit_transform(X_test), columns = X_test.c

# Importamos el modelo de Random Forest desde sklearn
from sklearn.ensemble import RandomForestClassifier

# Creamos una instancia del clasificador
clf = RandomForestClassifier(n_estimators=500, max_depth=8, max_features=2)

# Entrenamos el modelo
clf.fit(norm_train_data_X,y)

# Predecimos la salida con base en los datos de entrada (3 clases)
y_pred = clf.predict(X_test)

# Obtenemos la puntuación del modelo
clf.score(X_test,y_test)
```

Out[ ]:
```
0.4724383916990921
```

In [ ]:
```
# Imprimimos métricas de evaluación
from sklearn.metrics import classification_report, confusion_matrix, accurac
print('Matriz de confusión:\n', confusion_matrix(y_test, y_pred))
```

```
Matriz de confusión:
 [[1122   36  115]
 [ 771   42  184]
 [ 492   29  293]]
```

In [ ]:
```
print('\nReporte de clasificación:\n', classification_report(y_test, y_pred)
print('\nAccuracy score:\n', accuracy_score(y_test, y_pred))
```

```
Reporte de clasificación:
               precision    recall  f1-score   support

           1       0.47      0.88      0.61      1273
           2       0.39      0.04      0.08       997
           3       0.49      0.36      0.42       814

    accuracy                           0.47      3084
   macro avg       0.45      0.43      0.37      3084
weighted avg       0.45      0.47      0.39      3084


Accuracy score:
 0.4724383916990921
```

Random Forest acotado a dos clases (SLOW-NORMAL y AGGRESSIVE)

In [ ]:
```
# Importamos los datos
data_train_dropslow = pd.read_csv('./train_motion_data.csv')
data_test_dropslow = pd.read_csv('./test_motion_data.csv')

# Codificamos la etiqueta de cada clase en todas las ocurrencias del conjunt
data_train_dropslow.Class = [0 if i == 'NORMAL' else 1 if i == 'AGGRESSIVE' 
data_test_dropslow.Class = [0 if i == 'NORMAL' else 1 if i == 'AGGRESSIVE' e

# Separamos variables
X_tr_dr = data_train_dropslow[['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'Gy
y_tr_dr = data_train_dropslow['Class']  # Labels

X_te_dr = data_test_dropslow[['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'Gyr
y_te_dr = data_test_dropslow['Class']  # Labels

# Normalizamos los datos
sc2 = StandardScaler()
norm_train_data_X_dr = pd.DataFrame(sc2.fit_transform(X_tr_dr), columns = X_
norm_test_data_X_dr = pd.DataFrame(sc2.fit_transform(X_te_dr), columns = X_t

# Importamos el modelo de Random Forest desde sklearn
from sklearn.ensemble import RandomForestClassifier

# Creamos una instancia del clasificador
clf2 = RandomForestClassifier(n_estimators=10000)

# Entrenamos el modelo
clf2.fit(norm_train_data_X_dr,y_tr_dr)

# Obtenemos la puntuación del modelo
clf2.score(X_tr_dr,y_tr_dr)
```

Out[ ]:
```
0.756860592755214
```

In [ ]:
```
# Predecimos la salida con base en los datos de entrada (2 clases)
y_pred_dr = clf2.predict(X_te_dr)

# Imprimimos métricas de evaluación
from sklearn.metrics import classification_report, confusion_matrix, accurac
print('Matriz de confusión:\n', confusion_matrix(y_te_dr, y_pred_dr, labels=
```

```
Matriz de confusión:
 [[2087  183]
 [ 606  208]]
```

In [ ]:
```
print('\nReporte de clasificación:\n', classification_report(y_te_dr, y_pred
print('\nAccuracy score:\n', accuracy_score(y_te_dr, y_pred_dr))
```

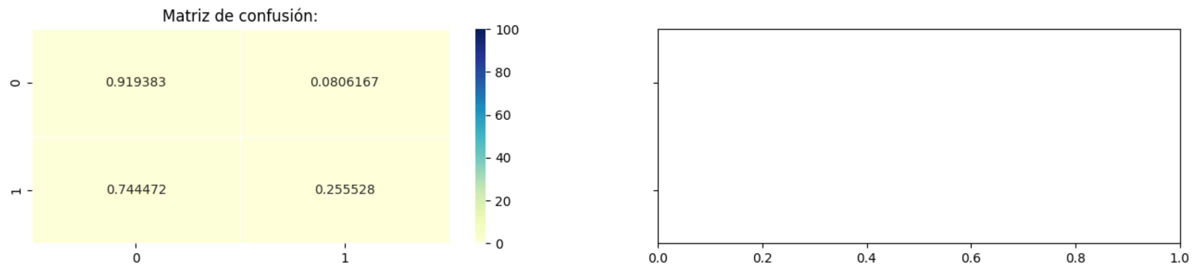
```
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.77      0.92      0.84      2270
           1       0.53      0.26      0.35       814

    accuracy                           0.74      3084
   macro avg       0.65      0.59      0.59      3084
weighted avg       0.71      0.74      0.71      3084


Accuracy score:
 0.7441634241245136
```

In [ ]:

```python
# Matriz de confusión a color
CM2     = confusion_matrix(y_te_dr, y_pred_dr, normalize = 'true')
fig, axes = plt.subplots(1, 2, figsize=(16, 3), sharey=True)

sns.heatmap(CM2, annot=True, fmt='g', ax=axes[0], cmap="YlGnBu", linewidths=
axes[0].set_title('Matriz de confusión:')
```

Out[ ]:

```
Text(0.5, 1.0, 'Matriz de confusión:')
```



Implementación de KNN y regresión logística

Se junta la clase 'SLOW' con 'NORMAL' y se elimina el ruido presente en los datos para una mejor presición en los modelos

In [ ]:
```
data_train.drop(data_train[data_train['AccX'] > 2.5].index, inplace = True)
data_train.drop(data_train[data_train['AccX'] < -2.5].index, inplace = True)

data_train.drop(data_train[data_train['AccY'] > 2.5].index, inplace = True)
data_train.drop(data_train[data_train['AccY'] < -2.5].index, inplace = True)

data_train.drop(data_train[data_train['AccZ'] > 2.5].index, inplace = True)
data_train.drop(data_train[data_train['AccZ'] < -2.5].index, inplace = True)

data_train.drop(data_train[data_train['GyroX'] > 0.4].index, inplace = True)
data_train.drop(data_train[data_train['GyroX'] < -0.4].index, inplace = True

data_train.drop(data_train[data_train['GyroY'] > 0.4].index, inplace = True)
data_train.drop(data_train[data_train['GyroY'] < -0.4].index, inplace = True

data_train.drop(data_train[data_train['GyroZ'] > 0.4].index, inplace = True)
data_train.drop(data_train[data_train['GyroZ'] < -0.4].index, inplace = True

data_train
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	2	3581629
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	2	3581630
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	2	3581630
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	2	3581631
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	2	3581631
...	...	...	...	...	...	...	...	...
3632	0.702303	-0.930822	0.809290	0.001985	-0.172264	-0.025733	1	3583786
3633	0.319258	0.272088	0.535243	0.018479	0.106901	-0.062385	1	3583786
3634	0.402702	0.432955	-0.683754	-0.090255	-0.085521	0.362167	1	3583787
3642	-0.349423	0.067261	0.394368	-0.132405	0.020159	-0.004963	1	3583791
3643	-0.402428	0.406218	-0.423009	-0.053603	-0.006720	0.001145	1	3583791

3347 rows × 8 columns

In [ ]:
```
data_train = data_train.replace({"Class": {"SLOW":0, "NORMAL":0, "AGGRESSIVE
test_motionDf = data_test.replace({"Class": {"SLOW":0, "NORMAL":0, "AGGRESSI
```

Se normalizan los datos

In [ ]:
```
# Initialize scaler
from sklearn.preprocessing import MinMaxScaler

print(data_train)

scaler = MinMaxScaler()

scaler.fit(data_train[['AccX']])

data_train['AccX_Scaled'] = scaler.transform(data_train[['AccX']])

scaler.fit(data_train[['AccY']])

data_train['AccY_Scaled'] = scaler.transform(data_train[['AccY']])

scaler.fit(data_train[['AccZ']])

data_train['AccZ_Scaled'] = scaler.transform(data_train[['AccZ']])

scaler.fit(data_train[['GyroX']])

data_train['GyroX_Scaled'] = scaler.transform(data_train[['GyroX']])

scaler.fit(data_train[['GyroY']])

data_train['GyroY_Scaled'] = scaler.transform(data_train[['GyroY']])

scaler.fit(data_train[['GyroZ']])

data_train['GyroZ_Scaled'] = scaler.transform(data_train[['GyroZ']])
data_train
```

```
          AccX      AccY      AccZ     GyroX     GyroY     GyroZ  Class  \
0     0.000000  0.000000  0.000000  0.059407 -0.174707  0.101938      2
1    -1.624864 -1.082492 -0.204183 -0.028558  0.051313  0.135536      2
2    -0.594660 -0.122410  0.220502 -0.019395 -0.029322  0.087888      2
3     0.738478 -0.228456  0.667732  0.069791 -0.029932  0.054902      2
4     0.101741  0.777568 -0.066730  0.030696 -0.003665  0.054902      2
...        ...       ...       ...       ...       ...       ...    ...
3632  0.702303 -0.930822  0.809290  0.001985 -0.172264 -0.025733      1
3633  0.319258  0.272088  0.535243  0.018479  0.106901 -0.062385      1
3634  0.402702  0.432955 -0.683754 -0.090255 -0.085521  0.362167      1
3642 -0.349423  0.067261  0.394368 -0.132405  0.020159 -0.004963      1
3643 -0.402428  0.406218 -0.423009 -0.053603 -0.006720  0.001145      1

      Timestamp
0       3581629
1       3581630
2       3581630
3       3581631
4       3581631
...         ...
3632    3583786
3633    3583786
3634    3583787
3642    3583791
3643    3583791

[3347 rows x 8 columns]
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp	AccX_!
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	2	3581629	0.4
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	2	3581630	0.1
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	2	3581630	0.3
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	2	3581631	0.6
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	2	3581631	0.5
...	...	...	...	...	...	...	...	...	
3632	0.702303	-0.930822	0.809290	0.001985	-0.172264	-0.025733	1	3583786	0.6
3633	0.319258	0.272088	0.535243	0.018479	0.106901	-0.062385	1	3583786	0.5
3634	0.402702	0.432955	-0.683754	-0.090255	-0.085521	0.362167	1	3583787	0.5
3642	-0.349423	0.067261	0.394368	-0.132405	0.020159	-0.004963	1	3583791	0.4
3643	-0.402428	0.406218	-0.423009	-0.053603	-0.006720	0.001145	1	3583791	0.4

3347 rows × 14 columns

In [ ]:
```
# Initialize scaler
scaler = MinMaxScaler()

scaler.fit(data_test[['AccX']])

data_test['AccX_Scaled'] = scaler.transform(data_test[['AccX']])

scaler.fit(data_test[['AccY']])

data_test['AccY_Scaled'] = scaler.transform(data_test[['AccY']])

scaler.fit(data_test[['AccZ']])

data_test['AccZ_Scaled'] = scaler.transform(data_test[['AccZ']])

scaler.fit(data_test[['GyroX']])

data_test['GyroX_Scaled'] = scaler.transform(data_test[['GyroX']])

scaler.fit(data_test[['GyroY']])

data_test['GyroY_Scaled'] = scaler.transform(data_test[['GyroY']])

scaler.fit(data_test[['GyroZ']])

data_test['GyroZ_Scaled'] = scaler.transform(data_test[['GyroZ']])
data_test
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp	AccX_!
0	0.758194	-0.217791	0.457263	0.000000	0.000000	0.000000	3	818922	0.!
1	0.667560	-0.038610	0.231416	-0.054367	-0.007712	0.225257	3	818923	0.!
2	2.724449	-7.584121	2.390926	0.023824	0.013668	-0.038026	3	818923	0.7
3	2.330950	-7.621754	2.529024	0.056810	-0.180587	-0.052076	3	818924	0.6
4	2.847215	-6.755621	2.224640	-0.031765	-0.035201	0.035277	3	818924	0.7
...	...	...	...	...	...	...	...	...	
3079	-0.713858	-0.652975	-0.164015	-0.147829	-1.309466	0.517250	1	820706	0.3
3080	1.514261	0.330070	1.020714	1.321302	1.707598	-0.674548	1	820707	0.!
3081	1.280216	-1.735172	-2.332695	0.583376	0.690507	-0.468075	1	820707	0.!
3082	0.912313	0.583314	-0.965622	0.235794	0.512745	0.406073	1	820708	0.!
3083	1.462172	0.190287	0.019377	-0.254731	-0.279547	0.076205	1	820709	0.!

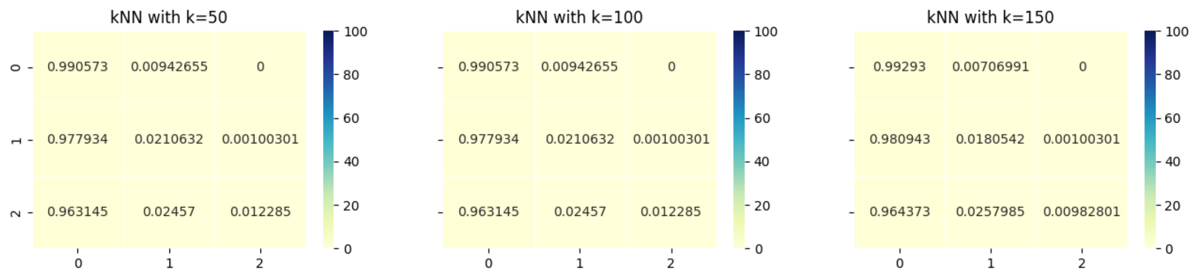
3084 rows × 14 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

Xtrain = data_train[['AccX_Scaled', 'AccY_Scaled', 'AccZ_Scaled', 'GyroX_Sca

ytrain = data_train['Class'].values

Xtest = data_test[['AccX_Scaled', 'AccY_Scaled', 'AccZ_Scaled', 'GyroX_Scale

ytest = data_test['Class'].values
```

KNN Learn

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

clf_kNN1ve    = KNeighborsClassifier(n_neighbors=50)
clf_kNN5ve    = KNeighborsClassifier(n_neighbors=100)
clf_kNN20ve    = KNeighborsClassifier(n_neighbors=150)

clf_kNN1ve.fit( Xtrain , ytrain )
clf_kNN5ve.fit( Xtrain , ytrain )
clf_kNN20ve.fit( Xtrain , ytrain )

ypred_kNN1ve  = clf_kNN1ve.predict(Xtest)
ypred_kNN5ve  = clf_kNN5ve.predict(Xtest)
ypred_kNN20ve  = clf_kNN20ve.predict(Xtest)


#print("Predictions in order")
#print(ypred_kNN1ve)
#print(ypred_kNN5ve)
#print(ypred_kNN20ve)

#print("\nReal data")
#print(ytest)

acc_kNN1ve    = 100*accuracy_score(ytest, ypred_kNN1ve)
acc_kNN5ve    = 100*accuracy_score(ytest, ypred_kNN5ve)
acc_kNN20ve    = 100*accuracy_score(ytest, ypred_kNN20ve)
print("\nAccuracy kNN with k=50:         {:.2f} %".format(acc_kNN1ve) )
print("Accuracy kNN with k=100:         {:.2f} %".format(acc_kNN5ve) )
print("Accuracy kNN with k=150:        {:.2f} %".format(acc_kNN20ve))

print('\nConfusion Matrices for every k used:')

CM_kNN1ve     = confusion_matrix(ytest, ypred_kNN1ve, normalize='true')
CM_kNN5ve     = confusion_matrix(ytest, ypred_kNN5ve, normalize='true')
CM_kNN20ve     = confusion_matrix(ytest, ypred_kNN20ve, normalize='true')

fig, axes = plt.subplots(1, 3, figsize=(16, 3), sharey=True)

sns.heatmap(CM_kNN5ve, annot=True, fmt='g', ax=axes[0], cmap="YlGnBu", linew
axes[0].set_title('kNN with k=50');

sns.heatmap(CM_kNN5ve, annot=True, fmt='g', ax=axes[1], cmap="YlGnBu", linew
axes[1].set_title('kNN with k=100');

sns.heatmap(CM_kNN20ve, annot=True, fmt='g', ax=axes[2], cmap="YlGnBu", line
axes[2].set_title('kNN with k=150');

plt.show()
```

```
Accuracy kNN with k=50:         41.96 %
Accuracy kNN with k=100:         41.89 %
Accuracy kNN with k=150:        41.83 %

Confusion Matrices for every k used:
```



Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(random_state=0)
logistic.fit(Xtrain, ytrain)
y_predLR = logistic.predict(Xtest)
#print("Prediction")
#print(y_predLR)
#print("\nReal data")
#print(ytest)
acc = 100*accuracy_score(y_predLR, ytest)
print('Accuracy: ', acc)
```

```
Accuracy:  35.27885862516213
```

Multi-Layer Perceptron Classifier

In [ ]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix

data_train=pd.read_csv('./train_motion_data.csv')
data_test=pd.read_csv('./test_motion_data.csv')

data_train
```

Out[ ]:

	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Class	Timestamp
0	0.000000	0.000000	0.000000	0.059407	-0.174707	0.101938	NORMAL	3581629
1	-1.624864	-1.082492	-0.204183	-0.028558	0.051313	0.135536	NORMAL	3581630
2	-0.594660	-0.122410	0.220502	-0.019395	-0.029322	0.087888	NORMAL	3581630
3	0.738478	-0.228456	0.667732	0.069791	-0.029932	0.054902	NORMAL	3581631
4	0.101741	0.777568	-0.066730	0.030696	-0.003665	0.054902	NORMAL	3581631
...	...	...	...	...	...	...	...	...
3639	0.915688	-2.017489	1.687505	0.450360	0.384845	-1.236468	SLOW	3583789
3640	-1.934203	0.914925	-0.096013	0.321468	0.649350	-0.477162	SLOW	3583790
3641	-0.222845	0.747304	-0.887430	0.361174	-0.406836	0.054291	SLOW	3583790
3642	-0.349423	0.067261	0.394368	-0.132405	0.020159	-0.004963	SLOW	3583791
3643	-0.402428	0.406218	-0.423009	-0.053603	-0.006720	0.001145	SLOW	3583791

3644 rows × 8 columns

In [ ]:

```
print((data_train.Class =='NORMAL').count())
print((data_train.Class =='AGGRESSIVE').count())
print((data_train.Class =='SLOW').count())
print((data_test.Class =='NORMAL').count())
print((data_test.Class =='AGGRESSIVE').count())
print((data_test.Class =='SLOW').count())
```

```
3644
3644
3644
3084
3084
3084
```

In [ ]:

```
y=data_train['Class']
X=data_train.drop(['Class','Timestamp','AccZ','GyroZ'], axis=1)
#X=data_train.drop(['Class'], axis=1)
y_test=data_test['Class']
X_test=data_test.drop(['Class','Timestamp','AccZ','GyroZ'], axis=1)
#X_test=data_test.drop(['Class'], axis=1)
sc=StandardScaler()
scaler = sc.fit(X)
#X = scaler.transform(X)
#X_test = scaler.transform(X_test)
#X
```

In [ ]:

```
clf = MLPClassifier(hidden_layer_sizes=(4,256,3),
                        max_iter =40,activation = 'relu',
                        solver = 'adam')
clf.fit(X, y)
```

```
/home/david586/.local/lib/python3.10/site-packages/sklearn/neural_network/_
multilayer_perceptron.py:702: ConvergenceWarning: Stochastic Optimizer: Max
imum iterations (40) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[ ]:

```
▾ MLPClassifier
MLPClassifier(hidden_layer_sizes=(4, 256, 3), max_iter=40)
```

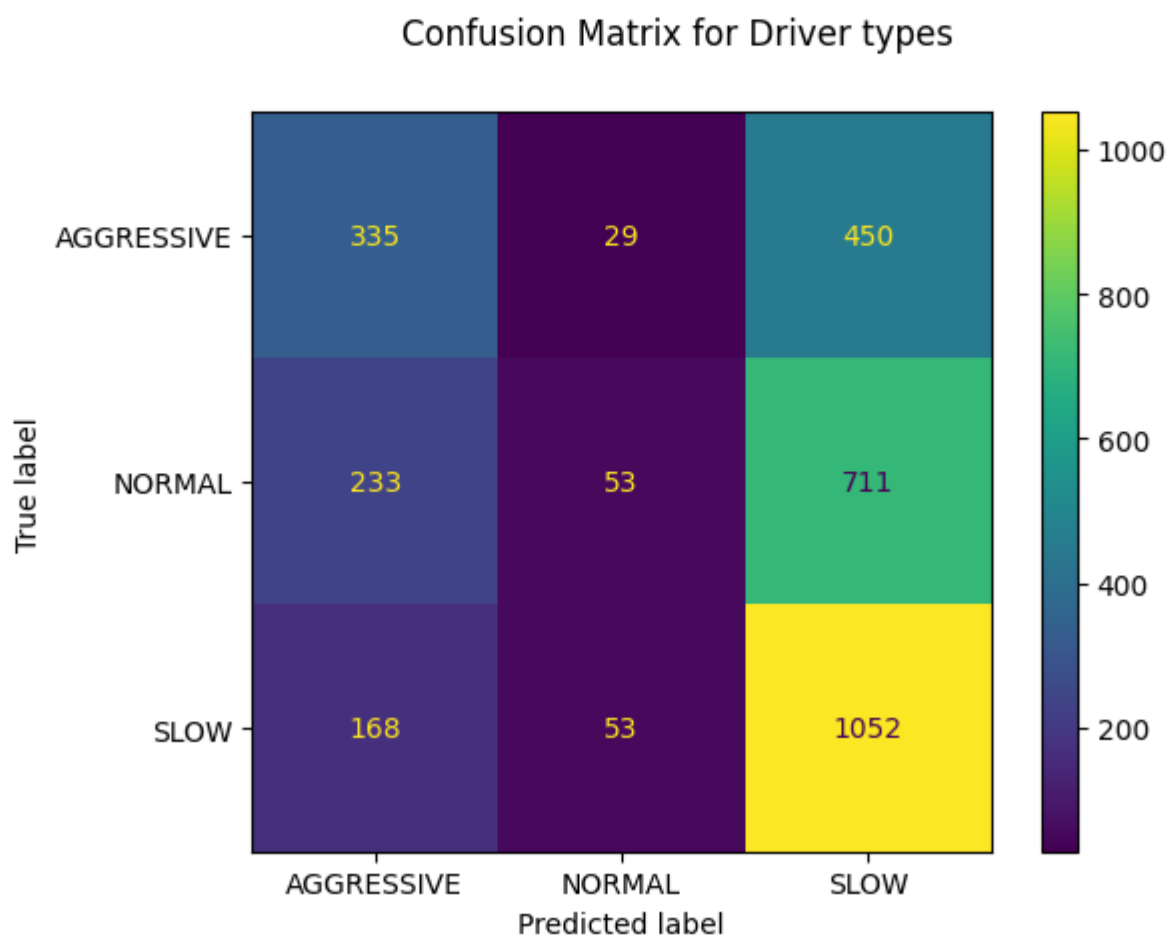
In [ ]:

```
y_pred = clf.predict(X_test)
```

In [ ]:

```
fig = plot_confusion_matrix(clf, X_test, y_test, display_labels=clf.classes_
fig.figure_.suptitle("Confusion Matrix for Driver types")
plt.show()
```

```
/home/david586/.local/lib/python3.10/site-packages/sklearn/utils/deprecatio
n.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Funct
ion `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.
2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [ ]:

```
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.47
```

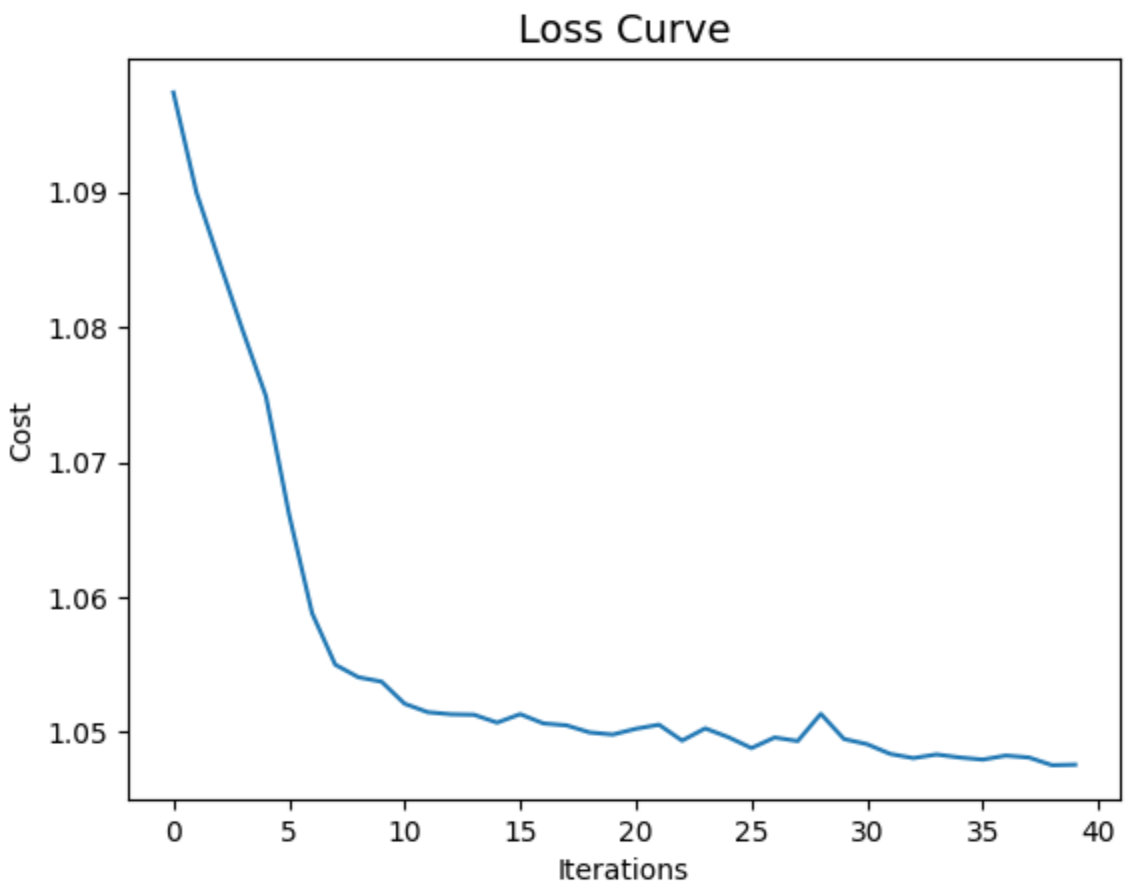
In [ ]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

  AGGRESSIVE       0.46      0.41      0.43       814
      NORMAL       0.39      0.05      0.09       997
        SLOW       0.48      0.83      0.60      1273

    accuracy                           0.47      3084
   macro avg       0.44      0.43      0.38      3084
weighted avg       0.44      0.47      0.39      3084
```

In [ ]:

```
plt.plot(clf.loss_curve_)
plt.title("Loss Curve", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```



In [ ]:

```
confusion_matrix(y_test, y_pred)
```

Out[ ]:

```
array([[ 335,   29,  450],
       [ 233,   53,  711],
       [ 168,   53, 1052]])
```