```
#!pip install ucimlrepo
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

from sklearn.preprocessing import StandardScaler
from IPython.display import display
```

Smartphone-Based Recognition of Human Activities and Postural Transitions

2. Dimensions and first 5 lines

```
DATA_DIR = "/Users/josueh/Desktop/Fusión de datos con Python/notebook

labels = pd.read_csv(os.path.join(DATA_DIR, "labels.txt"),
                     sep="\\s+",
                     header=None,
                     names=["exp_id", "user_id", "activity_id", "star

activity_map = {
    1: "WALKING",
    2: "WALKING_UPSTAIRS",
    3: "WALKING_DOWNSTAIRS",
    4: "SITTING",
    5: "STANDING",
    6: "LAYING",
    7: "STAND_TO_SIT",
    8: "SIT_TO_STAND",
    9: "SIT_TO_LIE",
    10: "LIE_TO_SIT",
    11: "STAND_TO_LIE",
    12: "LIE_TO_STAND"
}
labels["Activity"] = labels["activity_id"].map(activity_map)
```

```
merged_data = []

for _, row in labels.iterrows():
    exp = row["exp_id"]
    user = row["user_id"]
    start = row["start"]
    end = row["end"]

    acc_path = os.path.join(DATA_DIR, f"acc_exp{exp:02d}_user{user:02
    gyro_path = os.path.join(DATA_DIR, f"gyro_exp{exp:02d}_user{user:

    acc = pd.read_csv(acc_path, sep="\\s+", header=None, names=["X ax
    gyro = pd.read_csv(gyro_path, sep="\\s+", header=None, names=["X

    acc_slice = acc.iloc[start-1:end].reset_index(drop=True)
    gyro_slice = gyro.iloc[start-1:end].reset_index(drop=True)

    combined = pd.concat([acc_slice, gyro_slice], axis=1)
    combined["Activity"] = row["Activity"]
    combined["User"] = user
    combined["Experiment"] = exp

    merged_data.append(combined)

# Combine everything into one big DataFrame
full_dataset = pd.concat(merged_data, ignore_index=True)

full_dataset = full_dataset.drop(['User', 'Experiment'], axis=1)
```

```
display(full_dataset)
print(f"Dimensions: {full_dataset.shape}\n")
print(full_dataset["Activity"].value_counts())
```

	X axis_acc	Y axis_acc	Z axis_acc	X axis_gyro	Y axis_gyro	Z axis_gyro	
0	1.020833	-0.125000	0.104167	-0.000916	0.001833	0.002749	
1	1.020833	-0.125000	0.105556	-0.002749	-0.004276	0.002749	
2	1.025000	-0.125000	0.101389	-0.000305	-0.002138	0.006109	
3	1.020833	-0.125000	0.104167	0.012217	0.000916	-0.007330	
4	1.016667	-0.125000	0.108333	0.011301	-0.001833	-0.006414	
...	...	...	...	...	...	...	
815609	0.950000	-0.443056	-0.187500	1.184773	1.112386	-0.307876	W
815610	0.880556	-0.390278	-0.156944	1.163698	1.106277	-0.374155	W
815611	0.834722	-0.358333	-0.098611	1.177137	1.023810	-0.388816	W
815612	0.802778	-0.329167	-0.104167	1.213484	0.918130	-0.332311	W
815613	0.770833	-0.287500	-0.098611	1.326188	0.846659	-0.202502	W

815614 rows × 7 columns

```
Dimensions: (815614, 7)

Activity
STANDING              138105
LAYING                136865
SITTING               126677
WALKING               122091
WALKING_UPSTAIRS      116707
WALKING_DOWNSTAIRS    107961
STAND_TO_LIE           14418
SIT_TO_LIE             12428
LIE_TO_SIT             11150
LIE_TO_STAND           10867
STAND_TO_SIT           10316
SIT TO STAND            8029
```

3. Handling missing data

```
print(f"Na values for sensors data: {full_dataset.isna().values.any()}
print(f"Null values for sensors data: {full_dataset.isnull().values.a
```

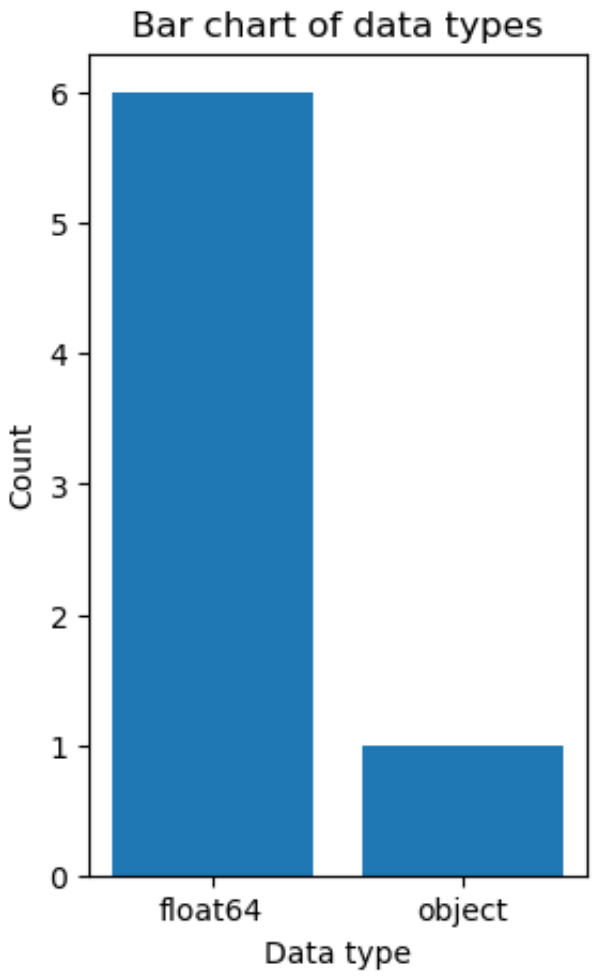
```
Na values for sensors data: False
Null values for sensors data: False
```

4. Data types per attribute

```
display(full_dataset.dtypes)

print("\n")

display(full_dataset.info())

print("\n")

dtype_counts = full_dataset.dtypes.value_counts()

plt.figure(figsize=(3, 5))
plt.bar(dtype_counts.index.astype(str), dtype_counts.values)

plt.xlabel("Data type")
plt.ylabel("Count")
plt.title("Bar chart of data types")
plt.show()
```

```
X axis_acc     float64
Y axis_acc     float64
Z axis_acc     float64
X axis_gyro    float64
Y axis_gyro    float64
Z axis_gyro    float64
Activity        object
dtype: object


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 815614 entries, 0 to 815613
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
```

```
 0   X axis_acc   815614 non-null  float64
 1   Y axis_acc   815614 non-null  float64
 2   Z axis_acc   815614 non-null  float64
 3   X axis_gyro  815614 non-null  float64
 4   Y axis_gyro  815614 non-null  float64
 5   Z axis_gyro  815614 non-null  float64
 6   Activity     815614 non-null  object
dtypes: float64(6), object(1)
memory usage: 43.6+ MB
None
```



5. Statistical summaries

5.1. For all attributes

```
display(full_dataset.describe())
```

```
# Remove last column of strings and compute skewness
skewness = full_dataset.iloc[:, :-1].skew()
kurtosis = full_dataset.iloc[:, :-1].kurt()



print("Skewness: \n")
display(skewness)

print("\nKurtosis: \n")
display(kurtosis)
```

	X axis_acc	Y axis_acc	Z axis_acc	X axis_gyro	Y axis_gyr
count	815614.000000	815614.000000	815614.000000	815614.000000	815614.00000
mean	0.806468	0.032947	0.087695	0.009292	-0.00356
std	0.409161	0.405570	0.345823	0.440613	0.41100
min	-0.647222	-1.698611	-1.837500	-5.249165	-6.32337
25%	0.675000	-0.223611	-0.119444	-0.048258	-0.07391
50%	0.950000	-0.072222	0.045833	0.007025	0.00061
75%	1.019445	0.213889	0.238889	0.077885	0.04795
max	2.004167	1.716667	1.502778	5.964488	6.24029

```
Skewness:

X axis_acc    -0.707468
Y axis_acc     0.731837
Z axis_acc     0.390440
X axis_gyro   -0.204585
Y axis_gyro    0.393263
Z axis_gyro   -0.202577
dtype: float64

Kurtosis:

X axis_acc     0.256857
Y axis_acc     0.036421
Z axis_acc     0.542543
X axis_gyro    8.599942
Y axis_gyro    8.991414
```

```
Z axis_gyro    6.135584
dtype: float64
```

5.2. For all activities

```
full_dataset.groupby("Activity").describe()
```

	X axis_acc					
	count	mean	std	min	25%	50%
Activity						
LAYING	136865.0	0.068684	0.134152	-0.523611	-0.012500	0.073
LIE_TO_SIT	11150.0	0.672231	0.371905	-0.472222	0.379514	0.815
LIE_TO_STAND	10867.0	0.664515	0.391104	-0.647222	0.291667	0.801
SITTING	126677.0	0.957249	0.096331	0.426389	0.937500	0.993
SIT_TO_LIE	12428.0	0.569750	0.375277	-0.497222	0.227778	0.650
SIT_TO_STAND	8029.0	0.943345	0.129987	0.376389	0.869444	0.96
STANDING	138105.0	1.001342	0.029063	0.605556	0.990278	1.009
STAND_TO_LIE	14418.0	0.637869	0.383902	-0.600000	0.315278	0.759
STAND_TO_SIT	10316.0	0.941518	0.132565	-0.304167	0.883333	0.959
WALKING	122091.0	0.997615	0.236304	-0.148611	0.825000	0.968
WALKING_DOWNSTAIRS	107961.0	0.996372	0.375211	-0.108333	0.733333	0.894
WALKING_UPSTAIRS	116707.0	0.952509	0.262548	-0.002778	0.766667	0.919

12 rows × 48 columns

5.3. Grouped by activity (accelerometer)

```
full_dataset.iloc[:, :3].groupby(full_dataset["Activity"]).describe()
```

	X axis_acc					
	count	mean	std	min	25%	50%
Activity						
LAYING	136865.0	0.068684	0.134152	-0.523611	-0.012500	0.073
LIE_TO_SIT	11150.0	0.672231	0.371905	-0.472222	0.379514	0.815
LIE_TO_STAND	10867.0	0.664515	0.391104	-0.647222	0.291667	0.801
SITTING	126677.0	0.957249	0.096331	0.426389	0.937500	0.993
SIT_TO_LIE	12428.0	0.569750	0.375277	-0.497222	0.227778	0.650
SIT_TO_STAND	8029.0	0.943345	0.129987	0.376389	0.869444	0.96
STANDING	138105.0	1.001342	0.029063	0.605556	0.990278	1.009
STAND_TO_LIE	14418.0	0.637869	0.383902	-0.600000	0.315278	0.759
STAND_TO_SIT	10316.0	0.941518	0.132565	-0.304167	0.883333	0.959
WALKING	122091.0	0.997615	0.236304	-0.148611	0.825000	0.968
WALKING_DOWNSTAIRS	107961.0	0.996372	0.375211	-0.108333	0.733333	0.894
WALKING_UPSTAIRS	116707.0	0.952509	0.262548	-0.002778	0.766667	0.919

12 rows × 24 columns

5.4. Grouped by activity (gyroscope)

```
full_dataset.iloc[:, 3:].groupby(full_dataset["Activity"]).describe()
```

	X axis_gyro					
	count	mean	std	min	25%	50%
Activity						
LAYING	136865.0	0.006827	0.077528	-3.524082	-0.001527	0.0(
LIE_TO_SIT	11150.0	0.259984	0.522768	-2.172542	-0.042455	0.1
LIE_TO_STAND	10867.0	0.217037	0.614861	-3.190244	-0.115148	0.1
SITTING	126677.0	0.006761	0.042528	-1.255633	0.001833	0.0(
SIT_TO_LIE	12428.0	-0.264125	0.542471	-3.396716	-0.548862	-0.1(
SIT_TO_STAND	8029.0	-0.030347	0.224270	-1.643838	-0.149051	-0.0
STANDING	138105.0	0.015178	0.192753	-3.741550	-0.008858	0.0(
STAND_TO_LIE	14418.0	-0.028896	0.704970	-3.198185	-0.414090	-0.0
STAND_TO_SIT	10316.0	0.019864	0.379952	-2.462092	-0.123089	0.0
WALKING	122091.0	0.002748	0.514968	-4.473366	-0.331700	0.0
WALKING_DOWNSTAIRS	107961.0	-0.265104	0.678136	-5.249165	-0.611782	-0.1
WALKING_UPSTAIRS	116707.0	0.260973	0.528513	-4.232685	-0.081551	0.1

12 rows × 24 columns

5.5. Grouped by activity (X axis both sensors)

```
full_dataset[["X axis_acc", "X axis_gyro", "Activity"]].groupby("Activ
```

	X axis_acc					
	count	mean	std	min	25%	50%
Activity						
LAYING	136865.0	0.068684	0.134152	-0.523611	-0.012500	0.073
LIE_TO_SIT	11150.0	0.672231	0.371905	-0.472222	0.379514	0.815
LIE_TO_STAND	10867.0	0.664515	0.391104	-0.647222	0.291667	0.801
SITTING	126677.0	0.957249	0.096331	0.426389	0.937500	0.993
SIT_TO_LIE	12428.0	0.569750	0.375277	-0.497222	0.227778	0.650
SIT_TO_STAND	8029.0	0.943345	0.129987	0.376389	0.869444	0.96
STANDING	138105.0	1.001342	0.029063	0.605556	0.990278	1.009
STAND_TO_LIE	14418.0	0.637869	0.383902	-0.600000	0.315278	0.759
STAND_TO_SIT	10316.0	0.941518	0.132565	-0.304167	0.883333	0.959
WALKING	122091.0	0.997615	0.236304	-0.148611	0.825000	0.968
WALKING_DOWNSTAIRS	107961.0	0.996372	0.375211	-0.108333	0.733333	0.894
WALKING_UPSTAIRS	116707.0	0.952509	0.262548	-0.002778	0.766667	0.919

5.6. Group by activity (Y axis both sensors)

```
full_dataset[["Y axis_acc", "Y axis_gyro", "Activity"]].groupby("Acti
```

	Y axis_acc					
	count	mean	std	min	25%	50%
Activity						
LAYING	136865.0	0.637964	0.378166	-1.525000	0.537500	0.75
LIE_TO_SIT	11150.0	0.352802	0.373851	-1.163889	0.058333	0.33
LIE_TO_STAND	10867.0	0.333360	0.426074	-1.361111	-0.011111	0.3
SITTING	126677.0	0.128840	0.198695	-0.605556	-0.009722	0.1(
SIT_TO_LIE	12428.0	0.484444	0.374266	-1.484722	0.233333	0.53
SIT_TO_STAND	8029.0	-0.138365	0.275911	-0.880556	-0.325000	-0.19
STANDING	138105.0	-0.166308	0.113910	-0.862500	-0.233333	-0.16
STAND_TO_LIE	14418.0	0.323216	0.431206	-1.356944	-0.008333	0.38
STAND_TO_SIT	10316.0	-0.110421	0.295371	-1.088889	-0.327778	-0.14
WALKING	122091.0	-0.179961	0.203191	-1.626389	-0.288889	-0.15
WALKING_DOWNSTAIRS	107961.0	-0.152532	0.217309	-1.698611	-0.255556	-0.12
WALKING_UPSTAIRS	116707.0	-0.268568	0.212435	-1.613889	-0.390278	-0.25

5.7. Group by activity (Z axis both sensors)

```
full_dataset[["Z axis_acc", "Z axis_gyro", "Activity"]].groupby("Acti
```

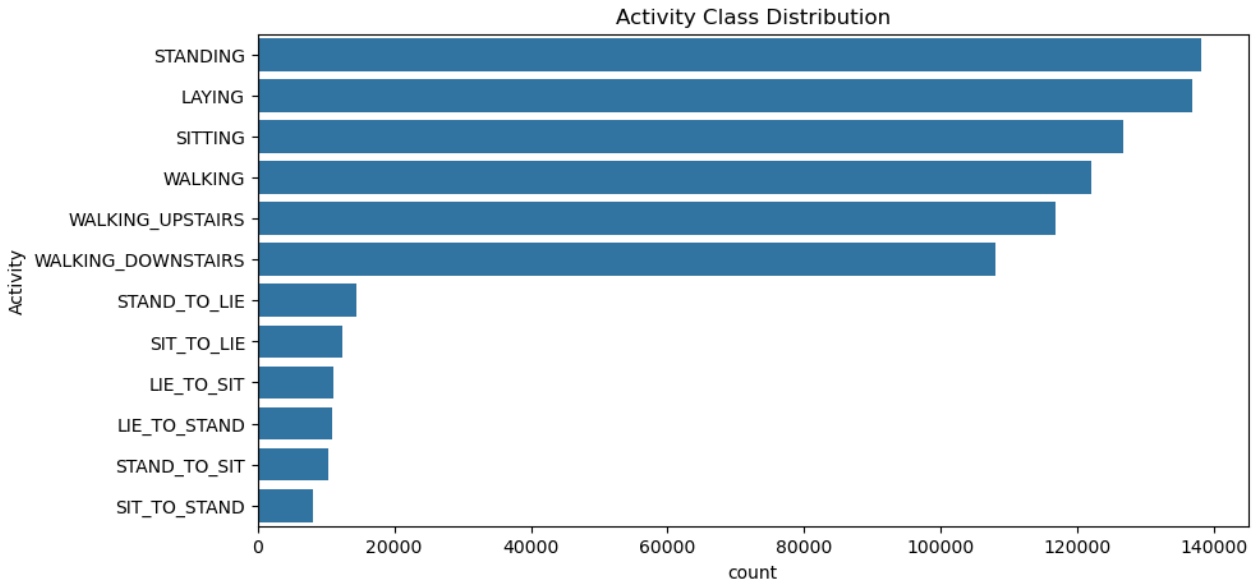
	Z axis_acc					
	count	mean	std	min	25%	50%
Activity						
LAYING	136865.0	0.541338	0.386362	-1.051389	0.397222	0.6
LIE_TO_SIT	11150.0	0.313043	0.294967	-0.601389	0.112500	0.3
LIE_TO_STAND	10867.0	0.165808	0.376300	-1.441667	-0.077778	0.1
SITTING	126677.0	0.146163	0.201699	-0.518056	0.002778	0.1
SIT_TO_LIE	12428.0	0.241071	0.373067	-1.533333	-0.012500	0.2
SIT_TO_STAND	8029.0	-0.016245	0.276693	-1.040278	-0.202778	0.0
STANDING	138105.0	-0.020766	0.152715	-0.975000	-0.113889	0.0
STAND_TO_LIE	14418.0	0.267781	0.368528	-0.956944	0.002778	0.2
STAND_TO_SIT	10316.0	0.026157	0.270360	-0.947222	-0.122222	0.0
WALKING	122091.0	-0.059898	0.210327	-1.236111	-0.177778	-0.0
WALKING_DOWNSTAIRS	107961.0	-0.054831	0.225922	-1.837500	-0.166667	-0.0
WALKING_UPSTAIRS	116707.0	-0.147967	0.248122	-1.587500	-0.295833	-0.1

6. Class distribution

```
full_dataset['Activity'].value_counts()
```

```
Activity
STANDING             138105
LAYING               136865
SITTING              126677
WALKING              122091
WALKING_UPSTAIRS     116707
WALKING_DOWNSTAIRS   107961
STAND_TO_LIE          14418
SIT_TO_LIE            12428
LIE_TO_SIT            11150
LIE_TO_STAND          10867
STAND_TO_SIT          10316
SIT_TO_STAND           8029
Name: count, dtype: int64
```

```
plt.figure(figsize=(10,5))
sns.countplot(y=full_dataset["Activity"], order=full_dataset["Activity
plt.title("Activity Class Distribution")
plt.show()
```



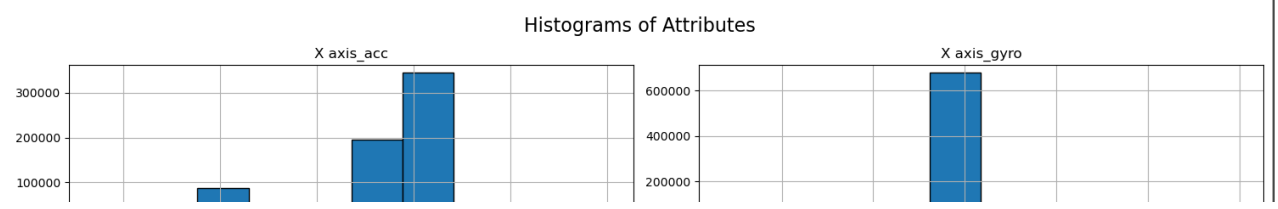
7. Correlation between attributes

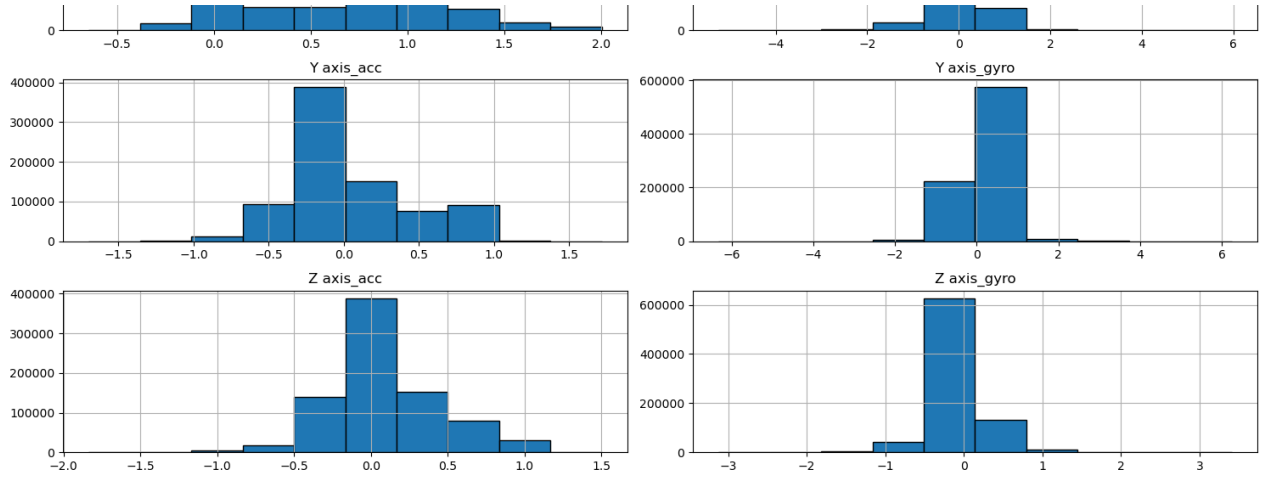
```
correlation_dataframe = full_dataset.drop(columns=["Activity"]).corr(

display(correlation_dataframe)
```

	X axis_acc	Y axis_acc	Z axis_acc	X axis_gyro	Y axis_gyro	Z axis_gyro
X axis_acc	1.000000	-0.676337	-0.582186	0.009182	0.040651	-0.011987
Y axis_acc	-0.676337	1.000000	0.558419	-0.054605	-0.002549	0.048439
Z axis_acc	-0.582186	0.558419	1.000000	-0.025221	-0.022633	0.025245
X axis_gyro	0.009182	-0.054605	-0.025221	1.000000	-0.164666	-0.026436

8. Histograms

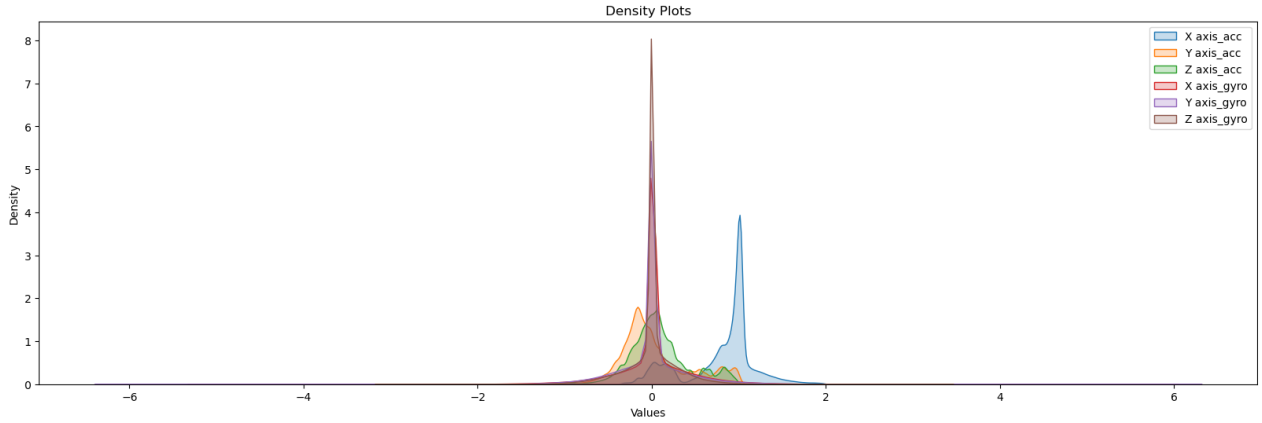
```
columns_left = full_dataset.columns[:3]
columns_right = full_dataset.columns[3:6]

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 8))

for i, col in enumerate(columns_left):
    full_dataset[col].hist(ax=axes[i, 0], bins=10, edgecolor='black')
    axes[i, 0].set_title(col)

for i, col in enumerate(columns_right):
    full_dataset[col].hist(ax=axes[i, 1], bins=10, edgecolor='black')
    axes[i, 1].set_title(col)


plt.suptitle("Histograms of Attributes", fontsize=16)
plt.tight_layout(w_pad=1, h_pad=1)
plt.show()
```





9. Density Plots

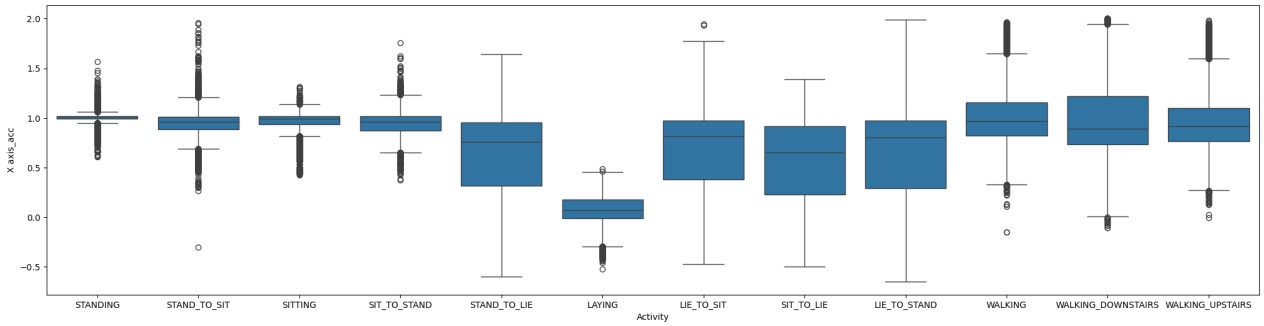
```
plt.figure(figsize=(20, 6))
for col in full_dataset.select_dtypes(include=np.number):
    sns.kdeplot(full_dataset[col], fill=True, label=col)
plt.legend()
plt.title("Density Plots")
plt.xlabel("Values")
plt.show()
```



10. Boxplots

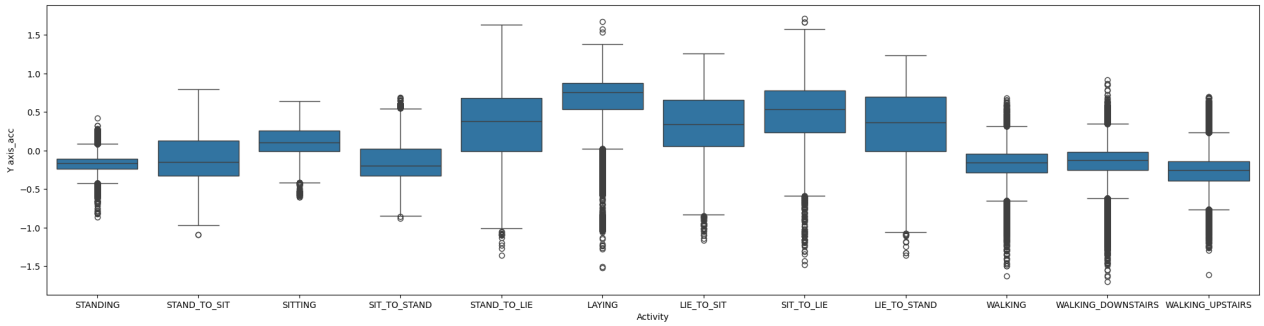
Accelerometer data on X axis per activity

```
plt.figure(figsize=(25, 6))
sns.boxplot(x="Activity", y="X axis_acc", data=full_dataset)
plt.show()
```



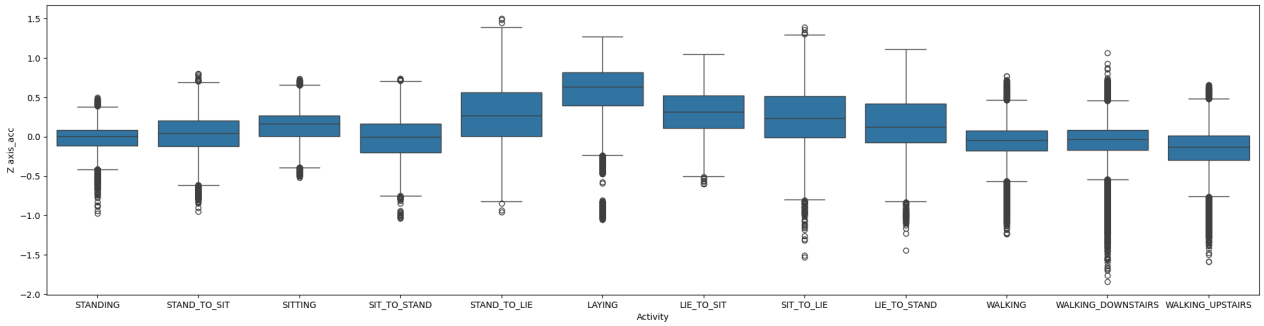
Accelerometer data on Y axis per activity

```
plt.figure(figsize=(25, 6))
sns.boxplot(x="Activity", y="Y axis_acc", data=full_dataset)
plt.show()
```



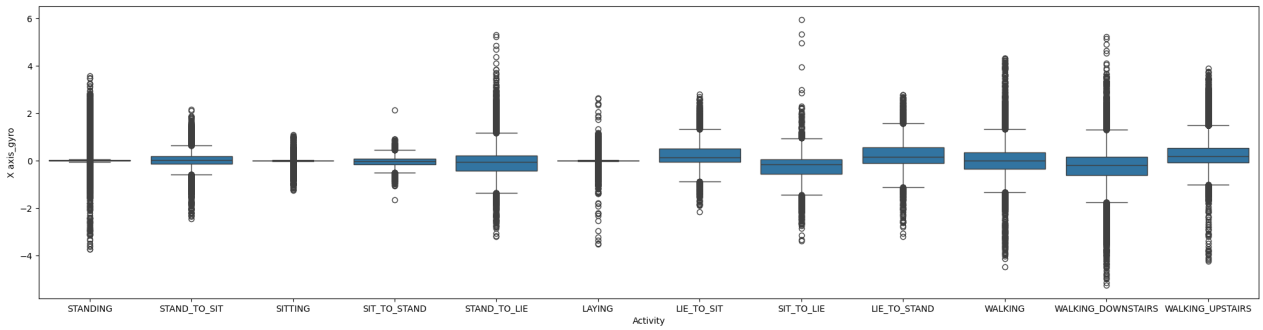
Accelerometer data on Z axis per activity

```
plt.figure(figsize=(25, 6))
sns.boxplot(x="Activity", y="Z axis_acc", data=full_dataset)
plt.show()
```



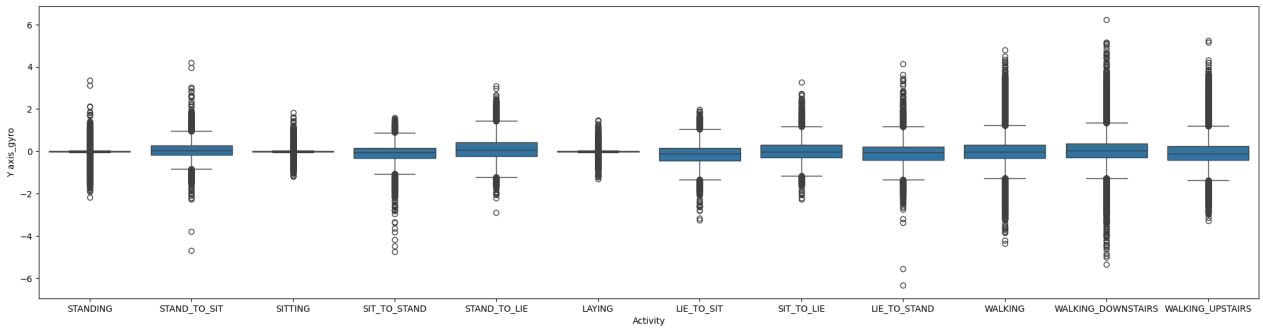
Gyroscope data on X axis per activity

```
plt.figure(figsize=(25, 6))
sns.boxplot(x="Activity", y="X axis_gyro", data=full_dataset)
plt.show()
```



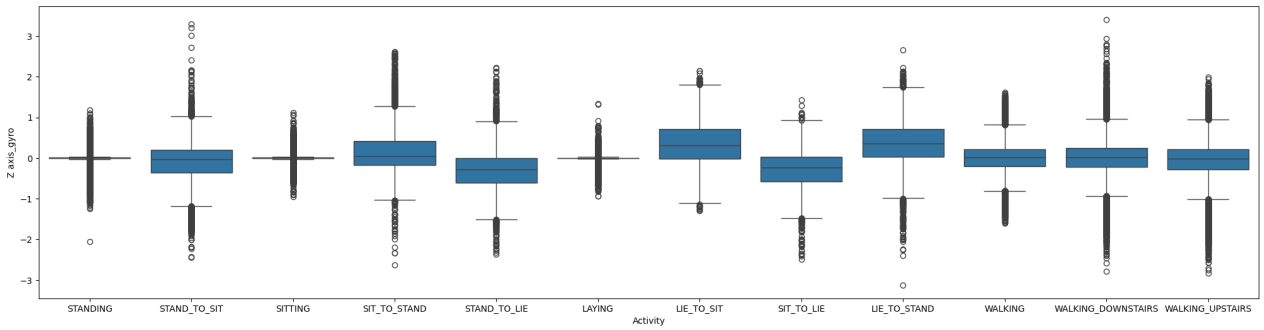
Gyroscope data on Y axis per activity

```
plt.figure(figsize=(25, 6))
sns.boxplot(x="Activity", y="Y axis_gyro", data=full_dataset)
plt.show()
```



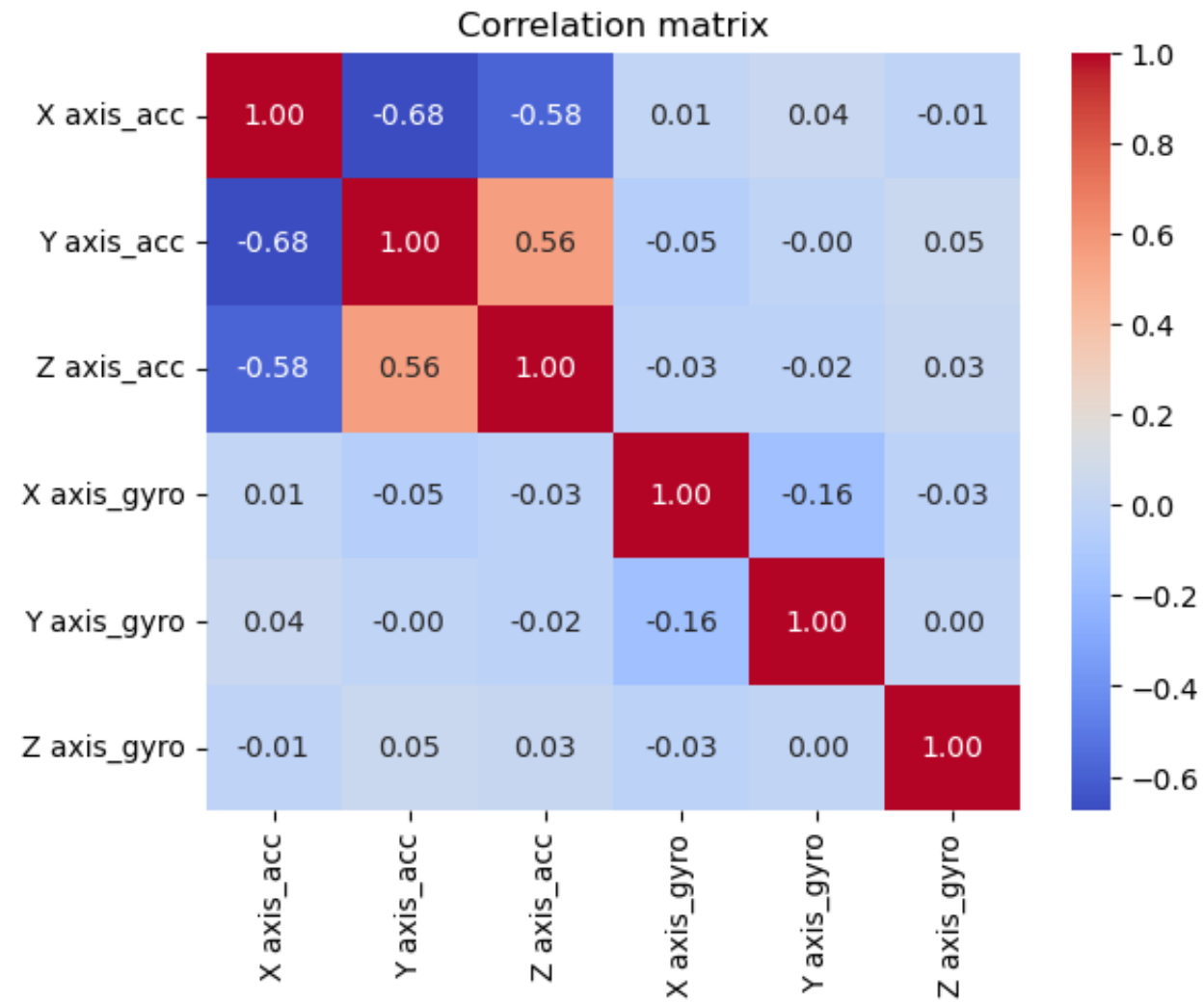
Gyroscope data on Z axis per activity

```
plt.figure(figsize=(25, 6))
sns.boxplot(x="Activity", y="Z axis_gyro", data=full_dataset)
plt.show()
```



11. Correlation matrix

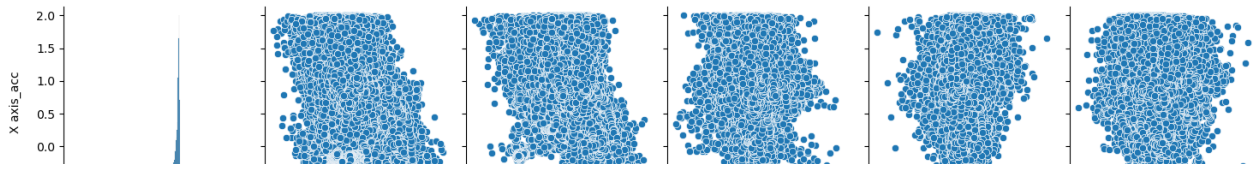
```
correlation_dataframe = full_dataset.drop(columns=["Activity"]).corr(

sns.heatmap(correlation_dataframe, annot=True, cmap='coolwarm', fmt="
plt.title("Correlation matrix")
plt.show()
```

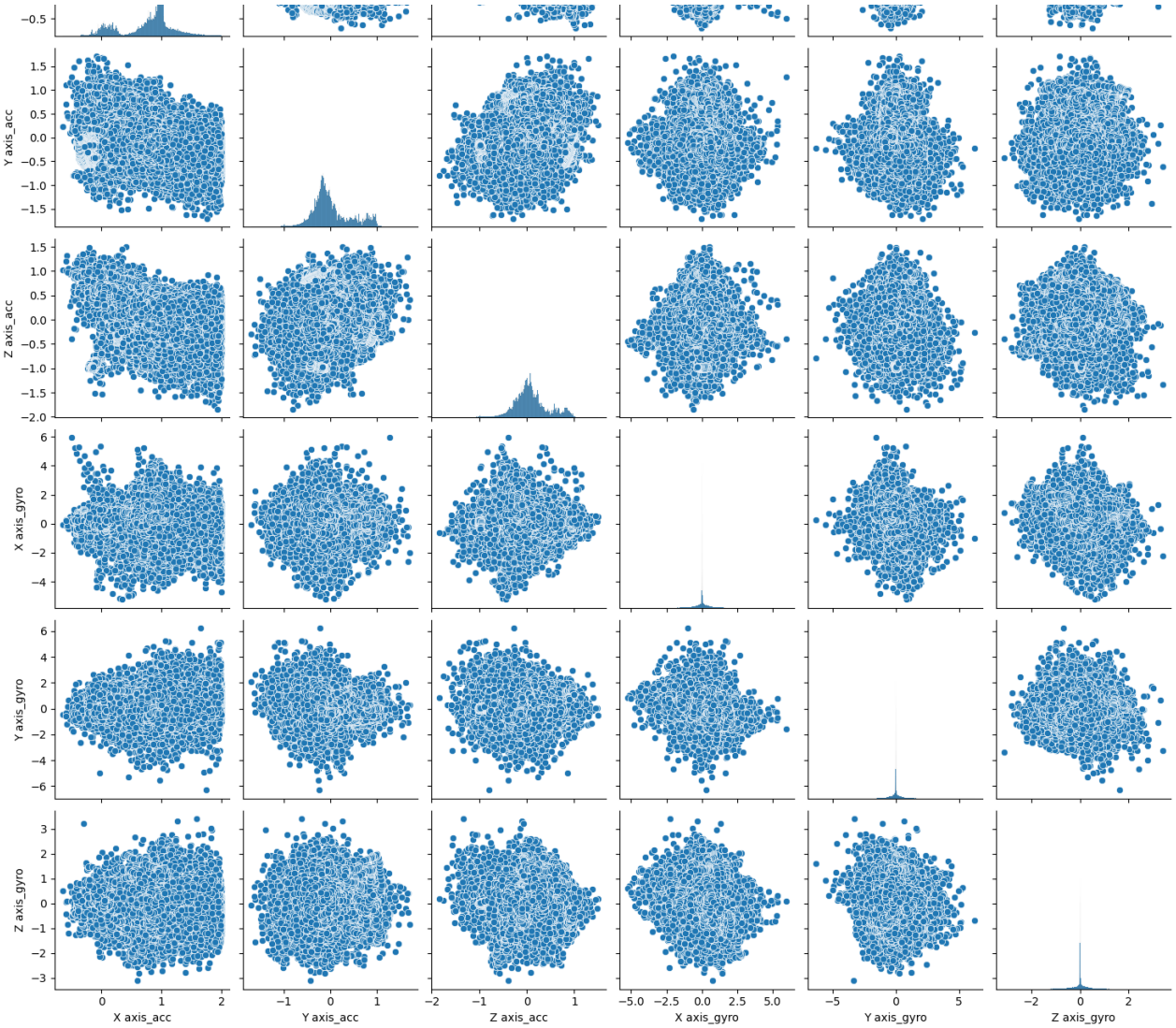


12. Scatter matrix

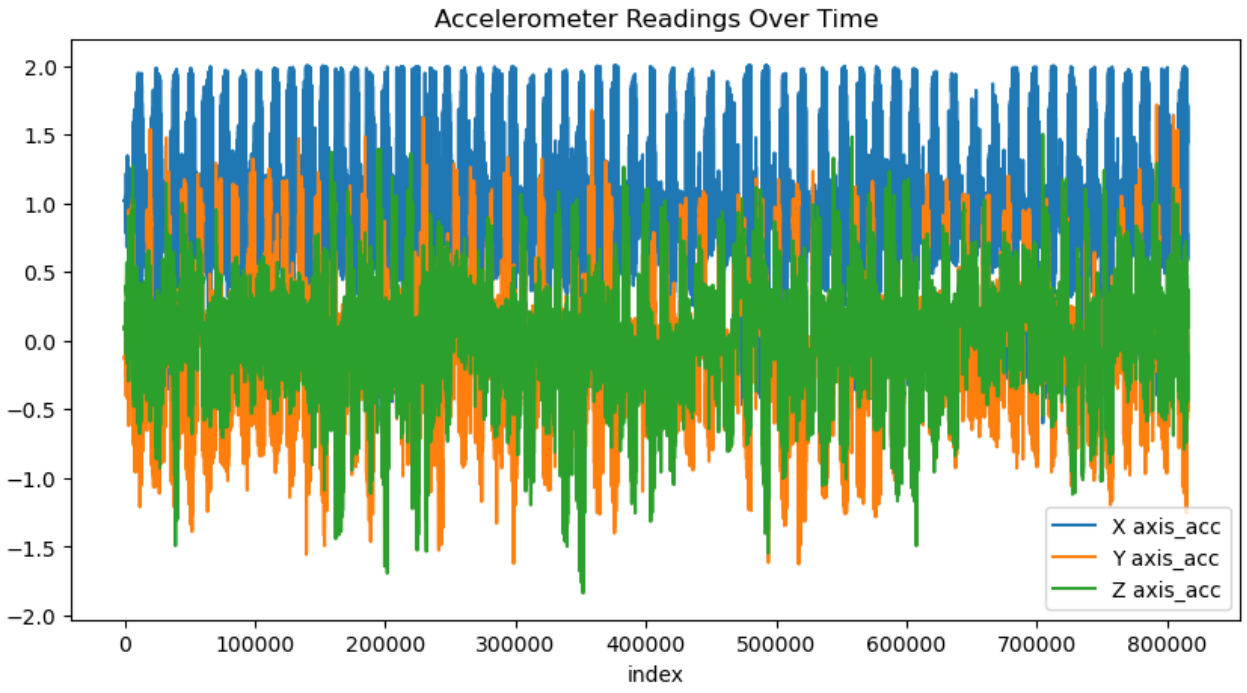
```
sns.pairplot(full_dataset)
```

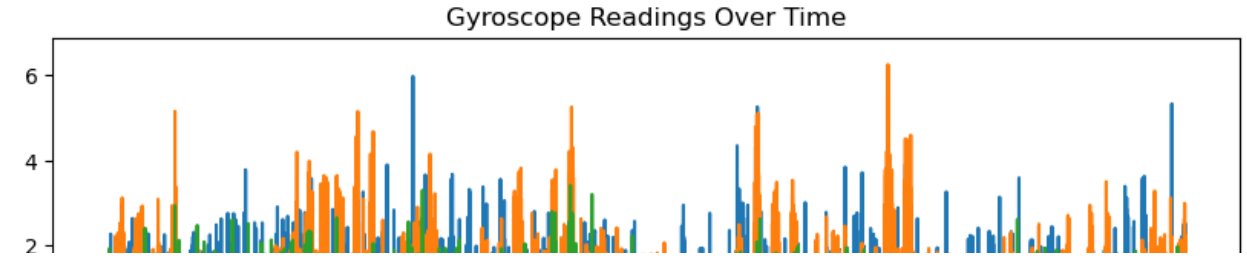
```
<seaborn.axisgrid.PairGrid at 0x141ffa9d0>
```

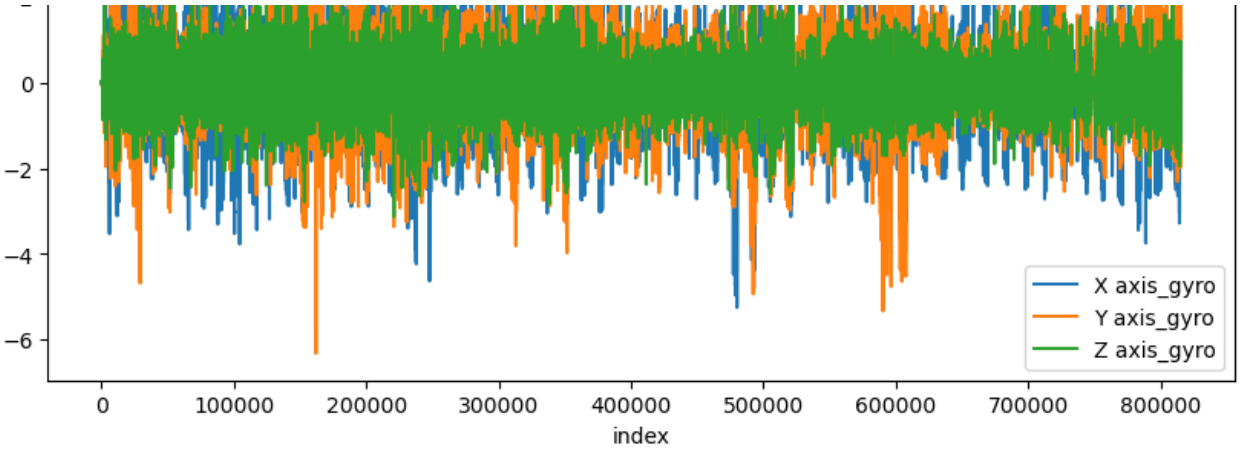



Y axis_acc
Z axis_acc
X axis_gyro
Y axis_gyro
Z axis_gyro
X axis_acc
Y axis_acc
Z axis_acc
X axis_gyro
Y axis_gyro
Z axis_gyro

```
full_dataset.reset_index().plot(x="index", y=["X axis_acc", "Y axis_a
full_dataset.reset_index().plot(x="index", y=["X axis_gyro", "Y axis_
plt.show()
```







13. Feature extraction

```
sampling_rate = 50
time_span = 3

window_size = sampling_rate * time_span
step_size = 150  # No overlap

def extract_features(window):
    features = {}

    acc_magnitude = np.sqrt(window['X axis_acc']**2 + window['Y axis_a
    gyro_magnitude = np.sqrt(window['X axis_gyro']**2 + window['Y axis

    for col in window.columns:
        features[f'{col}_mean'] = window[col].mean()
```

```
        for col in window.columns:
            features[f'{col}_std'] = window[col].std()

        for col in window.columns:
            features[f'{col}_max'] = window[col].max()

        features['acc_xy_corr'] = window[['X axis_acc', 'Y axis_acc']].co
        features['acc_xz_corr'] = window[['X axis_acc', 'Z axis_acc']].co
        features['acc_yz_corr'] = window[['Y axis_acc', 'Z axis_acc']].co

        features['gyro_xy_corr'] = window[['X axis_gyro', 'Y axis_gyro']]
        features['gyro_xz_corr'] = window[['X axis_gyro', 'Z axis_gyro']]
        features['gyro_yz_corr'] = window[['Y axis_gyro', 'Z axis_gyro']]

        features['acc_magnitude_mean'] = acc_magnitude.mean()
        features['gyro_magnitude_mean'] = gyro_magnitude.mean()

        features['acc_magnitude_std'] = acc_magnitude.std()
        features['gyro_magnitude_std'] = gyro_magnitude.std()

        features['acc_magnitude_auc'] = acc_magnitude.sum()
        features['gyro_magnitude_auc'] = gyro_magnitude.sum()

        features['acc_magnitude_meandif'] = np.mean(np.abs(np.diff(acc_ma
        features['gyro_magnitude_meandif'] = np.mean(np.abs(np.diff(gyro_

        return features

    segments = []
    labels = []

    for start in range(0, len(full_dataset) - window_size, step_size):
        window = full_dataset.iloc[start:start + window_size]
        features = extract_features(window.drop(columns=['Activity']))
        features['Activity'] = window['Activity'].mode()[0]
        segments.append(features)


    full_dataset_features = pd.DataFrame(segments)


    print(full_dataset_features.head())
    print(full_dataset_features.columns)
    full_dataset_features
```

```
   X axis_acc_mean  Y axis_acc_mean  Z axis_acc_mean  X axis_gyro_mean
0         1.019278        -0.124065         0.098463          0.011126
1         1.020130        -0.127602         0.090815          0.001517
2         1.020306        -0.126500         0.085787          0.001004
3         1.020574        -0.135778         0.078583          0.003728
4         1.021120        -0.133620         0.074981          0.007927

   Y axis_gyro_mean  Z axis_gyro_mean  X axis_acc_std  Y axis_acc_std
0         -0.003462          0.002171        0.002599        0.004167
1         -0.005042          0.005180        0.003144        0.007112
2          0.000379         -0.000387        0.002739        0.004956
3         -0.003065          0.005626        0.002894        0.007167
4         -0.000324          0.001303        0.003670        0.007297

   Z axis_acc_std  X axis_gyro_std  ...  gyro_yz_corr  acc_magnitude_me
0        0.006087         0.014279  ...     -0.391811            1.0315
1        0.007858         0.014730  ...     -0.345786            1.0321
2        0.005328         0.014036  ...     -0.589357            1.0317
3        0.007856         0.011213  ...     -0.648619            1.0326
4        0.006793         0.014978  ...     -0.428362            1.0325

   gyro_magnitude_mean  acc_magnitude_std  gyro_magnitude_std  \
0             0.016983           0.002750            0.011585
1             0.017814           0.003189            0.009726
2             0.016773           0.002805            0.009845
3             0.017803           0.002999            0.010669
4             0.020435           0.003753            0.013014

   acc_magnitude_auc  gyro_magnitude_auc  acc_magnitude_meandif  \
0         154.730478            2.547411               0.002579
1         154.820452            2.672080               0.002760
2         154.757367            2.515966               0.002519
3         154.892285            2.670486               0.002470
4         154.889922            3.065192               0.002902

   gyro_magnitude_meandif  Activity
0                0.004729  STANDING
1                0.004603  STANDING
2                0.004866  STANDING
3                0.004491  STANDING
4                0.006179  STANDING

[5 rows x 33 columns]
Index(['X axis_acc_mean', 'Y axis_acc_mean', 'Z axis_acc_mean',
       'X axis_gyro_mean', 'Y axis_gyro_mean', 'Z axis_gyro_mean',
       'X axis_acc_std', 'Y axis_acc_std', 'Z axis_acc_std', 'X axis_gy
       'Y axis_gyro_std', 'Z axis_gyro_std', 'X axis_acc_max',
```

```
       'Y axis_acc_max', 'Z axis_acc_max', 'X axis_gyro_max',
       'Y axis_gyro_max', 'Z axis_gyro_max', 'acc_xy_corr', 'acc_xz_cor
       'acc_yz_corr', 'gyro_xy_corr', 'gyro_xz_corr', 'gyro_yz_corr',
       'acc_magnitude_mean', 'gyro_magnitude_mean', 'acc_magnitude_std'
       'gyro_magnitude_std', 'acc_magnitude_auc', 'gyro_magnitude_auc',
       'acc_magnitude_meandif', 'gyro_magnitude_meandif', 'Activity'],
      dtype='object')
```

	X axis_acc_mean	Y axis_acc_mean	Z axis_acc_mean	X axis_gyro_mean	axis_
0	1.019278	-0.124065	0.098463	0.011126	
1	1.020130	-0.127602	0.090815	0.001517	
2	1.020306	-0.126500	0.085787	0.001004	
3	1.020574	-0.135778	0.078583	0.003728	
4	1.021120	-0.133620	0.074981	0.007927	
...	...	...	...	...	
5432	1.020444	-0.111389	0.073991	0.135337	
5433	1.009528	-0.219583	0.024889	0.025178	
5434	0.982389	-0.195648	-0.046481	0.574843	
5435	1.001694	-0.212491	-0.058093	0.441704	
5436	0.971824	-0.222407	-0.020787	0.024382	

5437 rows × 33 columns

14. Dataset split and scaling

```
!pip3 install imbalanced-learn
```

```
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.12.4-py3-none-any.whl.metadata (8.3 kB
Requirement already satisfied: numpy>=1.17.3 in /opt/homebrew/anaconda3
Requirement already satisfied: scipy>=1.5.0 in /opt/homebrew/anaconda3/
Requirement already satisfied: scikit-learn>=1.0.2 in /opt/homebrew/ana
Requirement already satisfied: joblib>=1.1.1 in /opt/homebrew/anaconda3
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/homebrew/an
Downloading imbalanced_learn-0.12.4-py3-none-any.whl (258 kB)
Installing collected packages: imbalanced-learn
Successfully installed imbalanced-learn-0.12.4
```

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import BorderlineSMOTE
from collections import Counter

X = full_dataset_features.iloc[:, :-1]
y = full_dataset_features.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Apply BorderlLineSMOTE only to training data

sm = BorderlineSMOTE(random_state=42, kind="borderline-1")
X_SMOTE, y_SMOTE = sm.fit_resample(X_train, y_train)

X_train_DF = pd.DataFrame(X_train)
X_test_DF = pd.DataFrame(X_test)
X_train_DF.columns = X_test_DF.columns = ['X axis_acc_mean', 'Y axis_a
       'X axis_gyro_mean', 'Y axis_gyro_mean', 'Z axis_gyro_mean',
       'X axis_acc_std', 'Y axis_acc_std', 'Z axis_acc_std', 'X axis_g
       'Y axis_gyro_std', 'Z axis_gyro_std', 'X axis_acc_max',
       'Y axis_acc_max', 'Z axis_acc_max', 'X axis_gyro_max',
       'Y axis_gyro_max', 'Z axis_gyro_max', 'acc_xy_corr', 'acc_xz_co
       'acc_yz_corr', 'gyro_xy_corr', 'gyro_xz_corr', 'gyro_yz_corr',
       'acc_magnitude_mean', 'gyro_magnitude_mean', 'acc_magnitude_st
       'gyro_magnitude_std', 'acc_magnitude_auc', 'gyro_magnitude_auc
```

```
       'acc_magnitude_meandif', 'gyro_magnitude_meandif']

X_SMOTE_DF = pd.DataFrame(X_SMOTE)
X_SMOTE_DF.columns = ['X axis_acc_mean', 'Y axis_acc_mean', 'Z axis_a
       'X axis_gyro_mean', 'Y axis_gyro_mean', 'Z axis_gyro_mean',
       'X axis_acc_std', 'Y axis_acc_std', 'Z axis_acc_std', 'X axis_
       'Y axis_gyro_std', 'Z axis_gyro_std', 'X axis_acc_max',
       'Y axis_acc_max', 'Z axis_acc_max', 'X axis_gyro_max',
       'Y axis_gyro_max', 'Z axis_gyro_max', 'acc_xy_corr', 'acc_xz_c
       'acc_yz_corr', 'gyro_xy_corr', 'gyro_xz_corr', 'gyro_yz_corr',
       'acc_magnitude_mean', 'gyro_magnitude_mean', 'acc_magnitude_st
       'gyro_magnitude_std', 'acc_magnitude_auc', 'gyro_magnitude_auc
       'acc_magnitude_meandif', 'gyro_magnitude_meandif']

#display(X_train_DF)
#display(X_SMOTE_DF)

original_distribution = Counter(y_train)
resampled_distribution = Counter(y_SMOTE)

display(original_distribution)
display(resampled_distribution)
```

```
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
Counter({'LAYING': 735,
         'STANDING': 734,
         'SITTING': 688,
         'WALKING': 667,
         'WALKING_UPSTAIRS': 591,
         'WALKING_DOWNSTAIRS': 587,
         'STAND_TO_LIE': 78,
         'SIT_TO_LIE': 65,
         'LIE_TO_SIT': 54,
         'STAND_TO_SIT': 54,
         'LIE_TO_STAND': 52,
         'SIT_TO_STAND': 44})
Counter({'WALKING_DOWNSTAIRS': 735,
         'WALKING_UPSTAIRS': 735,
         'SITTING': 735,
         'WALKING': 735,
         'LAYING': 735,
         'STANDING': 735,
         'STAND_TO_LIE': 735,
         'LIE_TO_STAND': 735,
         'LIE_TO_SIT': 735,
         'SIT_TO_STAND': 735,
         'STAND_TO_SIT': 735,
```

```
 'SIT_TO_LIE': 735})
```

15. Classifiers training

15.1. Building models, fitting data and testing

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline

# Logistic Regression

logreg = LogisticRegression(max_iter=1000)
pipeline_logreg = make_pipeline(StandardScaler(), logreg)
pipeline_logreg.fit(X_train, y_train)

print("Logistic Regression\n")
print(f"Training set score: {pipeline_logreg.score(X_train, y_train)}"
print(f"Test set score: {pipeline_logreg.score(X_test, y_test)}")
crossValScore_LogReg5 = cross_val_score(pipeline_logreg, X, y, cv=5)
crossValScore_LogReg10 = cross_val_score(pipeline_logreg, X, y, cv=10
print(f"Cross validation score with 5 folds: {crossValScore_LogReg5}"
print(f"Cross validation score with 10 folds: {crossValScore_LogReg10
print(f"Cross val score 5 folds mean: {crossValScore_LogReg5.mean()}"
print(f"Cross val score 10 folds mean: {crossValScore_LogReg10.mean()
print("")

# Random Forest

forest = RandomForestClassifier()
pipeline_forest = make_pipeline(forest)
pipeline_forest.fit(X_train, y_train)

print("Random Forests")
print(f"Training set score: {pipeline_forest.score(X_train, y_train)}"
print(f"Test set score: {pipeline_forest.score(X_test, y_test)}")
crossValScore_RF5 = cross_val_score(pipeline_forest, X, y, cv=5)
crossValScore_RF10 = cross_val_score(pipeline_forest, X, y, cv=10)
```

```
print(f"Cross validation score with 5 folds: {crossValScore_RF5}")
print(f"Cross validation score with 10 folds: {crossValScore_RF10}")
print(f"Cross val score 5 folds mean: {crossValScore_RF5.mean()}")
print(f"Cross val score 10 folds mean: {crossValScore_RF10.mean()}")
print("")

# Perceptron

perceptron = Perceptron()
pipeline_perceptron = make_pipeline(StandardScaler(), perceptron)
pipeline_perceptron.fit(X_train, y_train)

print("Perceptron")
print(f"Training set score: {pipeline_perceptron.score(X_train, y_tra
print(f"Test set score: {pipeline_perceptron.score(X_test, y_test)}")
crossValScore_Perceptron5 = cross_val_score(pipeline_perceptron, X, y
crossValScore_Perceptron10 = cross_val_score(pipeline_perceptron, X, 
print(f"Cross validation score with 5 folds: {crossValScore_Perceptro
print(f"Cross validation score with 10 folds: {crossValScore_Perceptr
print(f"Cross val score 5 folds mean: {crossValScore_Perceptron5.mean
print(f"Cross val score 10 folds mean: {crossValScore_Perceptron5.mea
print("")
```

```
Logistic Regression

Training set score: 0.9048057024603358
Test set score: 0.8823529411764706
Cross validation score with 5 folds: [0.88694853 0.81341912 0.83072677
Cross validation score with 10 folds: [0.91360294 0.86580882 0.75
 0.85110294 0.90791897 0.9281768  0.86003683]
Cross val score 5 folds mean: 0.8609566940851778
Cross val score 10 folds mean: 0.862076494962626

Random Forests
Training set score: 1.0
Test set score: 0.9108455882352942
Cross validation score with 5 folds: [0.88786765 0.86121324 0.84636615
Cross validation score with 10 folds: [0.91727941 0.88602941 0.8290441
 0.85477941 0.87108656 0.92081031 0.86372007]
Cross val score 5 folds mean: 0.8660995251366416
Cross val score 10 folds mean: 0.8740175766439172

Perceptron
Training set score: 0.848700850770292
Test set score: 0.8373161764705882
Cross validation score with 5 folds: [0.83180147 0.796875   0.80772769
Cross validation score with 10 folds: [0.875      0.84558824 0.7555147
```

```
 0.79779412 0.86740331 0.87108656 0.78084715]
Cross val score 5 folds mean: 0.827115238919855
Cross val score 10 folds mean: 0.827115238919855
```

```
from sklearn.model_selection import StratifiedKFold, cross_val_score

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
pipelines = [pipeline_logreg, pipeline_forest, pipeline_perceptron]
pipeline_names = ['Logistic Regression', 'Random Forest', 'Perceptron

for pipeline, pipeline_name in zip(pipelines, pipeline_names):
    scores = cross_val_score(pipeline, X, y, cv=skf)
    print(f"{pipeline_name} CV accuracy: {scores.mean():.4f} ± {score


# BEST MODEL IS RANDOM FOREST WITH STRATIFIEDKFOLD. SINCE SOME ACTIVI
# BALANCED CLASS WEIGHTING FOR ALL CLASSIFIERS
```

```
Logistic Regression CV accuracy: 0.8845 ± 0.0059
Random Forest CV accuracy: 0.9095 ± 0.0065
Perceptron CV accuracy: 0.8470 ± 0.0032
```

```
evaluate_model("Logistic Regression", y_test, y_pred_logreg)
evaluate_model("Random Forest", y_test, y_pred_forest)
evaluate_model("Perceptron", y_test, y_pred_perceptron)
```

```
Logistic Regression Metrics
Accuracy:  0.8786764705882353
Precision: 0.8780412608416714
Recall:    0.8786764705882353
F1 Score:  0.8779361235916237

Random Forest Metrics
Accuracy:  0.9071691176470589
Precision: 0.9079487262014536
Recall:    0.9071691176470589
F1 Score:  0.9061236676476062

Perceptron Metrics
Accuracy:  0.8235294117647058
Precision: 0.8406218095265997
Recall:    0.8235294117647058
F1 Score:  0.8169166797096838
```

15.1.1 Building models with BorderlineSMOTE

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import Pipeline

# Logistic Regression

pipeline_logreg_smote = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('classifier', logreg)
])

pipeline_forest_smote = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('classifier', forest)
])

pipeline_perceptron_smote = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('classifier', perceptron)
])

pipeline_logreg_smote.fit(X_train, y_train)
pipeline_forest_smote.fit(X_train, y_train)
pipeline_perceptron_smote.fit(X_train, y_train)


test_pipeline_logreg = Pipeline([
    ('scaler', pipeline_logreg_smote.named_steps['scaler']),  # reuse
    ('classifier', pipeline_logreg_smote.named_steps['classifier'])  #
])

y_pred_logreg_smote = test_pipeline_logreg.predict(X_test)

test_pipeline_forest = Pipeline([
    ('scaler', pipeline_forest_smote.named_steps['scaler']),
```

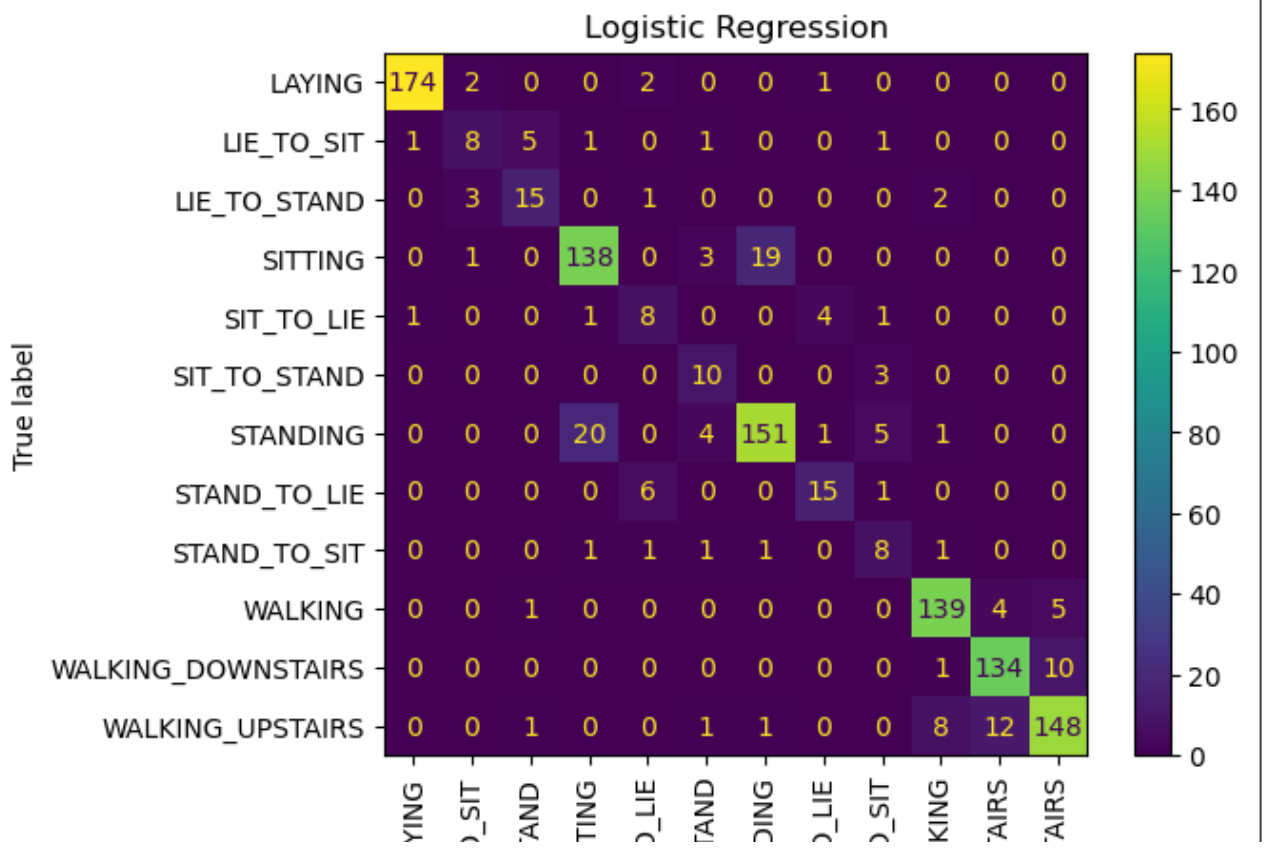
```
    ('classifier', pipeline_forest_smote.named_steps['classifier'])
])

y_pred_forest_smote = test_pipeline_forest.predict(X_test)


test_pipeline_perceptron = Pipeline([
    ('scaler', pipeline_perceptron_smote.named_steps['scaler']),
    ('classifier', pipeline_perceptron_smote.named_steps['classifier'])
])

y_pred_perceptron_smote = test_pipeline_perceptron.predict(X_test)

evaluate_model("Logistic Regression", y_test, y_pred_logreg_smote)
evaluate_model("Random Forest", y_test, y_pred_forest_smote)
evaluate_model("Perceptron", y_test, y_pred_perceptron_smote)


print("Logistic Regression\n")
crossValScore_LogReg5 = cross_val_score(pipeline_logreg_smote, X, y, c
crossValScore_LogReg10 = cross_val_score(pipeline_logreg_smote, X, y,
print(f"Cross validation score with 5 folds: {crossValScore_LogReg5}"
print(f"Cross validation score with 10 folds: {crossValScore_LogReg10
print(f"Cross val score 5 folds mean: {crossValScore_LogReg5.mean()}"
print(f"Cross val score 10 folds mean: {crossValScore_LogReg10.mean()
print("")

print("Random Forests")
crossValScore_RF5 = cross_val_score(pipeline_forest_smote, X, y, cv=5
crossValScore_RF10 = cross_val_score(pipeline_forest_smote, X, y, cv=1
print(f"Cross validation score with 5 folds: {crossValScore_RF5}")
print(f"Cross validation score with 10 folds: {crossValScore_RF10}")
print(f"Cross val score 5 folds mean: {crossValScore_RF5.mean()}")
print(f"Cross val score 10 folds mean: {crossValScore_RF10.mean()}")
print("")

print("Perceptron")
crossValScore_Perceptron5 = cross_val_score(pipeline_perceptron, X, y
crossValScore_Perceptron10 = cross_val_score(pipeline_perceptron, X, y
print(f"Cross validation score with 5 folds: {crossValScore_Perceptron
print(f"Cross validation score with 10 folds: {crossValScore_Perceptr
print(f"Cross val score 5 folds mean: {crossValScore_Perceptron5.mean
print(f"Cross val score 10 folds mean: {crossValScore_Perceptron5.mean
print("")
```

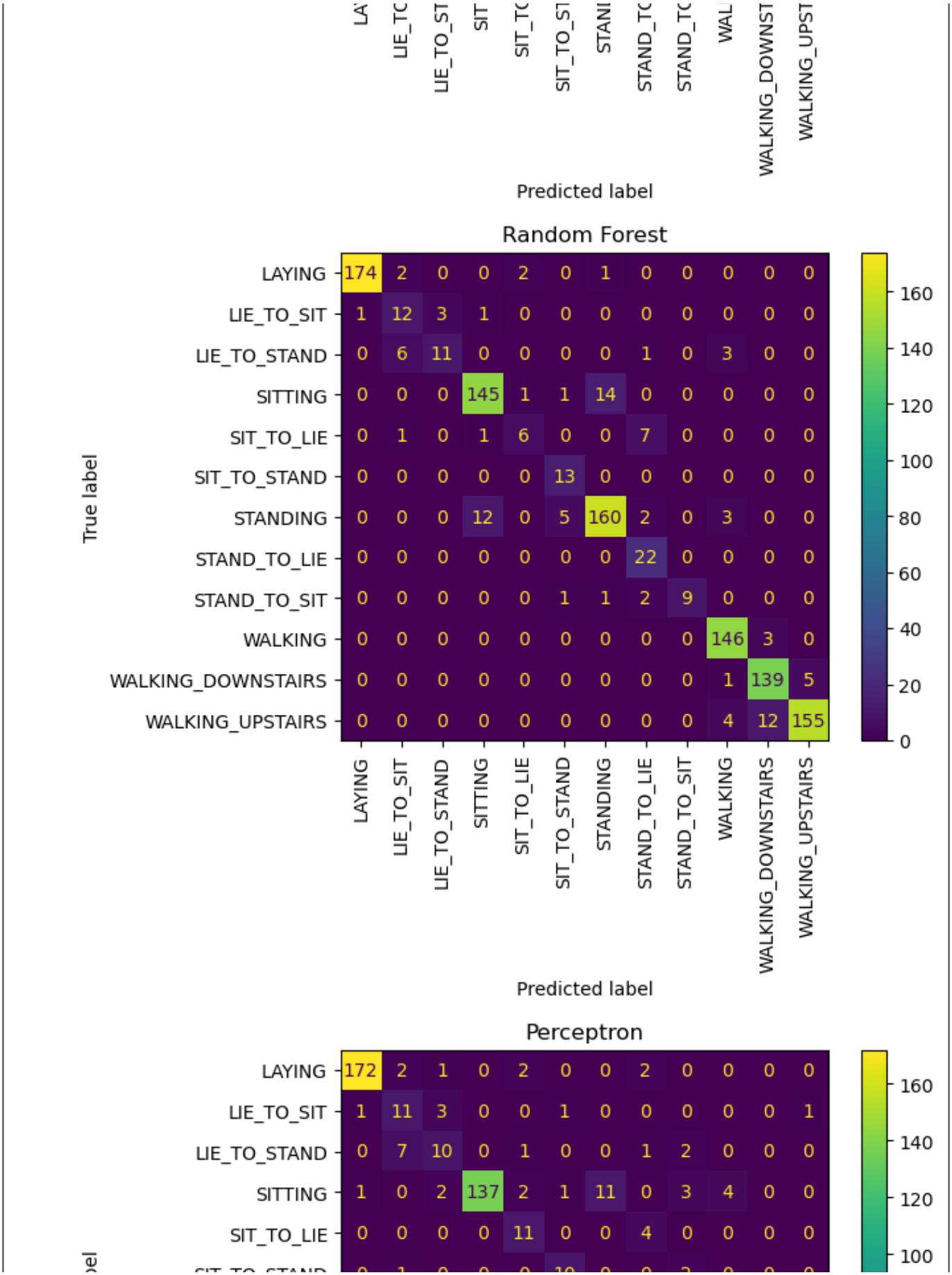
```
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
Cross validation score with 5 folds: [0.88051471 0.8125     0.8316467
Cross validation score with 10 folds: [0.90625    0.84007353 0.735294
 0.86029412 0.86556169 0.91160221 0.85635359]
Cross val score 5 folds mean: 0.8535983413604631
Cross val score 10 folds mean: 0.8510355730690067

Random Forests
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
Cross validation score with 5 folds: [0.87867647 0.84007353 0.8482060
Cross validation score with 10 folds: [0.91360294 0.87132353 0.821691
 0.86213235 0.86556169 0.92633517 0.87845304]
Cross val score 5 folds mean: 0.860769319227231
```

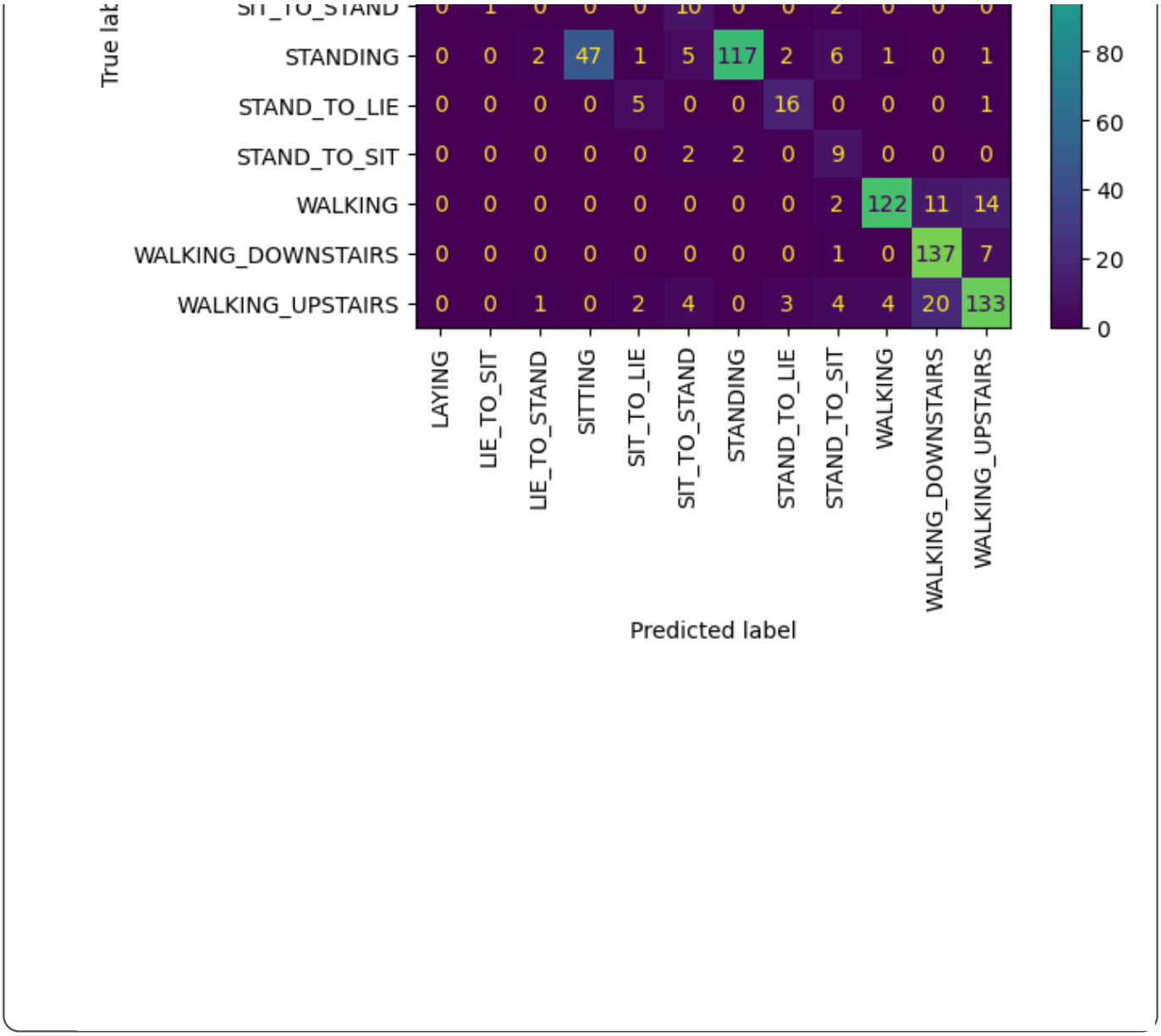
```
Cross val score 10 folds mean: 0.874020284909544

Perceptron
Cross validation score with 5 folds: [0.83180147 0.796875   0.8077276
Cross validation score with 10 folds: [0.875      0.84558824 0.755514
 0.79779412 0.86740331 0.87108656 0.78084715]
Cross val score 5 folds mean: 0.827115238919855
Cross val score 10 folds mean: 0.827115238919855
```

15.2. Plotting confusion matrixes for all models

```
from sklearn.metrics import ConfusionMatrixDisplay

models = [pipeline_logreg_smote, pipeline_forest_smote, pipeline_perc
model_names = ["Logistic Regression", "Random Forest", "Perceptron"]

for model, name in zip(models, model_names):
    ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)
    plt.xticks(rotation=90)
    plt.title(name)
    plt.show()
```




Predicted label
Random Forest
True label
LAYING
LIE_TO_SIT
LIE_TO_STAND
SITTING
SIT_TO_LIE
SIT_TO_STAND
STANDING
STAND_TO_LIE
STAND_TO_SIT
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
Predicted label
Perceptron



15.3. Tuning-in hyperparameters via Grid search

```
# WITH SMOTE
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.exceptions import ConvergenceWarning

warnings.filterwarnings("ignore", category=ConvergenceWarning)

# Note: if best hyperparameters fall at the edges, we want to expand
```

```
param_grid_logreg = [
    {
    'classifier__penalty': ['l1', 'l2'],
    'classifier__C': [0.01, 0.1, 1, 10],
    'classifier__solver': ['saga'],
    'classifier__class_weight': [None, 'balanced'],
    'classifier__max_iter': [500, 1000, 2000]
    },
    {
    'classifier__penalty': [None, 'l2'],
    'classifier__C': [0.01, 0.1, 1, 10],
    'classifier__solver': ['lbfgs'],
    'classifier__class_weight': [None, 'balanced'],
    'classifier__max_iter': [500, 1000, 2000]
    }
]

param_grid_rf = {
    'classifier__n_estimators': [50, 100, 200, 350, 500, 1000],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__class_weight': [None, 'balanced']
}

param_grid_perceptron = {
    'classifier__penalty': [None, 'l2', 'l1', 'elasticnet'],
    'classifier__alpha': [0.00007, 0.00006, 0.00005, 0.0001, 0.001, 0
    'classifier__max_iter': [500, 1000, 2000, 5000],
    'classifier__class_weight': [None, 'balanced']
}

models = [
    ('Logistic Regression', pipeline_logreg_smote),
    ('Random Forest', pipeline_forest_smote),
    ('Perceptron', pipeline_perceptron_smote)
]

param_grids = [param_grid_logreg, param_grid_rf, param_grid_perceptro

best_models = []
for (name, model), param_grid in zip(models, param_grids):
    print(f"\n Tuning {name}")
    grid = GridSearchCV(model, param_grid, cv=skf, scoring='accuracy'
    grid.fit(X_train, y_train)
```

```
    print(f"Best parameters for {name}: {grid.best_params_}")
    print(f"Best cross-validated score: {grid.best_score_:.4f}")

    # Evaluate on the test set
    y_pred = grid.predict(X_test)
    print(f"\nClassification Report for {name}:\n")
    print(classification_report(y_test, y_pred))

    best_models.append((name, grid.best_estimator_))
```

```
 Tuning Logistic Regression
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
```

```
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
```

16. Feature reduction

16.1. Reducing

```
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from imblearn.pipeline import Pipeline

# Principal Component Analysis (PCA)

pca = PCA(n_components=0.95)

pipeline_pca_logreg = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('pca', pca),
    ('classifier', logreg)
])

pipeline_pca_perceptron = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
```

```
    ('pca', pca),
    ('classifier', perceptron)
])

pipeline_pca_forest = Pipeline([
    ('scaler', StandardScaler()),  # Optional for trees, but used for
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('pca', pca),
    ('classifier', forest)
])

# Kernel PCA

kernelpca = KernelPCA(n_components=15)

pipeline_kpca_logreg = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('kpca', kernelpca),
    ('classifier', logreg)
])

pipeline_kpca_perceptron = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('kpca', kernelpca),
    ('classifier', perceptron)
])

pipeline_kpca_forest = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('kpca', kernelpca),
    ('classifier', forest)
])
```

```
# === Fit on training data ===
for name, pipeline in [
    ('PCA_LogReg', pipeline_pca_logreg),
    ('PCA_Perceptron', pipeline_pca_perceptron),
    ('PCA_Forest', pipeline_pca_forest),
    ('KernelPCA_LogReg', pipeline_kpca_logreg),
```

```
    ('KernelPCA_Perceptron', pipeline_kpca_perceptron),
    ('KernelPCA_Forest', pipeline_kpca_forest)
]:
    print(f"Fitting {name}...")
    pipeline.fit(X_train, y_train)

# === Predict and evaluate ===
y_pred_pca_logreg = pipeline_pca_logreg.predict(X_test)
y_pred_pca_perceptron = pipeline_pca_perceptron.predict(X_test)
y_pred_pca_forest = pipeline_pca_forest.predict(X_test)

y_pred_kpca_logreg = pipeline_kpca_logreg.predict(X_test)
y_pred_kpca_perceptron = pipeline_kpca_perceptron.predict(X_test)
y_pred_kpca_forest = pipeline_kpca_forest.predict(X_test)

evaluate_model("PCA + Logistic Regression", y_test, y_pred_pca_logreg
evaluate_model("PCA + Perceptron", y_test, y_pred_pca_perceptron)
evaluate_model("PCA + Random Forest", y_test, y_pred_pca_forest)

evaluate_model("KernelPCA + Logistic Regression", y_test, y_pred_kpca_
evaluate_model("KernelPCA + Perceptron", y_test, y_pred_kpca_perceptr
evaluate_model("KernelPCA + Random Forest", y_test, y_pred_kpca_fores
```

```
Fitting PCA_LogReg...
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
Fitting PCA_Perceptron...
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
Fitting PCA_Forest...
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
Fitting KernelPCA_LogReg...
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
Fitting KernelPCA_Perceptron...
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
Fitting KernelPCA_Forest...
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/bas
  warnings.warn(
PCA + Logistic Regression Metrics
Accuracy:  0.23161764705882354
Precision: 0.2685951686886858
Recall:    0.23161764705882354
```

```
F1 Score:  0.21359799029263438

PCA + Perceptron Metrics
Accuracy:  0.24448529411764705
Precision: 0.3016814168173023
Recall:    0.24448529411764705
F1 Score:  0.2567055432121889

PCA + Random Forest Metrics
Accuracy:  0.4494485294117647
Precision: 0.4810976719870111
Recall:    0.4494485294117647
F1 Score:  0.4304136370028929

KernelPCA + Logistic Regression Metrics
Accuracy:  0.8272058823529411
Precision: 0.8350237287169455
Recall:    0.8272058823529411
F1 Score:  0.8288876074688626

KernelPCA + Perceptron Metrics
Accuracy:  0.7481617647058824
Precision: 0.7865370540051796
Recall:    0.7481617647058824
F1 Score:  0.7475340643171248

KernelPCA + Random Forest Metrics
Accuracy:  0.8556985294117647
Precision: 0.8578389161745571
Recall:    0.8556985294117647
F1 Score:  0.8555886985697922
```

```
print("Cross-validation scores\n")

for name, pipeline, data in [
    ("PCA + Logistic Regression", pipeline_pca_logreg, X),
    ("PCA + Perceptron", pipeline_pca_perceptron, X),
    ("PCA + Random Forest", pipeline_pca_forest, X),
    ("KernelPCA + Logistic Regression", pipeline_kpca_logreg, X),
    ("KernelPCA + Perceptron", pipeline_kpca_perceptron, X),
    ("KernelPCA + Random Forest", pipeline_kpca_forest, X)
]:
    print(name)
    scores_5 = cross_val_score(pipeline, data, y, cv=5)
    scores_10 = cross_val_score(pipeline, data, y, cv=10)
    print(f"5-fold mean: {scores_5.mean():.4f}, 10-fold mean: {scores_1
```

```
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
5-fold mean: 0.7195, 10-fold mean: 0.6866

PCA + Random Forest
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
```

```
5-fold mean: 0.8264, 10-fold mean: 0.8334

KernelPCA + Logistic Regression
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
```

```
from sklearn.feature_selection import RFECV, SelectFromModel
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import BorderlineSMOTE

# === Configuración de clasificadores ===
logreg = LogisticRegression(max_iter=1000, random_state=42)
perceptron = Perceptron(max_iter=1000, random_state=42)
forest = RandomForestClassifier(n_estimators=100, random_state=42)

# === Configuración de validación cruzada para RFECV ===
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# === Pipelines con RFECV (Logistic Regression y Perceptron) ===

rfecv_logreg = RFECV(estimator=logreg, step=1, cv=cv, scoring='f1_wei
pipeline_rfecv_logreg = ImbPipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('feature_selection', rfecv_logreg),
    ('classifier', logreg)
])

rfecv_perceptron = RFECV(estimator=perceptron, step=1, cv=cv, scoring=
pipeline_rfecv_perceptron = ImbPipeline([
    ('scaler', StandardScaler()),
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('feature_selection', rfecv_perceptron),
    ('classifier', perceptron)
])
```

```
# === Pipeline con SelectFromModel (Random Forest) ===

select_rf = SelectFromModel(estimator=forest, threshold="median")  # (
pipeline_select_rf = ImbPipeline([
    ('scaler', StandardScaler()),  # Opcional para Random Forest
    ('smote', BorderlineSMOTE(kind='borderline-1', random_state=42)),
    ('feature_selection', select_rf),
    ('classifier', forest)
])

for name, pipeline, data in [
    ('RFECV + Logistic Regression', pipeline_rfecv_logreg, X),
    ('RFECV + Perceptron', pipeline_rfecv_perceptron, X),
    ('SelectFromModel + Random Forest', pipeline_select_rf, X)
]:
    print(name)
    scores_5 = cross_val_score(pipeline, data, y, cv=5)
    scores_10 = cross_val_score(pipeline, data, y, cv=10)
    print(f"5-fold mean: {scores_5.mean():.4f}, 10-fold mean: {scores_
```

```
RFECV + Logistic Regression
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
```

```
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
5-fold mean: 0.8553, 10-fold mean: 0.8516

RFECV + Perceptron
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
```

17. Performance metrics

```
from sklearn.metrics import accuracy_score, precision_score, recall_s

logreg_best = best_models[0][1]
forest_best = best_models[1][1]
perceptron_best = best_models[2][1]

y_pred_logreg = logreg_best.predict(X_test)
y_pred_forest = forest_best.predict(X_test)
y_pred_perceptron = perceptron_best.predict(X_test)
```

```
def evaluate_model(name, y_true, y_pred):
    print(f"{name} Metrics")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred)}")
    print(f"Precision: {precision_score(y_true, y_pred, average='weigl
    print(f"Recall:    {recall_score(y_true, y_pred, average='weighte
    print(f"F1 Score:  {f1_score(y_true, y_pred, average='weighted')}'
    print("")
    #sns.heatmap(confusion_matrix(y_true, y_pred), square=True, annot=
    #plt.xlabel('Predicted value')
    #plt.ylabel('True value')


evaluate_model("Logistic Regression", y_test, y_pred_logreg)
evaluate_model("Random Forest", y_test, y_pred_forest)
evaluate_model("Perceptron", y_test, y_pred_perceptron)

#display(pd.DataFrame(y_test))
#display(pd.DataFrame(y_pred_logreg))
```

```
Logistic Regression Metrics
Accuracy:  0.2523020257826888
Precision: 0.19826824164650267
Recall:    0.2523020257826888
F1 Score:  0.1731199176348857

Random Forest Metrics
Accuracy:  0.26519337016574585
Precision: 0.28853810400771723
Recall:    0.26519337016574585
F1 Score:  0.2009967936991626

Perceptron Metrics
Accuracy:  0.016574585635359115
Precision: 0.0003086515015895552
Recall:    0.016574585635359115
F1 Score:  0.0006060177563202601

/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/me
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/me
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/me
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result
```

```
# Obtener accuracy por fold (cv=10)
acc_perceptron = cross_val_score(perceptron, X_train, y_train, cv=10,
acc_rf = cross_val_score(forest, X_train, y_train, cv=10, scoring='ac
acc_logreg = cross_val_score(logreg, X_train, y_train, cv=10, scoring=

# Unificar en una matriz para análisis estadístico
results_matrix = np.vstack([acc_perceptron, acc_rf, acc_logreg]).T
```

```
from scipy.stats import friedmanchisquare

stat, p = friedmanchisquare(*results_matrix.T)
print(f"Friedman statistic: {stat:.4f}, p-value: {p:.4f}")
```

```
Friedman statistic: 19.5385, p-value: 0.0001
```

```
!pip3 install scikit-posthocs

import scikit_posthocs as sp
import pandas as pd
import numpy as np

# Crear DataFrame con los resultados por fold
df_scores = pd.DataFrame(results_matrix, columns=["Perceptron", "Random

# Test de Nemenyi post-hoc tras Friedman
nemenyi_result = sp.posthoc_nemenyi_friedman(df_scores.values)

# Etiquetar las filas y columnas con nombres de modelos
nemenyi_result.columns = df_scores.columns
nemenyi_result.index = df_scores.columns

# Mostrar tabla de p-valores
print("📌 Tabla de p-valores del test de Nemenyi:")
print(nemenyi_result.round(12))
```

```
Requirement already satisfied: scikit-posthocs in /opt/homebrew/anacon
Requirement already satisfied: numpy in /opt/homebrew/anaconda3/envs/D
Requirement already satisfied: scipy>=1.9.0 in /opt/homebrew/anaconda3
Requirement already satisfied: statsmodels in /opt/homebrew/anaconda3/
Requirement already satisfied: pandas>=0.20.0 in /opt/homebrew/anaconda
Requirement already satisfied: seaborn in /opt/homebrew/anaconda3/envs
Requirement already satisfied: matplotlib in /opt/homebrew/anaconda3/e
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/homebrew
```

```
Requirement already satisfied: pytz>=2020.1 in /opt/homebrew/anaconda3/
Requirement already satisfied: tzdata>=2022.7 in /opt/homebrew/anaconda
Requirement already satisfied: contourpy>=1.0.1 in /opt/homebrew/anacon
Requirement already satisfied: cycler>=0.10 in /opt/homebrew/anaconda3/
Requirement already satisfied: fonttools>=4.22.0 in /opt/homebrew/anac
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/homebrew/anac
Requirement already satisfied: packaging>=20.0 in /opt/homebrew/anacon
Requirement already satisfied: pillow>=8 in /opt/homebrew/anaconda3/env
Requirement already satisfied: pyparsing>=2.3.1 in /opt/homebrew/anacon
Requirement already satisfied: importlib-resources>=3.2.0 in /opt/homeb
Requirement already satisfied: patsy>=0.5.6 in /opt/homebrew/anaconda3/
Requirement already satisfied: zipp>=3.1.0 in /opt/homebrew/anaconda3/e
Requirement already satisfied: six>=1.5 in /opt/homebrew/anaconda3/envs
📌 Tabla de p-valores del test de Nemenyi:
                     Perceptron  Random Forest  Logistic Regression
Perceptron             1.000000       0.000039             0.049475
Random Forest          0.000039       1.000000             0.109180
Logistic Regression    0.049475       0.109180             1.000000
```

18. Saving models to disk

```
best_models
```

```
[('Logistic Regression',
  Pipeline(steps=[('standardscaler', StandardScaler()),
                  ('logisticregression',
                   LogisticRegression(C=10, max_iter=2000,
solver='saga'))])),
 ('Random Forest',
  Pipeline(steps=[('randomforestclassifier',
                   RandomForestClassifier(class_weight='balanced',
max_depth=20,
                                          n_estimators=350))])),
 ('Perceptron',
  Pipeline(steps=[('standardscaler', StandardScaler()),
                  ('perceptron',
                   Perceptron(alpha=6e-05, max_iter=500,
penalty='l1'))]))]
```

```
!pip install joblib

from joblib import dump, load



#dump(logreg_best, 'logreg_best.joblib')
#dump(forest_best, 'forest_best.joblib')
#dump(perceptron_best, 'perceptron_best.joblib')

#dump(logreg, 'logreg.joblib')
#dump(forest, 'forest.joblib')
#dump(perceptron, 'perceptron.joblib')
```

```
Requirement already satisfied: joblib in /opt/homebrew/anaconda3/envs/l
```

19. Data fusion

```
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_s
from imblearn.over_sampling import BorderlineSMOTE
import numpy as np


cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Definir clasificadores base
clf_rf = RandomForestClassifier(random_state=42)
clf_lr = LogisticRegression(max_iter=1000, random_state=42)
clf_nb = GaussianNB()
clf_ada = AdaBoostClassifier(random_state=42)

for idx, (X, y) in enumerate(datasets, start=1):
    print(f"\n📊 Resultados para el Dataset {idx}:\n")

    # 1. Random Forest con BorderlineSMOTE dentro de pipeline (imblea
    print("🔷 Agregación con Bosques Aleatorios (BorderlineSMOTE):")
    pipeline_rf_smote = Pipeline([
        ('smote', BorderlineSMOTE(random_state=42)),
```

```
    ('scaler', StandardScaler()),
    ('rf', clf_rf)
])
scores_rf = cross_validate(pipeline_rf_smote, X, y, cv=cv, scorin
for metric in scoring:
    print(f"{metric.capitalize()}: {np.mean(scores_rf[f'test_{met

# 2. Votación personalizada con BorderlineSMOTE
print("\n🔷 Votación personalizada con BorderlineSMOTE (Logistic

acc_scores, prec_scores, rec_scores, f1_scores = [], [], [], []

for train_idx, test_idx in cv.split(X, y):
    # Convertir a numpy arrays para evitar problemas de tipo en f
    X_train, X_test = X.iloc[train_idx].values, X.iloc[test_idx].
    y_train, y_test = y.iloc[train_idx].values, y.iloc[test_idx].

    # Aplicar BorderlineSMOTE solo en train
    smote = BorderlineSMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X_train, y_trai

    # Escalar datos
    scaler = StandardScaler()
    X_resampled = scaler.fit_transform(X_resampled)
    X_test_scaled = scaler.transform(X_test)

    # Entrenar clasificadores clonados para evitar "fit" compartid
    clf1 = clone(clf_lr).fit(X_resampled, y_resampled)
    clf2 = clone(clf_rf).fit(X_resampled, y_resampled)
    clf3 = clone(clf_nb).fit(X_resampled, y_resampled)

    # Predicciones de cada clasificador
    pred1 = clf1.predict(X_test_scaled)
    pred2 = clf2.predict(X_test_scaled)
    pred3 = clf3.predict(X_test_scaled)

    # Votación mayoritaria
    predictions = np.vstack((pred1, pred2, pred3)).T
    final_preds = []
    for preds in predictions:
        mode_val = pd.Series(preds).mode()
        final_preds.append(mode_val.iloc[0])
    final_preds = np.array(final_preds)
```

```
        # Evaluar métricas
        acc_scores.append(accuracy_score(y_test, final_preds))
        prec_scores.append(precision_score(y_test, final_preds, averag
        rec_scores.append(recall_score(y_test, final_preds, average='w
        f1_scores.append(f1_score(y_test, final_preds, average='weigh

    # 3. AdaBoost con BorderlineSMOTE dentro de pipeline
    print("\n🔷 AdaBoost (BorderlineSMOTE):")
    pipeline_ada_smote = Pipeline([
        ('smote', BorderlineSMOTE(random_state=42)),
        ('scaler', StandardScaler()),
        ('ada', clf_ada)
    ])
    scores_ada = cross_validate(pipeline_ada_smote, X, y, cv=cv, scor
    for metric in scoring:
        print(f"{metric.capitalize()}: {np.mean(scores_ada[f'test_{me

    # Resultados de votación personalizada
    print("\nResultados de votación personalizada:")
    print(f"Accuracy:  {np.mean(acc_scores):.4f}")
    print(f"Precision: {np.mean(prec_scores):.4f}")
    print(f"Recall:    {np.mean(rec_scores):.4f}")
    print(f"F1 Score:  {np.mean(f1_scores):.4f}")
```

```
📊 Resultados para el Dataset 1:

🔷 Agregación con Bosques Aleatorios (BorderlineSMOTE):
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
```

```
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
Accuracy: 0.9136
Precision: 0.9175
Recall: 0.9136
F1: 0.9144

🔷 Votación personalizada con BorderlineSMOTE (Logistic Regression, F
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(

🔷 AdaBoost (BorderlineSMOTE):
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
/opt/homebrew/anaconda3/envs/DS/lib/python3.9/site-packages/sklearn/b
  warnings.warn(
```