Group Assignment 1: Human Activity Detection

In this assignment you will create you own dataset for classification. You will explore which ML algorithms are best to classify this and you will present your best solution.

- Create your own dataset for custom human motions using Phyphox
- There should be at least 3 distinct types of motions
- The motions should be different to the ones used in the UCI dataset (Not: walking, sitting, standing, laying, stairs)
- Follow the steps and answer the questions given in this notebook

Generating your dataset:

For this assignment you will create your own dataset of motions that you collect with an Accelerometer and Gyroscope. For this you can use your phone as a sensor. To be able to collect your data you can best use an app called phyphox, this is a free app available in app stores. This app can be configured to acces your sensordata, sample it as given frequency's. you can set it up te have experiment timeslots, and the data with a timestamp can be exported to a needed output format.

When you installed the app you can setup a custum experiment by clicking on the + button. Define an experiment name, sample frequency and activate the Accelerometer and Gyroscope. Your custom experiment will be added, you can run it pressing the play button and you will see sensor motion. Pressing the tree dots (...) lets you define timed runs, remote access and exporting data.

Phyphox will generate 2 files with sensor data, one for the Accelerometer and one for the Giro. Both files will have timestamps which might not match the recorded sensor data for each sensor. Please, preprocess and merge the files for using it as your dataset for training, testing and deploying your own supervised learning model.

steps

With your own generated dataset the similar sequence of steps should be taken to train your model.

These are the generic steps to be taken

1. Frame the problem and look at the big picture.
2. Get the data.
3. Explore the data to gain insights.
4. Prepare the data to better expose the underlying data patterns to Machine Learning algorithms
5. Explore many different models and short-list the best ones.
6. Fine-tune your models and combine them into a great solution.
7. Present your solution.
8. Launch, monitor, and maintain your system.
9. Additional Questions

In the Notebook this structure is used for dividing the different steps, so make sure you do the implementation and analisis at these location in the notebook.

You may add additinal code blocks, but keep the seperation of the given structure.

At the end of each block summarize / comment / conclude your current step in the given textblocks.

```
Project group 20:
Lars Claassen   - 4159632
Tonnie Bour     - 4130456
Pjotr Maes      - 3839001
```

1. Frame the problem and look at the big picture

Describe the problem at hand and explain your approach

```
A model is designed to classify human activities. To achieve this, data is gathered to
train multiple models for classification.

We use our phones and their built-in sensors to track the following activities:
    - Casual walking
    - Walking while looking at the phone
    - Drinking
For each activity, gyroscope, acceleration, and linear acceleration data are recorded
and logged. Each activity is measured separately in 10-second intervals. The recorded
data is then compressed into various features, including minimum, maximum, mean, and
standard deviation values, resulting in 38 features per measurement.

The compressed values from each measurement are compiled into a single dataset, which is
then split into training and testing sets. Different model types are trained and
evaluated to determine the most suitable one, providing insight into which model
performs best with our data.
```

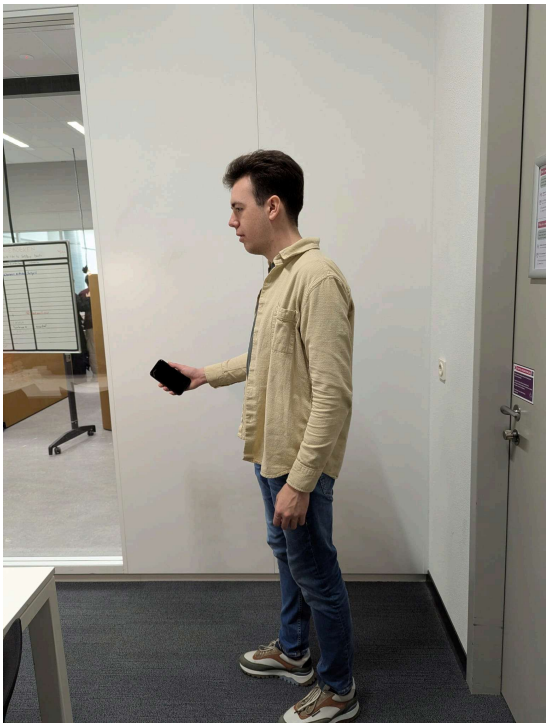
2. Get the data.

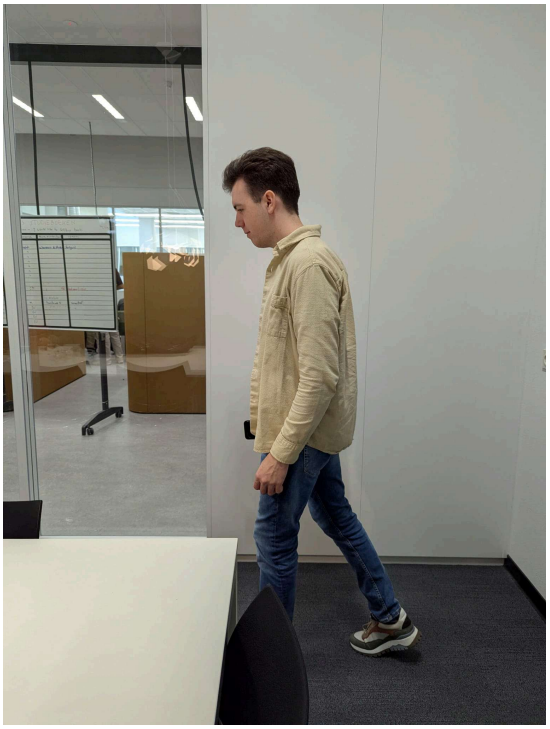
Initialize the system, get all needed libraries, retreive the data and import it

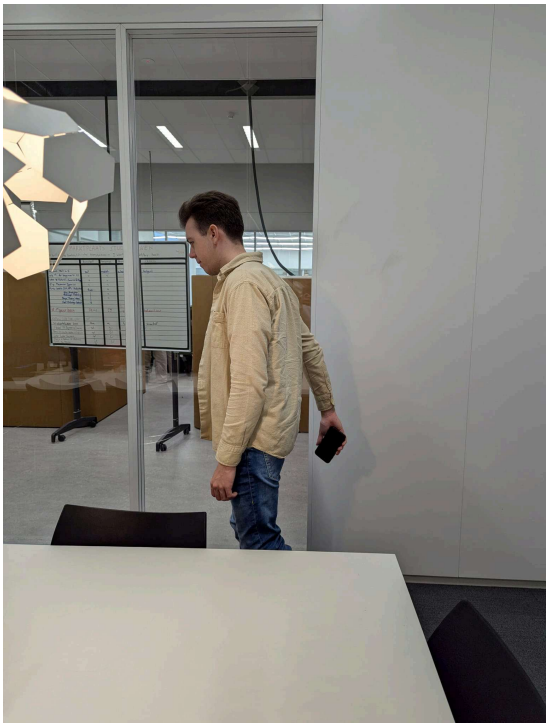
> Create your own dataset

> Explain and show (with a few images) which motions you are classifing, how you generated them, what the problems where you encountered in this process!

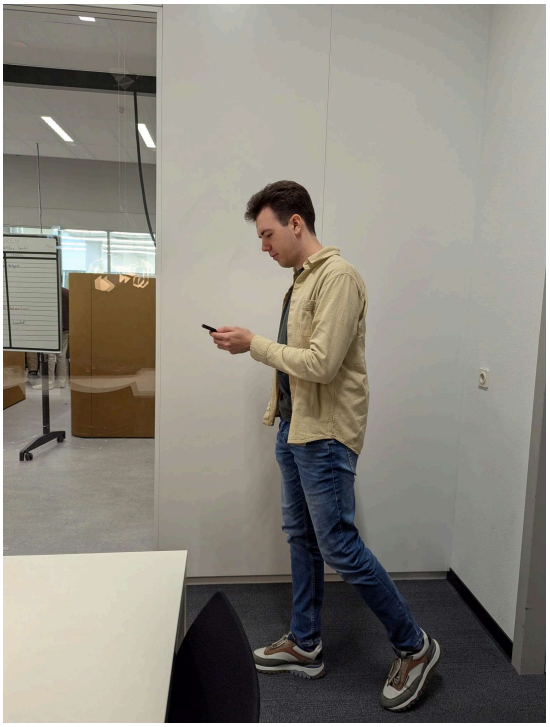
```
To gather the data we used an app called Phyphox. Within this app we created our own
exercise that tracks the gyroscope, acceleration, and linear acceleration. This data is
logged and saved per activity. Each activity/measurement records for about 10 seconds,
this is for each exercise the same to prevend a bias from forming towards the "longer
during" activity.

For casual walking we held the phone in the palm of our hand while swinging it as you
would naturally do while walking.
```

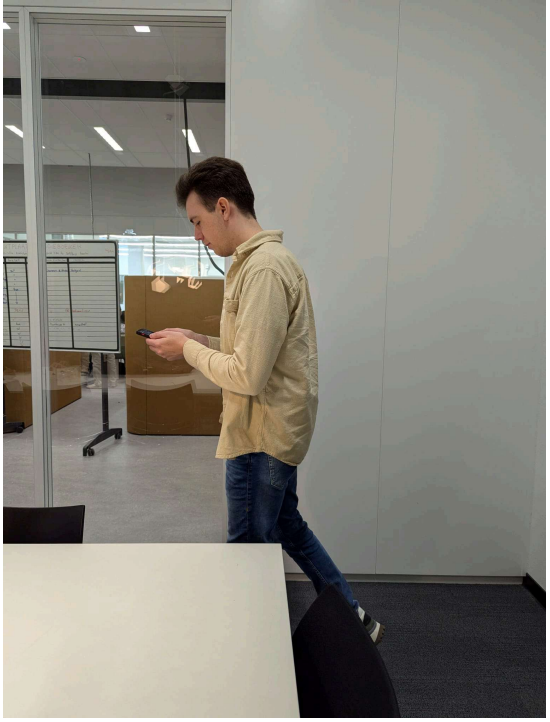




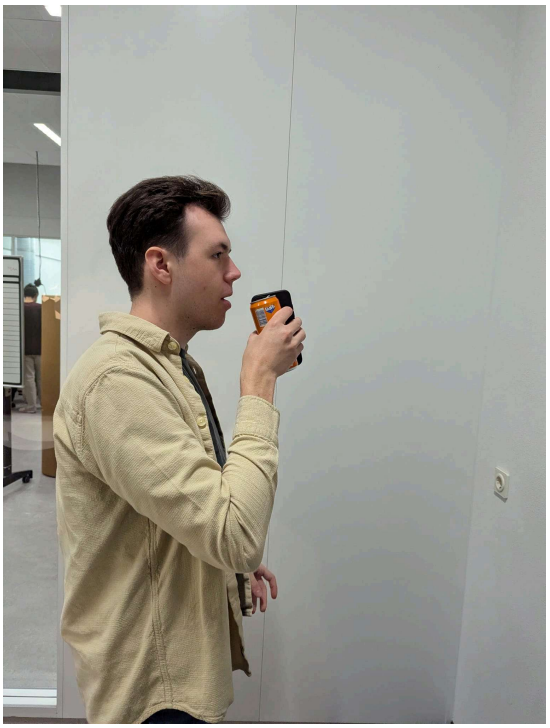


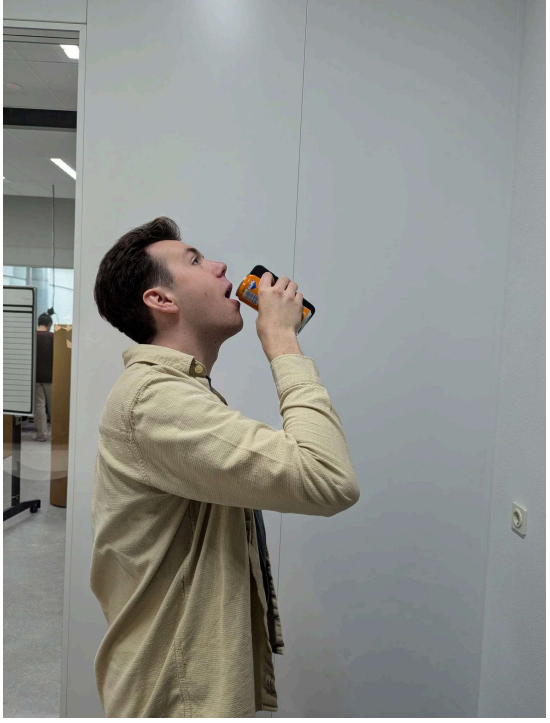
To measure walking while looking at the phone a subject would walk with the phone as if he/she would be texting or looking at their phone.

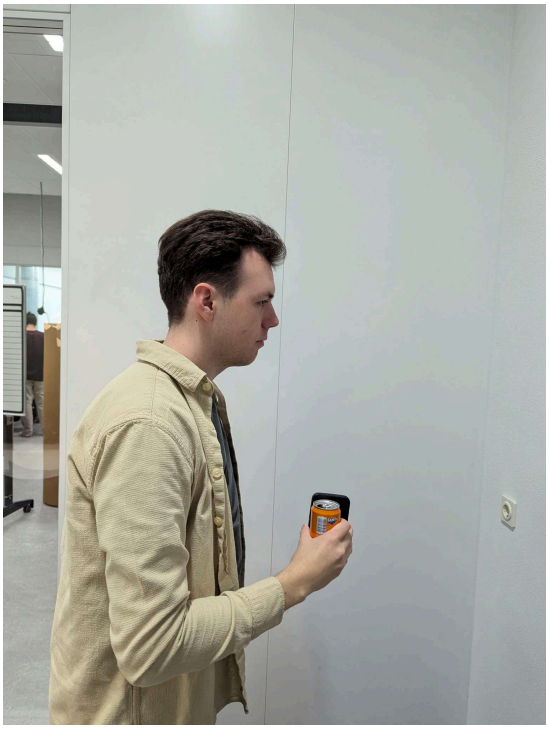




To measure drinking a subject would hold the phone in the hand together with their beverage (of their choice) and take multiple sips.







To achieve the best model performance, a large amount of data is required. However, creating a prototype model using a smaller dataset is still feasible.

In our case, three subjects perform each activity 10 times, resulting in 30 datasets per activity across three subjects. This brings the total dataset size to 90 measurements.

Two significant problems were encountered during data collection:
1) Finding motions that are distinct from each other and keep them consisten over multiple people deemed difficult.
2) Aligning different naming conventions, as various phone brands produced results with differing labels.

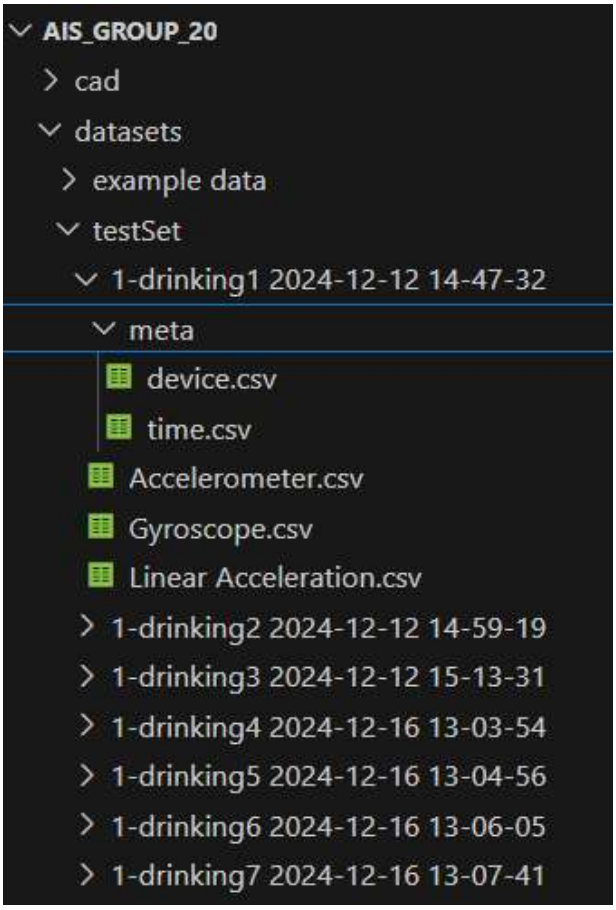
3. Explore the data to gain insights.

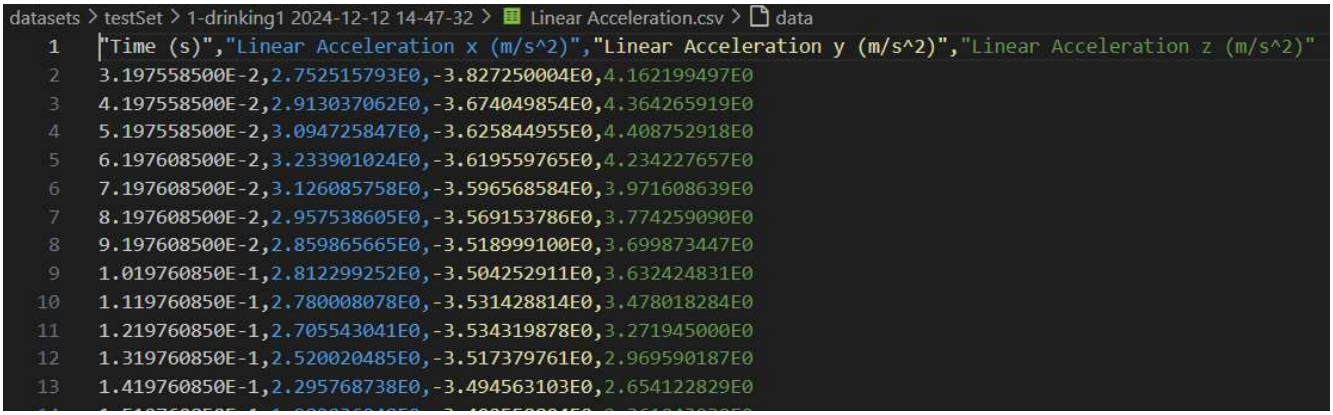
Explore the data in any possible way, visualize the results (if you have multiple plots of the same kind of data put them in one larger plot)

Each measurement is put seperately in a folder that specifies who has done the measurement, the action and the occurence of the action. The three csv files have raw data of the sensors, along with a timestamp.

In "1-drinking1 2024-12-12 14-47-32" the first 1 states the person who has done the measurement. Afterwards the action is stated along with the measurement index. Everything after that is stating the time, but this information is ignored, since this has no significant importance to the model or testing.

A plotted version of the data is visible below.





In [1]:
```python
import pandas as pd

# File paths
accelerometer_path = 'datasets/testSet/1-drinking1 2024-12-12 14-47-32/Accelerometer.csv'
gyroscope_path = 'datasets/testSet/1-drinking1 2024-12-12 14-47-32/Gyroscope.csv'
linear_acceleration_path = 'datasets/testSet/1-drinking1 2024-12-12 14-47-32/Linear Acceleration.csv'

# Load data from CSV files
accelerometer_df = pd.read_csv(accelerometer_path)
gyroscope_df = pd.read_csv(gyroscope_path)
linear_acceleration_df = pd.read_csv(linear_acceleration_path)
```

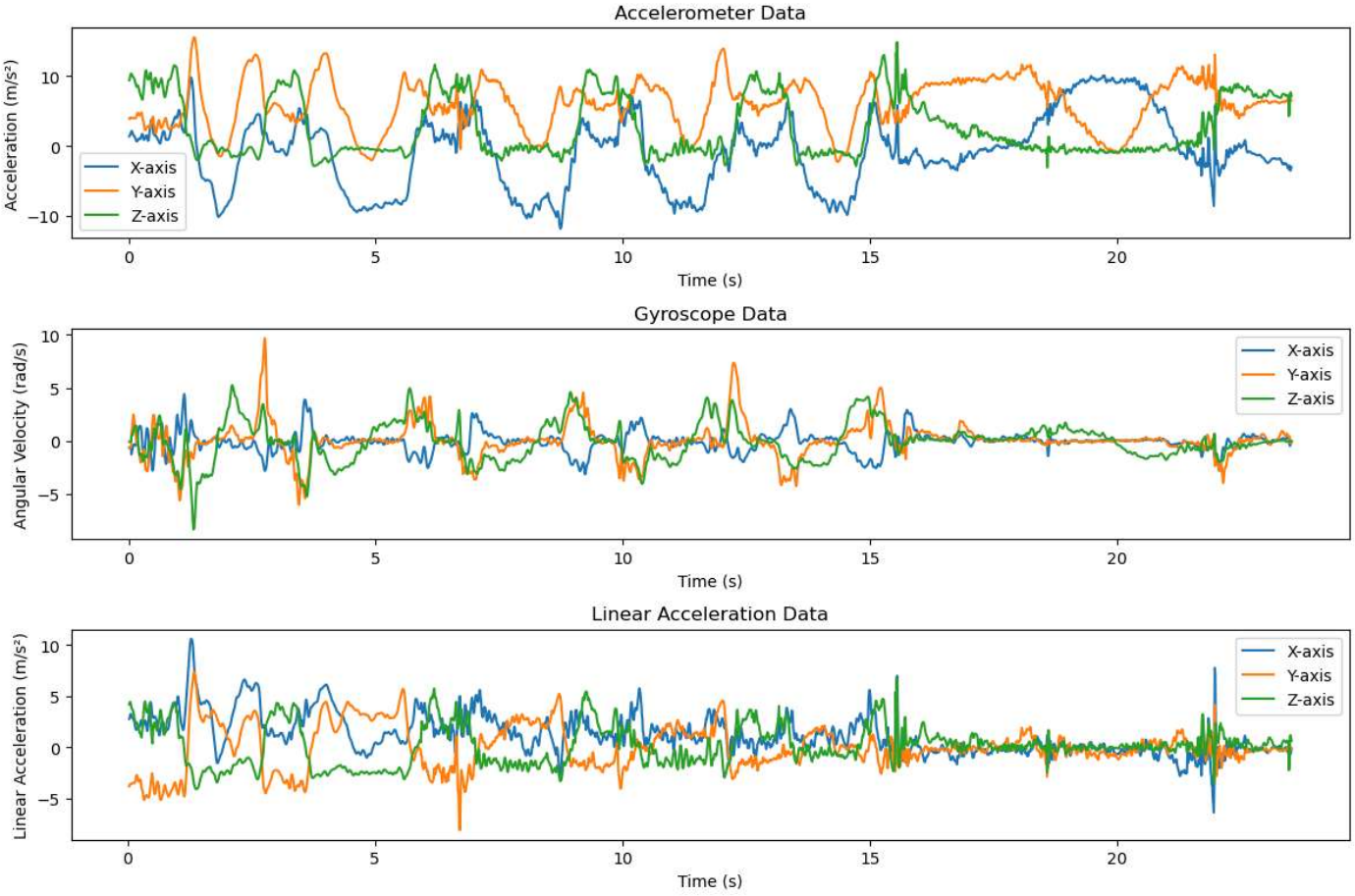
```
# Display the first few rows of each DataFrame to inspect their structure
accelerometer_df.head(), gyroscope_df.head(), linear_acceleration_df.head()
```

Out[1]:

```
(   Time (s)  Acceleration x (m/s^2)  Acceleration y (m/s^2)  \
 0  0.011976               1.367089                3.924097
 1  0.021976               1.450886                3.849876
 2  0.031976               1.608904                3.957615
 3  0.041976               1.800440                4.072537
 4  0.051976               2.013524                4.082114

    Acceleration z (m/s^2)
 0                9.411906
 1                9.682451
 2               10.015245
 3               10.273819
 4               10.374375  ,
    Time (s)  Gyroscope x (rad/s)  Gyroscope y (rad/s)  Gyroscope z (rad/s)
 0  0.011976            -0.638201            -0.616058            -0.067042
 1  0.027086            -0.638201            -0.616058            -0.067042
 2  0.037086            -0.638201            -0.616058            -0.067042
 3  0.047086            -0.638201            -0.616058            -0.067042
 4  0.057086            -1.264338             0.798095             0.465021,
    Time (s)  Linear Acceleration x (m/s^2)  Linear Acceleration y (m/s^2)  \
 0  0.031976                       2.752516                      -3.827250
 1  0.041976                       2.913037                      -3.674050
 2  0.051976                       3.094726                      -3.625845
 3  0.061976                       3.233901                      -3.619560
 4  0.071976                       3.126086                      -3.596569

    Linear Acceleration z (m/s^2)
 0                       4.162199
 1                       4.364266
 2                       4.408753
 3                       4.234228
 4                       3.971609  )
```

In [2]:

```
import matplotlib.pyplot as plt

# Plot Accelerometer Data
plt.figure(figsize=(12, 8))
plt.subplot(3, 1, 1)
plt.plot(accelerometer_df['Time (s)'], accelerometer_df['Acceleration x (m/s^2)'], label='X-axis')
plt.plot(accelerometer_df['Time (s)'], accelerometer_df['Acceleration y (m/s^2)'], label='Y-axis')
plt.plot(accelerometer_df['Time (s)'], accelerometer_df['Acceleration z (m/s^2)'], label='Z-axis')
plt.title('Accelerometer Data')
plt.xlabel('Time (s)')
plt.ylabel('Acceleration (m/s²)')
plt.legend()

# Plot Gyroscope Data
plt.subplot(3, 1, 2)
plt.plot(gyroscope_df['Time (s)'], gyroscope_df['Gyroscope x (rad/s)'], label='X-axis')
plt.plot(gyroscope_df['Time (s)'], gyroscope_df['Gyroscope y (rad/s)'], label='Y-axis')
plt.plot(gyroscope_df['Time (s)'], gyroscope_df['Gyroscope z (rad/s)'], label='Z-axis')
plt.title('Gyroscope Data')
plt.xlabel('Time (s)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()

# Plot Linear Acceleration Data
plt.subplot(3, 1, 3)
plt.plot(linear_acceleration_df['Time (s)'], linear_acceleration_df['Linear Acceleration x (m/s^2)'], labe
plt.plot(linear_acceleration_df['Time (s)'], linear_acceleration_df['Linear Acceleration y (m/s^2)'], labe
plt.plot(linear_acceleration_df['Time (s)'], linear_acceleration_df['Linear Acceleration z (m/s^2)'], labe
plt.title('Linear Acceleration Data')
plt.xlabel('Time (s)')
plt.ylabel('Linear Acceleration (m/s²)')
plt.legend()

plt.tight_layout()
plt.show()
```



Of course, this data does not have any context and is not ready to be used as input for a learning model. Given that the information comes from multiple instances of the same action in a measurement, transforming this data into a mean, standard deviation, minimum value, and maximum value will generalize each action. This will improve the comprehensiveness of the data.

4. Prepare the data to better expose the underlying data patterns to Machine Learning algorithms

prepare your data, is it normalized? are there outlier? Make a training and a test set.

At first all of the data must be collected into one file. This makes reading the data for the model a novel task. While this task is done the before mentioned mean, std, min and max are also calculated for each seperete stream of data. This results in a data size of (90, 38).

In [3]:
```python
# YOUR CODE HERE
import os
import pandas as pd

# Define the path to the root folder containing the action folders
root_folder = "datasets/testSet"

# Create an empty list to hold the summarized data
summary_data = []

# Iterate through all folders in the root folder
for folder_name in os.listdir(root_folder):
    folder_path = os.path.join(root_folder, folder_name)

    # Check if the folder_path is a directory
    if os.path.isdir(folder_path):
        # Extract person and action from the folder name
        parts = folder_name.split("-")
        if len(parts) >= 2:
```

```
person = parts[0]
action_with_index = parts[1].rsplit(" ", 1)[0]
action = ''.join([i for i in action_with_index if not i.isdigit()])

summary_row = {
    'subject': person,
    'Activity': action,

}

# Iterate through all CSV files in the folder
for file_name in os.listdir(folder_path):
    if file_name.endswith(".csv"):
        file_path = os.path.join(folder_path, file_name)
        try:
            # Read the CSV file
            data = pd.read_csv(file_path)

            # Skip empty files
            if data.empty:
                print(f"Skipping empty file: {file_path}")
                continue

            # Transfer names to insure device campatibility (Apple using different naming conv
            if {"X (m/s^2)", "Y (m/s^2)", "Z (m/s^2)"}.issubset(data.columns) and file_name ==
                data.rename(columns={
                    "X (m/s^2)": "Acceleration x (m/s^2)",
                    "Y (m/s^2)": "Acceleration y (m/s^2)",
                    "Z (m/s^2)": "Acceleration z (m/s^2)"
                }, inplace=True)
            elif {"X (rad/s)", "Y (rad/s)", "Z (rad/s)"}.issubset(data.columns):
                data.rename(columns={
                    "X (rad/s)": 'Gyroscope x (rad/s)',
                    "Y (rad/s)": 'Gyroscope y (rad/s)',
                    "Z (rad/s)": 'Gyroscope z (rad/s)'
                }, inplace=True)
            elif {"X (m/s^2)","Y (m/s^2)","Z (m/s^2)"}.issubset(data.columns):
                data.rename(columns={
                    "X (m/s^2)": 'Linear Acceleration x (m/s^2)',
                    "Y (m/s^2)": 'Linear Acceleration y (m/s^2)',
                    "Z (m/s^2)": 'Linear Acceleration z (m/s^2)'
                }, inplace=True)



            # Determine which type of data (Accelerometer, Gyroscope, or Linear Acceleration)
            if {'Acceleration x (m/s^2)', 'Acceleration y (m/s^2)', 'Acceleration z (m/s^2)'}.
                data_type = "Accelerometer"
                columns = ['Acceleration x (m/s^2)', 'Acceleration y (m/s^2)', 'Acceleration z

            elif {'Gyroscope x (rad/s)', 'Gyroscope y (rad/s)', 'Gyroscope z (rad/s)'}.issubse
                data_type = "Gyroscope"
                columns = ['Gyroscope x (rad/s)', 'Gyroscope y (rad/s)', 'Gyroscope z (rad/s)'

            elif {'Linear Acceleration x (m/s^2)', 'Linear Acceleration y (m/s^2)', 'Linear Ac
                data_type = "Linear Acceleration"
                columns = ['Linear Acceleration x (m/s^2)', 'Linear Acceleration y (m/s^2)',

            else:
                print(f"File {file_path} does not contain recognized column names. Skipping.")
                continue

            # Calculate statistics for relevant columns
            mean_values = data[columns].mean()
            std_values = data[columns].std()
            min_values = data[columns].min()
            max_values = data[columns].max()

            # add to the summary row
            for col in columns:
                summary_row[f'{col}_mean'] = mean_values[col]
                summary_row[f'{col}_std'] = std_values[col]
                summary_row[f'{col}_min'] = min_values[col]
```

```
                            summary_row[f'{col}_max'] = max_values[col]

                    except Exception as e:
                        print(f"Error reading file {file_path}: {e}")
            summary_data.append(summary_row)
# Create a dataframe from the summary data
if not summary_data:
    print("No valid data found. Summary CSV will not be created.")
else:
    summary_df = pd.DataFrame(summary_data)

    # Save the summarized data to a single CSV file
    output_path = os.path.join(root_folder, "data_total.csv")
    summary_df.to_csv(output_path, index=False)

    print(f"Summary data saved to {output_path}")

print(summary_df.shape)
```

```
Summary data saved to datasets/testSet\data_total.csv
(90, 38)
```

Due to the small sample size the test/train split is set on 80% training data.

In [4]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = 'datasets/testSet/data_total.csv'
data = pd.read_csv(file_path)

# Set your split ratio (e.g., 0.8 for 80% training, 20% testing)
split_ratio = 0.8  # Change this value as desired (0 < split_ratio < 1)

# Split the dataset into train and test sets
train_set, test_set = train_test_split(data, test_size=(1 - split_ratio), random_state=42)

# Save the splits to new CSV files (optional)
train_set.to_csv('datasets/train_set.csv', index=False)
test_set.to_csv('datasets/test_set.csv', index=False)

# Print the sizes of the splits
print(f"Train set size: {train_set.shape}")
print(f"Test set size: {test_set.shape}")
```

```
Train set size: (72, 38)
Test set size: (18, 38)
```

Now the data is ready to be checked for outliers, missing data, and other anomalies.

In [5]:
```
# import libraries
import os
import time

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
from matplotlib.colors import rgb2hex
from matplotlib.cm import get_cmap
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
from mpl_toolkits import mplot3d
from pylab import rcParams

from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression,RidgeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier, GradientBoostingClassifier
import xgboost
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import RMSprop, Adam


import warnings
warnings.filterwarnings('ignore')

sns.set(rc={'figure.facecolor':'#ffffff'})
```

In [6]:
```
# Load data
df_train = pd.read_csv('datasets/train_set.csv')
df_test = pd.read_csv('datasets/test_set.csv')
```

In [7]:
```
# shape data
print(f"Train Dataset Shape: {df_train.shape}")
print(f"Test Dataset Shape: {df_test.shape}")
```

```
Train Dataset Shape: (72, 38)
Test Dataset Shape: (18, 38)
```

In [8]:
```
# missing data counts
print(f"Train Dataset Missing Data Counts: {df_train.isna().sum().sum()}")
print(f"Test Dataset Missing Data Counts: {df_test.isna().sum().sum()}")
```

```
Train Dataset Missing Data Counts: 0
Test Dataset Missing Data Counts: 0
```

In [9]:
```
# duplicates data counts
print(f"Train Dataset Duplicate Data Counts: {df_train.duplicated().sum()}")
print(f"Test Dataset Duplicate Data Counts: {df_test.duplicated().sum()}")
```

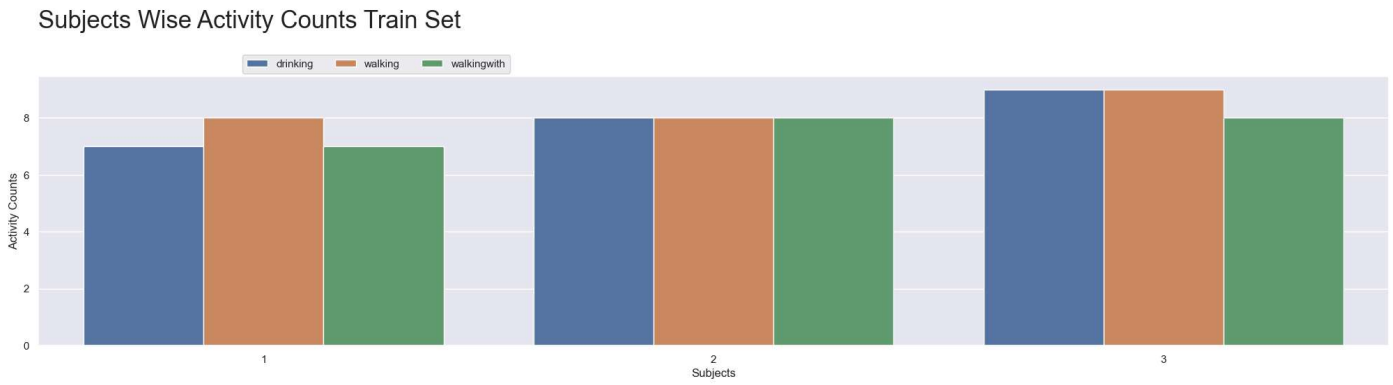
```
Train Dataset Duplicate Data Counts: 0
Test Dataset Duplicate Data Counts: 0
```

As can be seen above, no duplicates or missing data is found inside of both the train and test sets.

Now below the spread of the actions between the test and train set is plotted. This makes it possible to check if the training or testing will have blindspots for present data.

In [10]:
```
# check data balanced or not for train
plt.figure(figsize=(25, 5))
count_plot=sns.countplot(data=df_train, x='subject', hue='Activity')
plt.gca().tick_params(axis='x')
plt.gca().tick_params(axis='y')
plt.xlabel( xlabel='Subjects')
plt.ylabel( ylabel='Activity Counts')
plt.legend(["drinking", "walking", "walkingwith"],bbox_to_anchor = (0.25, 1.1), ncol=6, loc='upper center'
plt.title("Subjects Wise Activity Counts Train Set", fontsize=25, loc='left', pad=50)
plt.show()


plt.figure(figsize=(5, 5))
label_counts = df_train['Activity'].value_counts()
colors = px.colors.qualitative.Plotly

graph = go.Bar(x=label_counts.index, y=label_counts.values, marker = dict(color = colors))
```

```
layout = go.Layout(
    height=450, width=1100,
    title = 'Acticity Counts Distribution Train Set',
    xaxis = dict(title = 'Activity', tickangle=0, showgrid=False),
    yaxis = dict(title = 'Count', showgrid=False),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    title_font=dict(size=25, color='#a5a7ab', family='roboto'),
    margin=dict(t=80, b=30, l=70, r=40),
    font=dict(color='#8a8d93'))
fig = go.Figure(data=[graph], layout = layout)
fig.update_traces(textfont=dict(color='#fff'), marker=dict(line=dict(color='#ffffff', width=2)))

iplot(fig)
```




```
<Figure size 500x500 with 0 Axes>
```

In [11]:
```
# Acticity Counts Distribution For Train & Test Set using pie chart
label_counts_for_train = df_train['Activity'].value_counts()
colors = px.colors.qualitative.Plotly

label_counts_for_test = df_test['Activity'].value_counts()


fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'},{'type':'domain'}]])
fig.add_trace(go.Pie(hole=0.5, labels=label_counts_for_train.index, values=label_counts_for_train.values,
fig.add_trace(go.Pie(hole=0.5, labels=label_counts_for_test.index, values=label_counts_for_test.values, na

fig.update_layout(
#     height=450, width=1100,
    title = 'Acticity Counts Distribution For Train & Test Set',
    xaxis = dict(title = 'Activity', tickangle=0, showgrid=False),
    yaxis = dict(title = 'Count', showgrid=False),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    title_font=dict(size=25, color='#a5a7ab', family='roboto'),
#     margin=dict(t=80, b=30, l=70, r=40),
    font=dict(color='#8a8d93'))
# graph = go.Figure(data=[graph], layout = layout)
fig.update_traces(textfont=dict(color='#fff'), marker=dict(line=dict(color='#ffffff', width=2)))
fig.show()
```

The proportion between outliers and the range is also important to understand. This can be achieved with the following piece of code.

In [12]:
```
# fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'},{'type':'domain'}]])
fig = go.Figure()
fig.add_trace(go.Box(y=df_train['subject'], name="Train Set", notched=True))
fig.add_trace(go.Box( y=df_test['subject'], name="Test Set", notched=True))


fig.update_xaxes(showgrid=False)
fig.update_layout(
    title="Check Outliers In Train & Test",
    height=500, width=500,
    title_font=dict(size=25, color='#a5a7ab', family='Muli'),
    font=dict(color='#8a8d93'),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    margin=dict(t=100, b=10, l=70, r=40),
    )
```

```
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
```

The Train and Test Sets have a similar median, but the Train Set is slightly less spread out and has fewer extreme values. The Test Set shows longer whiskers, suggesting it may have a few larger or more outliers.

In [13]:
```
# Accelerometer & Gyroscope Columns Counts
lin = 0
acc = 0
gyro = 0
other = 0

for value in df_train.columns:
    if "Linear" in str(value):
        lin += 1
    elif "Acc" in str(value):
        acc += 1
    elif "Gyro" in str(value):
        gyro += 1
    else:
        other += 1

graph = go.Bar(x=['Accelerometer', 'Gyroscope', "Linear", 'Others'], y=[acc,gyro,lin,other], marker = dict
layout = go.Layout(
    height=450, width=500,
    title = 'Accelerometer & Gyroscope Columns Counts',
    xaxis = dict(title = 'Activity', tickangle=0, showgrid=False),
    yaxis = dict(title = 'Count', showgrid=False),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    title_font=dict(size=25, color='#a5a7ab', family='roboto'),
    margin=dict(t=80, b=30, l=70, r=40),
    font=dict(color='#8a8d93'))
fig = go.Figure(data=[graph], layout = layout)
fig.update_traces(textfont=dict(color='#fff'), marker=dict(line=dict(color='#ffffff', width=2)))

iplot(fig)
```

in this case the "others" are activity and person performing the action

Below we will check the correlation matrices for the three different activities

In [14]:
```
corl = df_train.drop(['subject'], axis=1)
encoded_columns = pd.get_dummies(corl['Activity'])
corl = pd.concat([encoded_columns, corl.drop(columns=['Activity'])], axis=1)
corr_matrix = corl.corr()
corr_matrix["walking"].sort_values(ascending=False)
```

Out[14]:
```
walking                              1.000000
Linear Acceleration y (m/s^2)_std    0.854779
Linear Acceleration x (m/s^2)_std    0.838585
Linear Acceleration x (m/s^2)_max    0.801277
Gyroscope x (rad/s)_std              0.748054
Gyroscope y (rad/s)_std              0.743683
Gyroscope z (rad/s)_std              0.715002
Acceleration x (m/s^2)_max           0.677213
Gyroscope x (rad/s)_max              0.586792
Acceleration x (m/s^2)_mean          0.576655
Gyroscope z (rad/s)_max              0.493195
Gyroscope y (rad/s)_max              0.484486
Linear Acceleration z (m/s^2)_max    0.373078
Linear Acceleration x (m/s^2)_mean   0.324364
Linear Acceleration z (m/s^2)_mean   0.318132
Acceleration y (m/s^2)_std           0.308860
Linear Acceleration y (m/s^2)_max    0.283337
Acceleration x (m/s^2)_std           0.227211
Acceleration x (m/s^2)_min           0.155995
Linear Acceleration z (m/s^2)_std    0.151835
Acceleration z (m/s^2)_std           0.000855
Acceleration z (m/s^2)_min          -0.037099
Acceleration z (m/s^2)_max          -0.076040
Gyroscope x (rad/s)_mean            -0.096662
Gyroscope z (rad/s)_mean            -0.160348
Gyroscope y (rad/s)_mean            -0.264225
Acceleration y (m/s^2)_max          -0.281167
Linear Acceleration z (m/s^2)_min   -0.312775
Acceleration z (m/s^2)_mean         -0.343631
walkingwith                         -0.484452
drinking                            -0.515711
Gyroscope x (rad/s)_min             -0.518043
Gyroscope y (rad/s)_min             -0.521690
Linear Acceleration x (m/s^2)_min   -0.615862
Gyroscope z (rad/s)_min             -0.678160
Linear Acceleration y (m/s^2)_mean  -0.791007
Acceleration y (m/s^2)_min          -0.838996
Linear Acceleration y (m/s^2)_min   -0.856651
Acceleration y (m/s^2)_mean         -0.935236
Name: walking, dtype: float64
```

In [15]:
```
corr_matrix["walkingwith"].sort_values(ascending=False)
```

Out[15]:
```
walkingwith                          1.000000
Acceleration z (m/s^2)_mean          0.934921
Linear Acceleration z (m/s^2)_std    0.613086
Gyroscope z (rad/s)_min              0.591595
Gyroscope y (rad/s)_min              0.565143
Acceleration y (m/s^2)_min           0.556409
Gyroscope x (rad/s)_min              0.421871
Acceleration z (m/s^2)_min           0.413521
Linear Acceleration y (m/s^2)_min    0.379359
Acceleration z (m/s^2)_max           0.312413
Linear Acceleration z (m/s^2)_max    0.310240
Linear Acceleration x (m/s^2)_min    0.307972
Acceleration x (m/s^2)_min           0.284386
Acceleration y (m/s^2)_mean          0.274520
Linear Acceleration y (m/s^2)_mean   0.247314
Acceleration z (m/s^2)_std           0.234122
Gyroscope z (rad/s)_mean             0.186151
Gyroscope y (rad/s)_mean             0.121785
Gyroscope x (rad/s)_mean             0.100424
Linear Acceleration z (m/s^2)_mean   0.051620
Linear Acceleration z (m/s^2)_min    0.011427
Linear Acceleration x (m/s^2)_mean  -0.108232
Acceleration x (m/s^2)_mean         -0.209603
Acceleration y (m/s^2)_max          -0.244157
Linear Acceleration y (m/s^2)_std   -0.320368
Linear Acceleration x (m/s^2)_std   -0.327485
Gyroscope x (rad/s)_std             -0.329317
Gyroscope x (rad/s)_max             -0.381147
Linear Acceleration y (m/s^2)_max   -0.396848
Linear Acceleration x (m/s^2)_max   -0.401828
walking                             -0.484452
drinking                            -0.499674
Gyroscope y (rad/s)_std             -0.517311
Gyroscope z (rad/s)_std             -0.536470
Gyroscope z (rad/s)_max             -0.544859
Acceleration x (m/s^2)_max          -0.589463
Gyroscope y (rad/s)_max             -0.592359
Acceleration y (m/s^2)_std          -0.783196
Acceleration x (m/s^2)_std          -0.808801
Name: walkingwith, dtype: float64
```

In [16]:
```
corr_matrix["drinking"].sort_values(ascending=False)
```

Out[16]:
```
drinking                              1.000000
Acceleration y (m/s^2)_mean           0.657184
Acceleration x (m/s^2)_std            0.567132
Linear Acceleration y (m/s^2)_mean    0.541017
Acceleration y (m/s^2)_max            0.517519
Linear Acceleration y (m/s^2)_min     0.476696
Acceleration y (m/s^2)_std            0.461210
Linear Acceleration x (m/s^2)_min     0.308189
Linear Acceleration z (m/s^2)_min     0.298507
Acceleration y (m/s^2)_min            0.285818
Gyroscope y (rad/s)_mean              0.142355
Linear Acceleration y (m/s^2)_max     0.108108
Gyroscope y (rad/s)_max               0.100413
Gyroscope x (rad/s)_min               0.099784
Gyroscope z (rad/s)_min               0.092104
Gyroscope z (rad/s)_max               0.045270
Gyroscope x (rad/s)_mean             -0.002640
Gyroscope z (rad/s)_mean             -0.023538
Gyroscope y (rad/s)_min              -0.036921
Acceleration x (m/s^2)_max           -0.093255
Gyroscope z (rad/s)_std              -0.182572
Gyroscope x (rad/s)_max              -0.207740
Linear Acceleration x (m/s^2)_mean   -0.215175
Gyroscope y (rad/s)_std              -0.229734
Acceleration z (m/s^2)_std           -0.230137
Acceleration z (m/s^2)_max           -0.230674
Linear Acceleration z (m/s^2)_mean   -0.365558
Acceleration x (m/s^2)_mean          -0.365706
Acceleration z (m/s^2)_min           -0.368253
Linear Acceleration x (m/s^2)_max    -0.399861
Gyroscope x (rad/s)_std              -0.418176
Acceleration x (m/s^2)_min           -0.432977
walkingwith                          -0.499674
Linear Acceleration x (m/s^2)_std    -0.509611
walking                              -0.515711
Linear Acceleration y (m/s^2)_std    -0.532616
Acceleration z (m/s^2)_mean          -0.575375
Linear Acceleration z (m/s^2)_max    -0.673245
Linear Acceleration z (m/s^2)_std    -0.750774
Name: drinking, dtype: float64
```

in the corrolation matrixes you can see that some features are more important for some of the activities and that all of the features are used
no anomolies are spotted

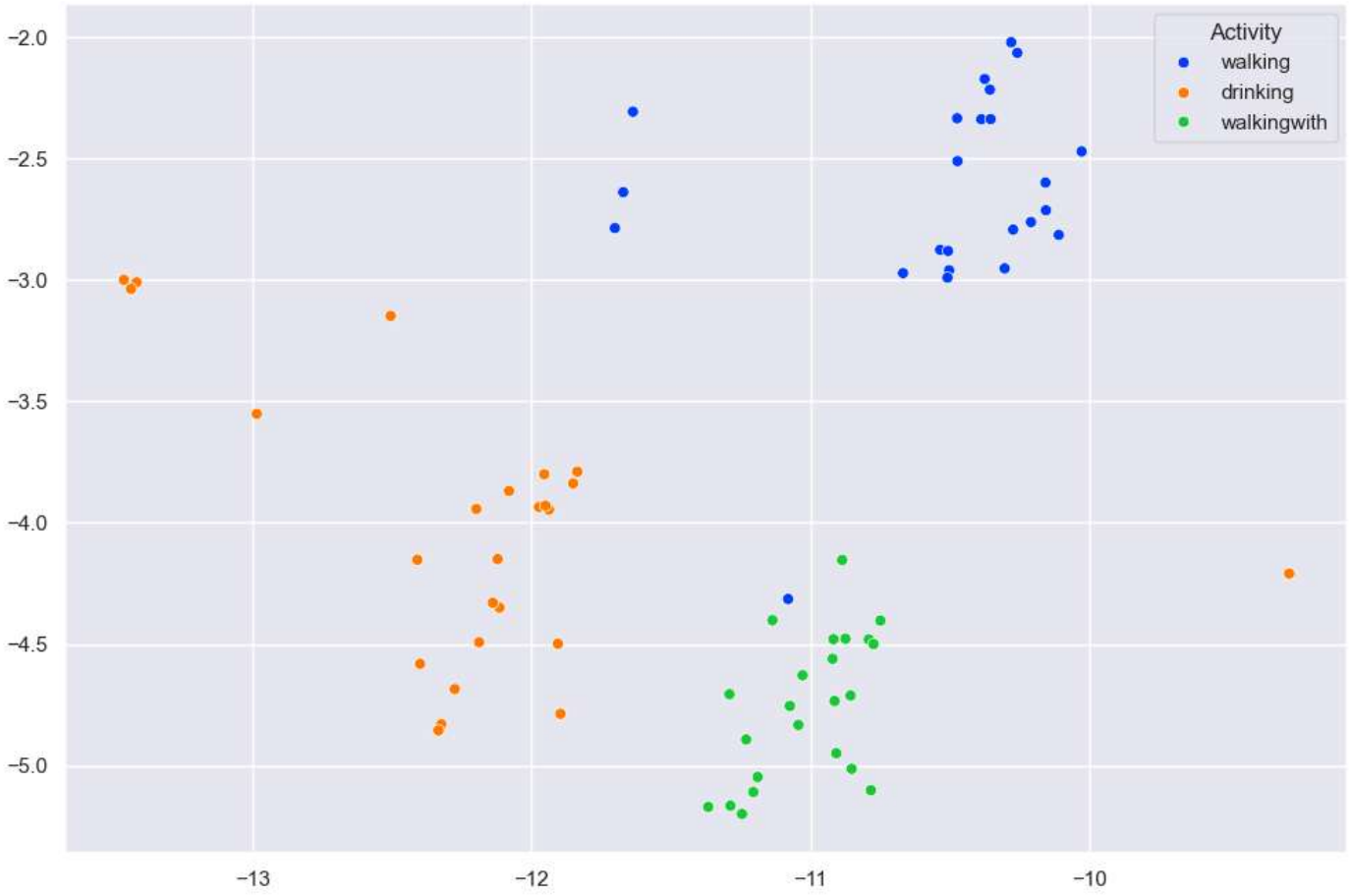
In [17]:
```
# update columns name
columns_name = df_train.columns

columns_name = columns_name.str.replace('[()]','')
columns_name = columns_name.str.replace('[-]', '')
columns_name = columns_name.str.replace('[,]','')

df_train.columns = columns_name
df_train.columns = columns_name
```

t-SNE (t-distributed Stochastic Neighbor Embedding) is used to reduce the dimentions (in this case determined by the features)
t-SNE preserves the local relationships (similarity) between points in the original multidimentional space
we do this to be able to easily view the similiarity of our data

In [18]:
```
# t-sne (2D)
x_for_tsne = df_train.drop(['subject', 'Activity'], axis=1)

tsne = TSNE(random_state = 42, n_components=2, verbose=1, perplexity=50, n_iter=1000).fit_transform(x_for_
plt.figure(figsize=(12,8))
sns.scatterplot(x =tsne[:, 0], y = tsne[:, 1], hue = df_train["Activity"],palette="bright")
```

```
[t-SNE] Computing 71 nearest neighbors...
[t-SNE] Indexed 72 samples in 0.001s...
[t-SNE] Computed neighbors for 72 samples in 0.144s...
[t-SNE] Computed conditional probabilities for sample 72 / 72
[t-SNE] Mean sigma: 21.202668
[t-SNE] KL divergence after 250 iterations with early exaggeration: 44.459499
[t-SNE] KL divergence after 1000 iterations: 0.025495
```

Out[18]: <Axes: >



this looks very promesing seeing nice clusters form

And here we do the same but then to go to 3d space

In [19]:
```
# t-sne (3D)
x_for_tsne = df_train.drop(['subject', 'Activity'], axis=1)

tsne = TSNE(random_state = 42, n_components=3, verbose=1, perplexity=45, n_iter=1000).fit_transform(x_for_
fig = px.scatter_3d(
    x =tsne[:, 0],
    y = tsne[:, 1],
    z = tsne[:, 2],
    color=df_train['Activity']
)
fig.update_layout(
    title="Cluster Of Activities",
    title_font=dict(size=25, color='#a5a7ab', family='roboto'),
    font=dict(color='#8a8d93'),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    margin=dict(t=100, b=10, l=70, r=40),
    )
fig.show()
```

```
[t-SNE] Computing 71 nearest neighbors...
[t-SNE] Indexed 72 samples in 0.000s...
[t-SNE] Computed neighbors for 72 samples in 0.005s...
[t-SNE] Computed conditional probabilities for sample 72 / 72
[t-SNE] Mean sigma: 18.908332
[t-SNE] KL divergence after 250 iterations with early exaggeration: 98.980064
[t-SNE] KL divergence after 1000 iterations: 0.976722
```

its more difficult to see the clusters here but they are still present

here we drop the subject feature because its useless for training and the activity gets put into the label dataframe
then the same is done for the test set

In [20]:
```
X_train, y_train = df_train.drop(['subject', 'Activity'], axis=1), df_train['Activity']
X_test, y_test = df_test.drop(['subject', 'Activity'], axis=1), df_test['Activity']
```

the following piece of code reduces the the amount of features to the ones that contain 95% of the usefull information

In [21]:
```
#pca with 95% info
pca = PCA(0.95)

pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
print(X_test.shape)
```

```
(18, 6)
```

In [22]:
```
encoder = LabelEncoder()
y_train=encoder.fit_transform(y_train)
y_test=encoder.fit_transform(y_test)
```

The function below is to plot a confusion matrix (this was provided in the example code)

In [23]:
```
# function to plot confusion matrix
def plot_confusion_matrix(cm,lables):
    fig, ax = plt.subplots(figsize=(15,5)) # for plotting confusion matrix as image
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax.figure.colorbar(im, ax=ax)
    ax.set(xticks=np.arange(cm.shape[1]),
    yticks=np.arange(cm.shape[0]),
    xticklabels=lables, yticklabels=lables,
    ylabel='True label',
    xlabel='Predicted label')
    plt.xticks(rotation = 90)
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, int(cm[i, j]),ha="center", va="center",color="white" if cm[i, j] > thresh else '
    fig.tight_layout()
```

5. Explore many different models and short-list the best ones.

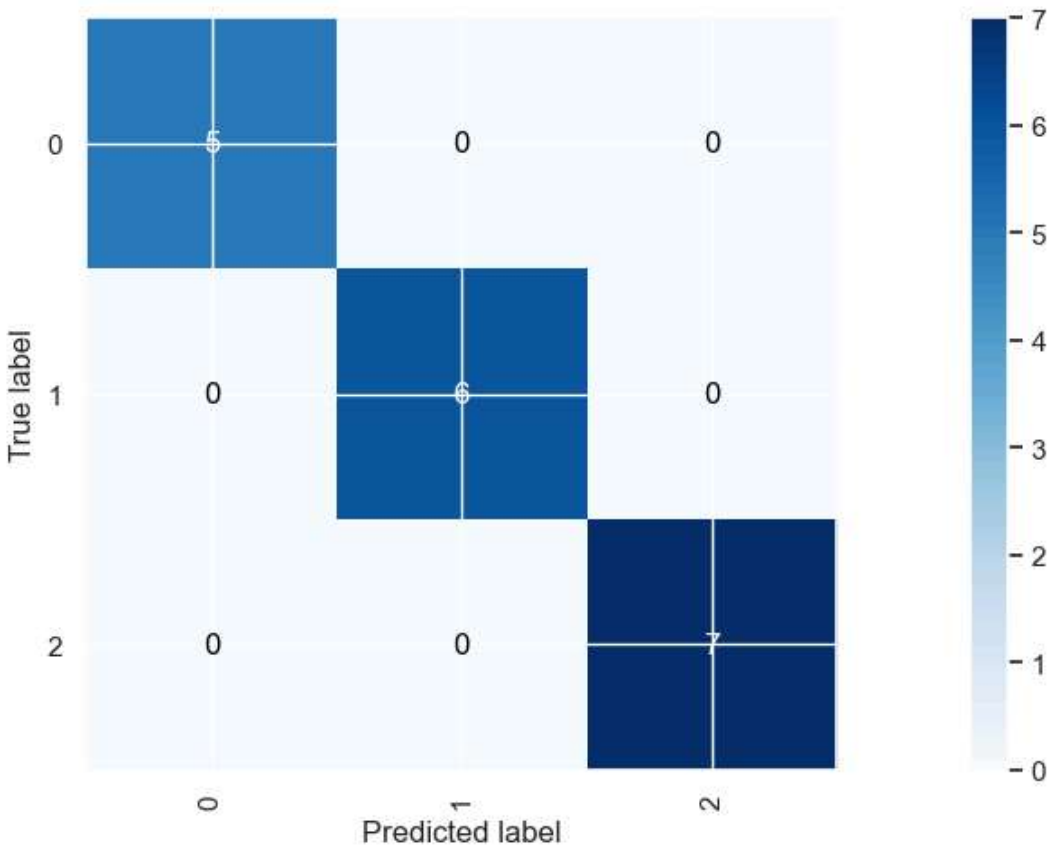
Explore / train and list the top 3 algorithms that score best on this dataset.

here we will start with some different models to check their effectiveness
starting with logistic regression

In [24]:
```
# logistic regresssion
lr_model = LogisticRegression()
# c = regulisation
#
#
#
#
#
params = {
```

```
    'C': np.arange(10,61,10),
    'penalty': ['l2','l1']
}

random_cv = RandomizedSearchCV(lr_model, param_distributions=params, cv=5, random_state=42)
random_cv.fit(X_train, y_train)
lr_model = LogisticRegression(penalty='l2', C=1,solver='lbfgs',class_weight='balanced', max_iter=1,random_
lr_model.fit(X_train, y_train)
y_pred = lr_model.predict(X_test)

lr_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Of Logistic Regression : ", lr_accuracy)
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
Accuracy Of Logistic Regression :  1.0
```



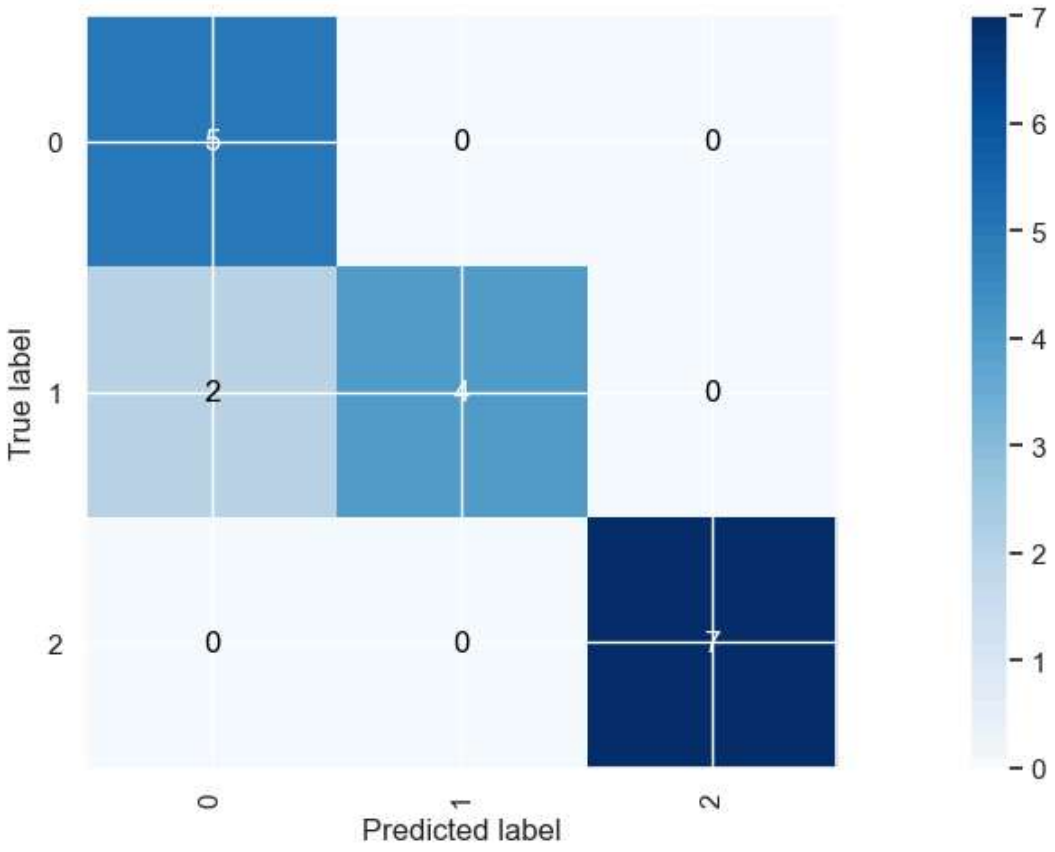
the part above calls upon the confusion matrix function and plots the results

here we use the SVC model

In [25]:
```
# svc
params = {
    'C':[2,4,8,16],
    'gamma': [0.125, 0.250, 0.5, 1]
}
svm_model = SVC(kernel='rbf')

random_cv = RandomizedSearchCV(svm_model, param_distributions=params, random_state = 42, verbose=10)
random_cv.fit(X_train, y_train)
random_cv.best_params_
svc_model = SVC(gamma=0.125, C=2)
svc_model.fit(X_train, y_train)
y_pred = svc_model.predict(X_test)

svc_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Of Logistic Regression : ", svc_accuracy)
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5; 1/10] START C=2, gamma=0.125.....................................
[CV 1/5; 1/10] END ............C=2, gamma=0.125;, score=0.933 total time=   0.0s
[CV 2/5; 1/10] START C=2, gamma=0.125.....................................
[CV 2/5; 1/10] END ............C=2, gamma=0.125;, score=0.733 total time=   0.0s
[CV 3/5; 1/10] START C=2, gamma=0.125.....................................
[CV 3/5; 1/10] END ............C=2, gamma=0.125;, score=0.857 total time=   0.0s
[CV 4/5; 1/10] START C=2, gamma=0.125.....................................
[CV 4/5; 1/10] END ............C=2, gamma=0.125;, score=0.714 total time=   0.0s
[CV 5/5; 1/10] START C=2, gamma=0.125.....................................
[CV 5/5; 1/10] END ............C=2, gamma=0.125;, score=1.000 total time=   0.0s
[CV 1/5; 2/10] START C=2, gamma=0.25......................................
[CV 1/5; 2/10] END ............C=2, gamma=0.25;, score=0.867 total time=   0.0s
[CV 2/5; 2/10] START C=2, gamma=0.25......................................
[CV 2/5; 2/10] END ............C=2, gamma=0.25;, score=0.600 total time=   0.0s
[CV 3/5; 2/10] START C=2, gamma=0.25......................................
[CV 3/5; 2/10] END ............C=2, gamma=0.25;, score=0.643 total time=   0.0s
[CV 4/5; 2/10] START C=2, gamma=0.25......................................
[CV 4/5; 2/10] END ............C=2, gamma=0.25;, score=0.643 total time=   0.0s
[CV 5/5; 2/10] START C=2, gamma=0.25......................................
[CV 5/5; 2/10] END ............C=2, gamma=0.25;, score=0.714 total time=   0.0s
[CV 1/5; 3/10] START C=4, gamma=0.25......................................
[CV 1/5; 3/10] END ............C=4, gamma=0.25;, score=0.867 total time=   0.0s
[CV 2/5; 3/10] START C=4, gamma=0.25......................................
[CV 2/5; 3/10] END ............C=4, gamma=0.25;, score=0.600 total time=   0.0s
[CV 3/5; 3/10] START C=4, gamma=0.25......................................
[CV 3/5; 3/10] END ............C=4, gamma=0.25;, score=0.643 total time=   0.0s
[CV 4/5; 3/10] START C=4, gamma=0.25......................................
[CV 4/5; 3/10] END ............C=4, gamma=0.25;, score=0.643 total time=   0.0s
[CV 5/5; 3/10] START C=4, gamma=0.25......................................
[CV 5/5; 3/10] END ............C=4, gamma=0.25;, score=0.714 total time=   0.0s
[CV 1/5; 4/10] START C=16, gamma=0.5......................................
[CV 1/5; 4/10] END ............C=16, gamma=0.5;, score=0.600 total time=   0.0s
[CV 2/5; 4/10] START C=16, gamma=0.5......................................
[CV 2/5; 4/10] END ............C=16, gamma=0.5;, score=0.533 total time=   0.0s
[CV 3/5; 4/10] START C=16, gamma=0.5......................................
[CV 3/5; 4/10] END ............C=16, gamma=0.5;, score=0.429 total time=   0.0s
[CV 4/5; 4/10] START C=16, gamma=0.5......................................
[CV 4/5; 4/10] END ............C=16, gamma=0.5;, score=0.571 total time=   0.0s
[CV 5/5; 4/10] START C=16, gamma=0.5......................................
[CV 5/5; 4/10] END ............C=16, gamma=0.5;, score=0.643 total time=   0.0s
[CV 1/5; 5/10] START C=16, gamma=0.25.....................................
[CV 1/5; 5/10] END ............C=16, gamma=0.25;, score=0.867 total time=   0.0s
[CV 2/5; 5/10] START C=16, gamma=0.25.....................................
[CV 2/5; 5/10] END ............C=16, gamma=0.25;, score=0.600 total time=   0.0s
[CV 3/5; 5/10] START C=16, gamma=0.25.....................................
[CV 3/5; 5/10] END ............C=16, gamma=0.25;, score=0.643 total time=   0.0s
[CV 4/5; 5/10] START C=16, gamma=0.25.....................................
[CV 4/5; 5/10] END ............C=16, gamma=0.25;, score=0.643 total time=   0.0s
[CV 5/5; 5/10] START C=16, gamma=0.25.....................................
[CV 5/5; 5/10] END ............C=16, gamma=0.25;, score=0.714 total time=   0.0s
[CV 1/5; 6/10] START C=8, gamma=1.........................................
[CV 1/5; 6/10] END ..............C=8, gamma=1;, score=0.400 total time=   0.0s
[CV 2/5; 6/10] START C=8, gamma=1.........................................
[CV 2/5; 6/10] END ..............C=8, gamma=1;, score=0.467 total time=   0.0s
[CV 3/5; 6/10] START C=8, gamma=1.........................................
[CV 3/5; 6/10] END ..............C=8, gamma=1;, score=0.357 total time=   0.0s
[CV 4/5; 6/10] START C=8, gamma=1.........................................
[CV 4/5; 6/10] END ..............C=8, gamma=1;, score=0.500 total time=   0.0s
[CV 5/5; 6/10] START C=8, gamma=1.........................................
[CV 5/5; 6/10] END ..............C=8, gamma=1;, score=0.500 total time=   0.0s
[CV 1/5; 7/10] START C=8, gamma=0.125.....................................
[CV 1/5; 7/10] END ............C=8, gamma=0.125;, score=0.933 total time=   0.0s
[CV 2/5; 7/10] START C=8, gamma=0.125.....................................
[CV 2/5; 7/10] END ............C=8, gamma=0.125;, score=0.733 total time=   0.0s
[CV 3/5; 7/10] START C=8, gamma=0.125.....................................
[CV 3/5; 7/10] END ............C=8, gamma=0.125;, score=0.857 total time=   0.0s
[CV 4/5; 7/10] START C=8, gamma=0.125.....................................
[CV 4/5; 7/10] END ............C=8, gamma=0.125;, score=0.714 total time=   0.0s
[CV 5/5; 7/10] START C=8, gamma=0.125.....................................
[CV 5/5; 7/10] END ............C=8, gamma=0.125;, score=1.000 total time=   0.0s
[CV 1/5; 8/10] START C=8, gamma=0.25......................................
[CV 1/5; 8/10] END ............C=8, gamma=0.25;, score=0.867 total time=   0.0s
```

```
[CV 2/5; 8/10] START C=8, gamma=0.25.....................................
[CV 2/5; 8/10] END .............C=8, gamma=0.25;, score=0.600 total time=   0.0s
[CV 3/5; 8/10] START C=8, gamma=0.25.....................................
[CV 3/5; 8/10] END .............C=8, gamma=0.25;, score=0.643 total time=   0.0s
[CV 4/5; 8/10] START C=8, gamma=0.25.....................................
[CV 4/5; 8/10] END .............C=8, gamma=0.25;, score=0.643 total time=   0.0s
[CV 5/5; 8/10] START C=8, gamma=0.25.....................................
[CV 5/5; 8/10] END .............C=8, gamma=0.25;, score=0.714 total time=   0.0s
[CV 1/5; 9/10] START C=2, gamma=0.5......................................
[CV 1/5; 9/10] END .............C=2, gamma=0.5;, score=0.600 total time=   0.0s
[CV 2/5; 9/10] START C=2, gamma=0.5......................................
[CV 2/5; 9/10] END .............C=2, gamma=0.5;, score=0.533 total time=   0.0s
[CV 3/5; 9/10] START C=2, gamma=0.5......................................
[CV 3/5; 9/10] END .............C=2, gamma=0.5;, score=0.429 total time=   0.0s
[CV 4/5; 9/10] START C=2, gamma=0.5......................................
[CV 4/5; 9/10] END .............C=2, gamma=0.5;, score=0.571 total time=   0.0s
[CV 5/5; 9/10] START C=2, gamma=0.5......................................
[CV 5/5; 9/10] END .............C=2, gamma=0.5;, score=0.643 total time=   0.0s
[CV 1/5; 10/10] START C=16, gamma=1......................................
[CV 1/5; 10/10] END .............C=16, gamma=1;, score=0.400 total time=   0.0s
[CV 2/5; 10/10] START C=16, gamma=1......................................
[CV 2/5; 10/10] END .............C=16, gamma=1;, score=0.467 total time=   0.0s
[CV 3/5; 10/10] START C=16, gamma=1......................................
[CV 3/5; 10/10] END .............C=16, gamma=1;, score=0.357 total time=   0.0s
[CV 4/5; 10/10] START C=16, gamma=1......................................
[CV 4/5; 10/10] END .............C=16, gamma=1;, score=0.500 total time=   0.0s
[CV 5/5; 10/10] START C=16, gamma=1......................................
[CV 5/5; 10/10] END .............C=16, gamma=1;, score=0.500 total time=   0.0s
Accuracy Of Logistic Regression :  0.8888888888888888
```

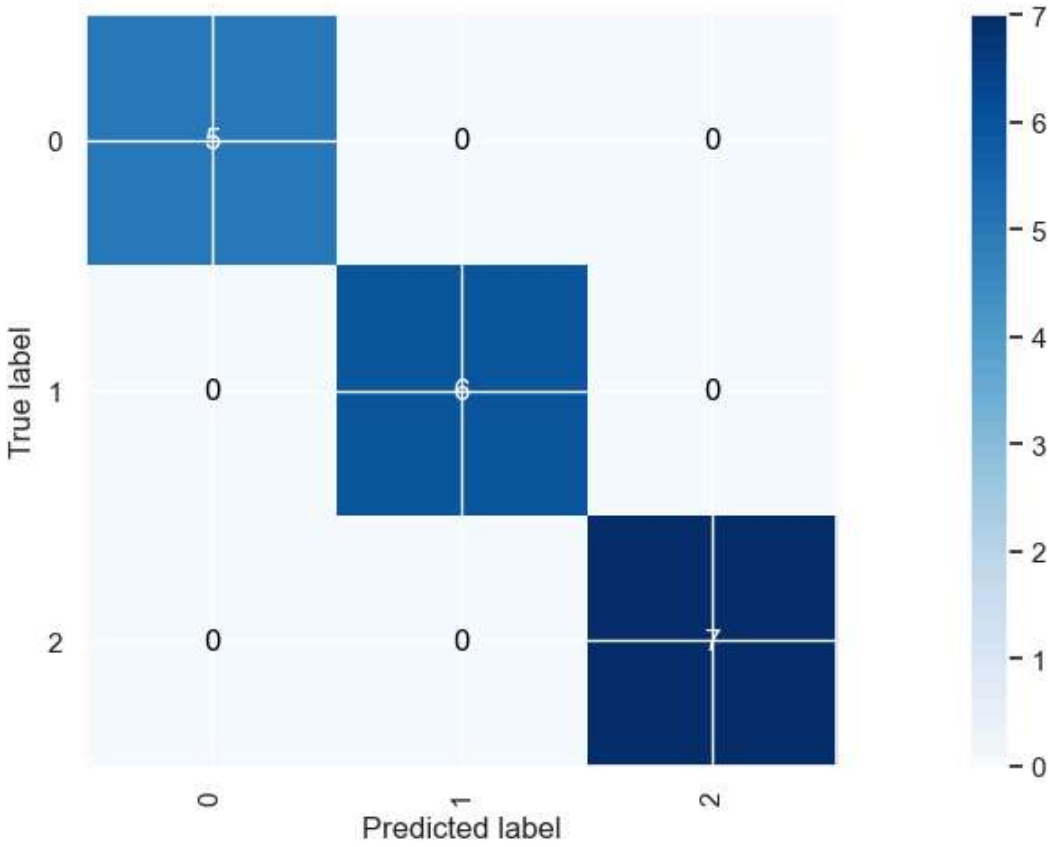


here we use the decision tree model

In [26]:
```
# decision tree
params = {'max_depth':np.arange(2,10,2)}

dt_model = DecisionTreeClassifier()
random_cv = RandomizedSearchCV(dt_model, param_distributions=params, random_state = 42)
random_cv.fit(X_train, y_train)
random_cv.best_params_
dt_model = DecisionTreeClassifier(max_depth=6)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)
```

```
dt_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Of Logistic Regression : ", dt_accuracy)
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
Accuracy Of Logistic Regression :  1.0
```



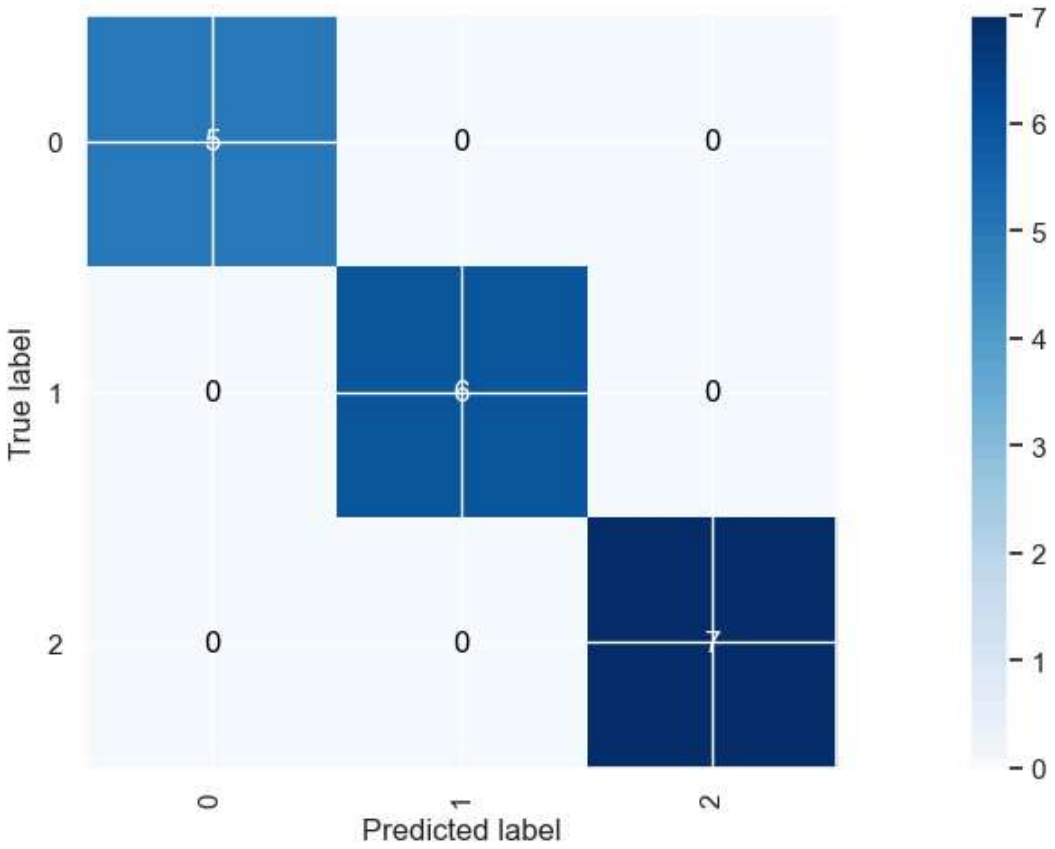
here we use the random forest clasifier model

In [27]:
```
# random forest
params = {
    'n_estimators': np.arange(20,101,10),
    'max_depth':np.arange(2,16,2),
    'max_features':['auto', 'sqrt'],
    'bootstrap':[True, False]
}
rf_model = RandomForestClassifier()
random_cv = RandomizedSearchCV(rf_model, param_distributions=params,random_state = 42, verbose=10)
random_cv.fit(X_train, y_train)
random_cv.best_params_
rf_model = RandomForestClassifier(n_estimators=90,
                                  max_features=56,
                                  max_depth=14,
                                  bootstrap=True)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

rf_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Of Logistic Regression : ", rf_accuracy)
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5; 1/10] START bootstrap=False, max_depth=6, max_features=auto, n_estimators=50
[CV 1/5; 1/10] END bootstrap=False, max_depth=6, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 2/5; 1/10] START bootstrap=False, max_depth=6, max_features=auto, n_estimators=50
[CV 2/5; 1/10] END bootstrap=False, max_depth=6, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 3/5; 1/10] START bootstrap=False, max_depth=6, max_features=auto, n_estimators=50
[CV 3/5; 1/10] END bootstrap=False, max_depth=6, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 4/5; 1/10] START bootstrap=False, max_depth=6, max_features=auto, n_estimators=50
[CV 4/5; 1/10] END bootstrap=False, max_depth=6, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 5/5; 1/10] START bootstrap=False, max_depth=6, max_features=auto, n_estimators=50
[CV 5/5; 1/10] END bootstrap=False, max_depth=6, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 1/5; 2/10] START bootstrap=True, max_depth=2, max_features=auto, n_estimators=80
[CV 1/5; 2/10] END bootstrap=True, max_depth=2, max_features=auto, n_estimators=80;, score=nan total time=
0.0s
[CV 2/5; 2/10] START bootstrap=True, max_depth=2, max_features=auto, n_estimators=80
[CV 2/5; 2/10] END bootstrap=True, max_depth=2, max_features=auto, n_estimators=80;, score=nan total time=
0.0s
[CV 3/5; 2/10] START bootstrap=True, max_depth=2, max_features=auto, n_estimators=80
[CV 3/5; 2/10] END bootstrap=True, max_depth=2, max_features=auto, n_estimators=80;, score=nan total time=
0.0s
[CV 4/5; 2/10] START bootstrap=True, max_depth=2, max_features=auto, n_estimators=80
[CV 4/5; 2/10] END bootstrap=True, max_depth=2, max_features=auto, n_estimators=80;, score=nan total time=
0.0s
[CV 5/5; 2/10] START bootstrap=True, max_depth=2, max_features=auto, n_estimators=80
[CV 5/5; 2/10] END bootstrap=True, max_depth=2, max_features=auto, n_estimators=80;, score=nan total time=
0.0s
[CV 1/5; 3/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=50
[CV 1/5; 3/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 2/5; 3/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=50
[CV 2/5; 3/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 3/5; 3/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=50
[CV 3/5; 3/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 4/5; 3/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=50
[CV 4/5; 3/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 5/5; 3/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=50
[CV 5/5; 3/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 1/5; 4/10] START bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30
[CV 1/5; 4/10] END bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30;, score=0.933 total tim
e=   0.0s
[CV 2/5; 4/10] START bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30
[CV 2/5; 4/10] END bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30;, score=1.000 total tim
e=   0.0s
[CV 3/5; 4/10] START bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30
[CV 3/5; 4/10] END bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30;, score=1.000 total tim
e=   0.0s
[CV 4/5; 4/10] START bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30
[CV 4/5; 4/10] END bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30;, score=0.857 total tim
e=   0.0s
[CV 5/5; 4/10] START bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30
[CV 5/5; 4/10] END bootstrap=False, max_depth=6, max_features=sqrt, n_estimators=30;, score=1.000 total tim
e=   0.0s
[CV 1/5; 5/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=90
[CV 1/5; 5/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=90;, score=nan total time=
0.0s
[CV 2/5; 5/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=90
[CV 2/5; 5/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=90;, score=nan total time=
0.0s
[CV 3/5; 5/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=90
[CV 3/5; 5/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=90;, score=nan total time=
0.0s
[CV 4/5; 5/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=90
[CV 4/5; 5/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=90;, score=nan total time=
0.0s
```

```
[CV 5/5; 5/10] START bootstrap=True, max_depth=14, max_features=auto, n_estimators=90
[CV 5/5; 5/10] END bootstrap=True, max_depth=14, max_features=auto, n_estimators=90;, score=nan total time=
0.0s
[CV 1/5; 6/10] START bootstrap=False, max_depth=8, max_features=auto, n_estimators=50
[CV 1/5; 6/10] END bootstrap=False, max_depth=8, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 2/5; 6/10] START bootstrap=False, max_depth=8, max_features=auto, n_estimators=50
[CV 2/5; 6/10] END bootstrap=False, max_depth=8, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 3/5; 6/10] START bootstrap=False, max_depth=8, max_features=auto, n_estimators=50
[CV 3/5; 6/10] END bootstrap=False, max_depth=8, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 4/5; 6/10] START bootstrap=False, max_depth=8, max_features=auto, n_estimators=50
[CV 4/5; 6/10] END bootstrap=False, max_depth=8, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 5/5; 6/10] START bootstrap=False, max_depth=8, max_features=auto, n_estimators=50
[CV 5/5; 6/10] END bootstrap=False, max_depth=8, max_features=auto, n_estimators=50;, score=nan total time=
0.0s
[CV 1/5; 7/10] START bootstrap=False, max_depth=10, max_features=auto, n_estimators=30
[CV 1/5; 7/10] END bootstrap=False, max_depth=10, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 2/5; 7/10] START bootstrap=False, max_depth=10, max_features=auto, n_estimators=30
[CV 2/5; 7/10] END bootstrap=False, max_depth=10, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 3/5; 7/10] START bootstrap=False, max_depth=10, max_features=auto, n_estimators=30
[CV 3/5; 7/10] END bootstrap=False, max_depth=10, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 4/5; 7/10] START bootstrap=False, max_depth=10, max_features=auto, n_estimators=30
[CV 4/5; 7/10] END bootstrap=False, max_depth=10, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 5/5; 7/10] START bootstrap=False, max_depth=10, max_features=auto, n_estimators=30
[CV 5/5; 7/10] END bootstrap=False, max_depth=10, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 1/5; 8/10] START bootstrap=False, max_depth=14, max_features=auto, n_estimators=30
[CV 1/5; 8/10] END bootstrap=False, max_depth=14, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 2/5; 8/10] START bootstrap=False, max_depth=14, max_features=auto, n_estimators=30
[CV 2/5; 8/10] END bootstrap=False, max_depth=14, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 3/5; 8/10] START bootstrap=False, max_depth=14, max_features=auto, n_estimators=30
[CV 3/5; 8/10] END bootstrap=False, max_depth=14, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 4/5; 8/10] START bootstrap=False, max_depth=14, max_features=auto, n_estimators=30
[CV 4/5; 8/10] END bootstrap=False, max_depth=14, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 5/5; 8/10] START bootstrap=False, max_depth=14, max_features=auto, n_estimators=30
[CV 5/5; 8/10] END bootstrap=False, max_depth=14, max_features=auto, n_estimators=30;, score=nan total time
=   0.0s
[CV 1/5; 9/10] START bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20
[CV 1/5; 9/10] END bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20;, score=0.800 total time
=   0.0s
[CV 2/5; 9/10] START bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20
[CV 2/5; 9/10] END bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20;, score=1.000 total time
=   0.0s
[CV 3/5; 9/10] START bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20
[CV 3/5; 9/10] END bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20;, score=1.000 total time
=   0.0s
[CV 4/5; 9/10] START bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20
[CV 4/5; 9/10] END bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20;, score=0.786 total time
=   0.0s
[CV 5/5; 9/10] START bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20
[CV 5/5; 9/10] END bootstrap=True, max_depth=2, max_features=sqrt, n_estimators=20;, score=1.000 total time
=   0.0s
[CV 1/5; 10/10] START bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50
[CV 1/5; 10/10] END bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50;, score=0.933 total tim
e=   0.0s
[CV 2/5; 10/10] START bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50
[CV 2/5; 10/10] END bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50;, score=0.867 total tim
e=   0.0s
[CV 3/5; 10/10] START bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50
[CV 3/5; 10/10] END bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50;, score=1.000 total tim
e=   0.0s
[CV 4/5; 10/10] START bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50
```

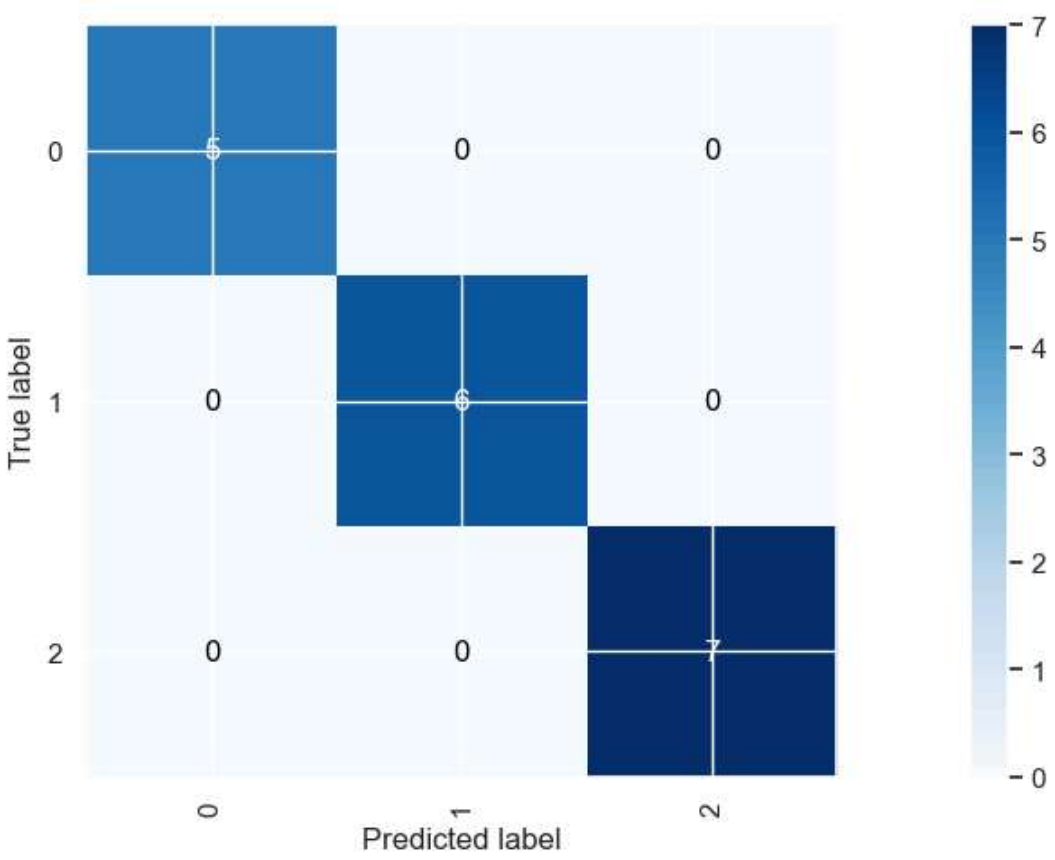
```
[CV 4/5; 10/10] END bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50;, score=0.857 total tim
e=   0.0s
[CV 5/5; 10/10] START bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50
[CV 5/5; 10/10] END bootstrap=True, max_depth=4, max_features=sqrt, n_estimators=50;, score=1.000 total tim
e=   0.0s
Accuracy Of Logistic Regression :  1.0
```



here we use the XGB model

In [28]:
```
xgb_model = xgboost.XGBClassifier()
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)

xgb_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Of Logistic Regression : ", xgb_accuracy)
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
Accuracy Of Logistic Regression :  1.0
```



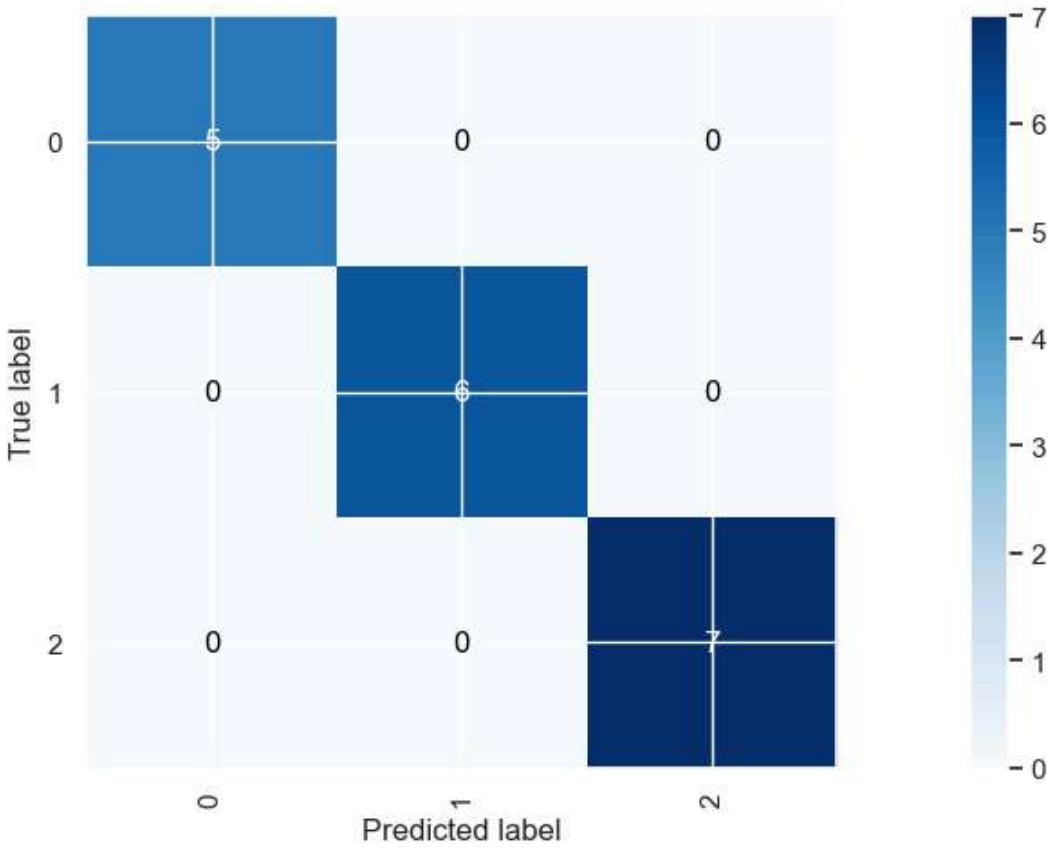
In [29]:
```
model=Sequential()
model.add(Dense(units=64,kernel_initializer='uniform',activation='relu',input_dim=X_train.shape[1]))
model.add(Dense(units=128,kernel_initializer='uniform',activation='relu'))
model.add(Dense(units=64,kernel_initializer='uniform',activation='relu'))
model.add(Dense(units=10,kernel_initializer='uniform',activation='softmax'))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.fit(X_train,y_train,batch_size=256,epochs=10,validation_data=(X_test,y_test))
```

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.1806 - loss: 2.3012 - val_accuracy: 1.0000 - val_loss: 2.
2938
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.9444 - loss: 2.2934 - val_accuracy: 1.0000 - val_loss:
2.2856
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.9583 - loss: 2.2851 - val_accuracy: 1.0000 - val_loss:
2.2761
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.9444 - loss: 2.2753 - val_accuracy: 1.0000 - val_loss:
2.2646
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.9444 - loss: 2.2637 - val_accuracy: 1.0000 - val_loss:
2.2508
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.9444 - loss: 2.2497 - val_accuracy: 1.0000 - val_loss:
2.2341
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.9444 - loss: 2.2328 - val_accuracy: 1.0000 - val_loss:
2.2140
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.9444 - loss: 2.2124 - val_accuracy: 1.0000 - val_loss:
2.1902
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.9444 - loss: 2.1882 - val_accuracy: 1.0000 - val_loss:
2.1619
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.9444 - loss: 2.1595 - val_accuracy: 1.0000 - val_loss:
2.1286
```

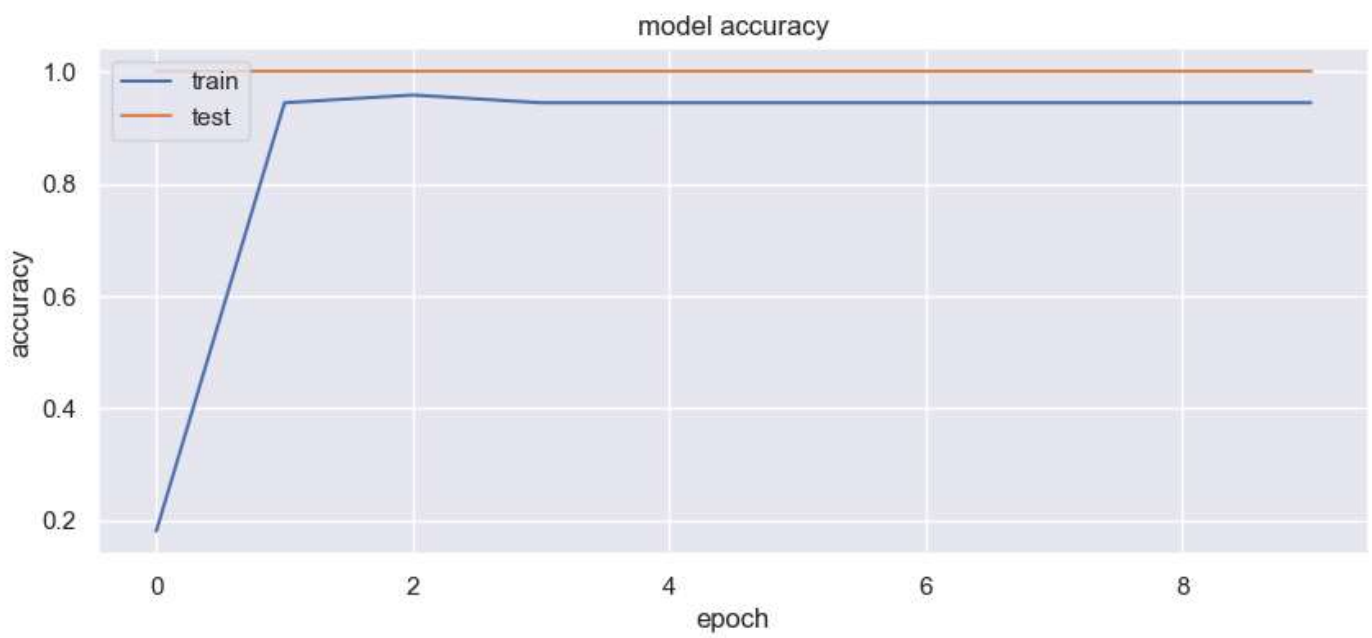
In [30]:
```
print(X_test.shape)
pred = model.predict(X_test)
y_pred = pred.argmax(axis=1)
print(y_pred)
print(y_test)
```

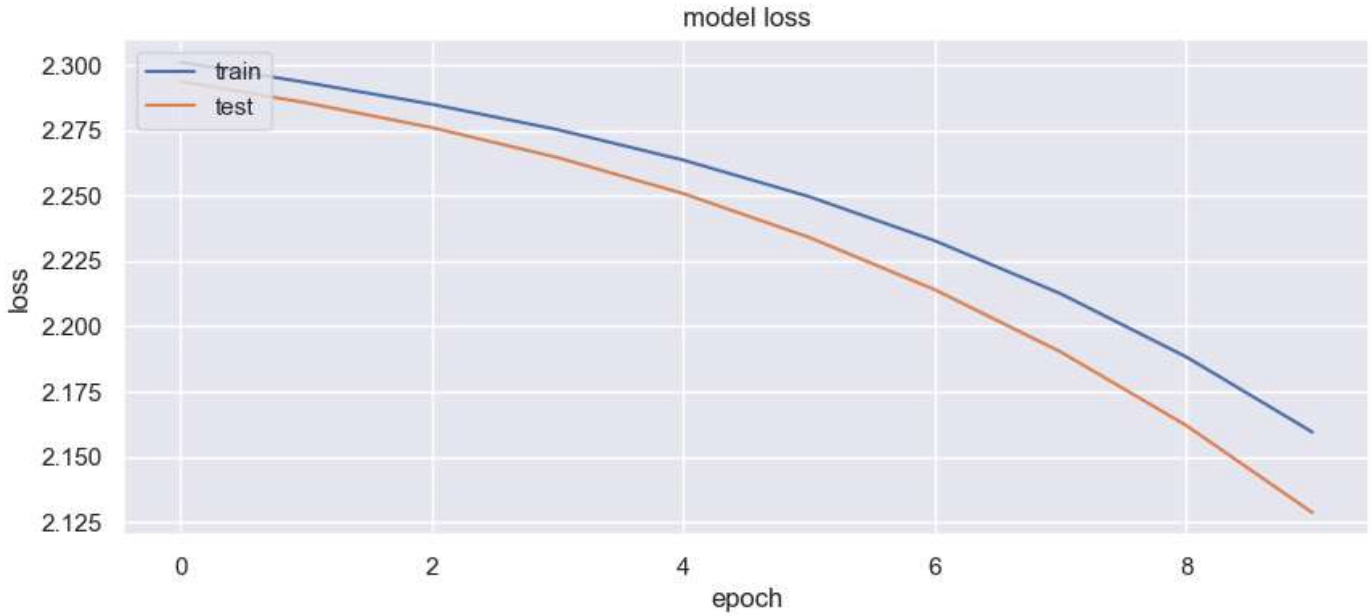
```
dlacc = accuracy_score(y_pred, y_test)
print(dlacc)
cm = confusion_matrix(y_test, y_pred)
print(cm)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
(18, 6)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
[1 2 2 1 0 2 0 0 1 1 2 0 2 2 1 0 1 2]
[1 2 2 1 0 2 0 0 1 1 2 0 2 2 1 0 1 2]
1.0
[[5 0 0]
 [0 6 0]
 [0 0 7]]
```



In [31]:
```
rcParams['figure.figsize'] = 10, 4
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





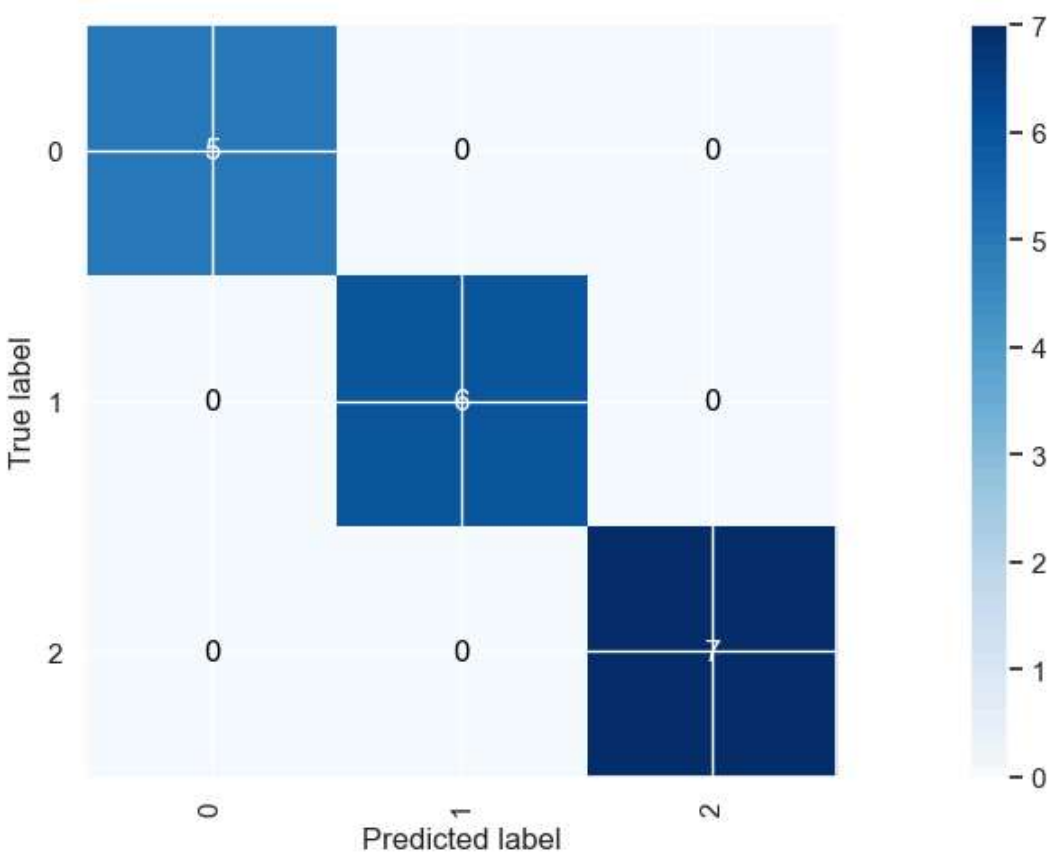
6. Fine-tune your models and combine them into a great solution.

can you get better performance within a model? e.g if you use a KNN classifier how does it behave if you change K (k=3 vs k=5 vs k=?). Which parameters are here to tune in the chosen models?

In [32]:
```
estimators = [
        ('RFC' ,RandomForestClassifier(n_estimators=500, random_state = 42)),
        ('KNC', KNeighborsClassifier(5)),
        ('DTC', DecisionTreeClassifier()),
        ('SVC', SVC(kernel="rbf")),
        ('XGB', xgboost.XGBClassifier()),
        ('RC',  RidgeClassifier())
]

clf = StackingClassifier(
    estimators=estimators,
    final_estimator=GradientBoostingClassifier()
)
def f_score(X_train, X_test, y_train, y_test):
    for clf in classifiers:
        s = time.time()
        clf.fit(X_train,y_train)
        y_pred = clf.predict(X_test)
        f = f1_score(y_true=y_test,y_pred=y_pred,average="macro")
        e = time.time()
```

```
        print(f"Score: {round(f,3)} \t Time(in secs): {round(e-s,3)} \t Classifier: {clf.__class__.__name_
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
f1_score(y_true=y_test,y_pred=y_pred,average="macro")
stackuracy = accuracy_score(y_pred, y_test)
print(stackuracy)
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,np.unique(y_pred))
```

```
1.0
```



In [33]:
```
model_name = ['LR', 'SVC', 'DT', 'RF', 'XGB', 'Stacking', 'DeepLearning']
model_acc1 = [ lr_accuracy, svc_accuracy, dt_accuracy, rf_accuracy, xgb_accuracy, stackuracy, dlacc]
model_acc = [np.round(i, decimals=2) for i in model_acc1]

graph = go.Bar(
    x=model_name,
    y=model_acc,
    opacity=1,
    showlegend=False,
    marker=dict(color=px.colors.qualitative.Pastel1)

)

layout = go.Layout(
    height=450, width=700,
    title = 'Model Accuracy Chart',
    xaxis = dict(title = 'Model', showgrid=False,tickangle=0),
    yaxis = dict(title = 'Accuracy', showgrid=False),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    title_font=dict(size=25, color='#a5a7ab', family='roboto'),
    margin=dict(t=80, b=30, l=70, r=40),
    font=dict(color='#8a8d93'))
fig = go.Figure(data=[graph], layout = layout)
fig.update_traces(textfont=dict(color='#fff'), marker=dict(line=dict(color='#ffffff', width=2)))
iplot(fig)
```

The model accuracy in most of our models is 1. This indicates overfitting, but with the limited amount of data and the nice clusters that formed in our 2D plot, these accuracy numbers may be possible. They perform very well on our test set and other additional tests we threw at it.

With more data, these unknowns could be cleared up, but considering the scope and time available for this project, this was not possible.

7. Present your solution.

Explain why you would choose for a specific model

we chose for deeplearning because we found it easier to implement in the deployment, there wasnt really any other reason because the accuracy of most our models was very high.(see reasons above)

8. Launch, monitor, and maintain your system.

Can you Deployment the model?

> NOTE: The app provides the option for remote access, so you are able to get live sensordata from the phone

In [35]:
```python
import os
import pandas as pd

# Define the path to the folder containing the 3 CSV files
folder_path = "datasets/deployset/2-drinking11 2024-12-18 16-23-57"

# Check if the folder_path exists
if not os.path.exists(folder_path):
    print(f"The folder {folder_path} does not exist.")
else:
    # Create an empty list to hold the summarized data
    summary_data = []

    # Extract person and action from the folder name
    parts = os.path.basename(folder_path).split("-")
    if len(parts) >= 2:
        person = parts[0]
        action_with_index = parts[1].rsplit(" ", 1)[0]
        action = ''.join([i for i in action_with_index if not i.isdigit()])

        summary_row = {
            'subject': person,
            'Activity': action,
        }

        # Iterate through all CSV files in the folder
        for file_name in os.listdir(folder_path):
            if file_name.endswith(".csv"):
                file_path = os.path.join(folder_path, file_name)
                try:
                    # Read the CSV file
                    data = pd.read_csv(file_path)

                    # Skip empty files
                    if data.empty:
                        print(f"Skipping empty file: {file_path}")
                        continue

                    # Rename columns for device compatibility
                    if {"X (m/s^2)", "Y (m/s^2)", "Z (m/s^2)"}.issubset(data.columns) and file_name == "Ac
                        data.rename(columns={
                            "X (m/s^2)": "Acceleration x (m/s^2)",
                            "Y (m/s^2)": "Acceleration y (m/s^2)",
                            "Z (m/s^2)": "Acceleration z (m/s^2)"
                        }, inplace=True)
                    elif {"X (rad/s)", "Y (rad/s)", "Z (rad/s)"}.issubset(data.columns):
                        data.rename(columns={
```

```
                        "X (rad/s)": 'Gyroscope x (rad/s)',
                        "Y (rad/s)": 'Gyroscope y (rad/s)',
                        "Z (rad/s)": 'Gyroscope z (rad/s)'
                    }, inplace=True)
                elif {"X (m/s^2)", "Y (m/s^2)", "Z (m/s^2)"}.issubset(data.columns):
                    data.rename(columns={
                        "X (m/s^2)": 'Linear Acceleration x (m/s^2)',
                        "Y (m/s^2)": 'Linear Acceleration y (m/s^2)',
                        "Z (m/s^2)": 'Linear Acceleration z (m/s^2)'
                    }, inplace=True)

                # Determine which type of data (Accelerometer, Gyroscope, or Linear Acceleration)
                if {'Acceleration x (m/s^2)', 'Acceleration y (m/s^2)', 'Acceleration z (m/s^2)'}.issu
                    columns = ['Acceleration x (m/s^2)', 'Acceleration y (m/s^2)', 'Acceleration z (m/
                elif {'Gyroscope x (rad/s)', 'Gyroscope y (rad/s)', 'Gyroscope z (rad/s)'}.issubset(da
                    columns = ['Gyroscope x (rad/s)', 'Gyroscope y (rad/s)', 'Gyroscope z (rad/s)']
                elif {'Linear Acceleration x (m/s^2)', 'Linear Acceleration y (m/s^2)', 'Linear Accele
                    columns = ['Linear Acceleration x (m/s^2)', 'Linear Acceleration y (m/s^2)', 'Line
                else:
                    print(f"File {file_path} does not contain recognized column names. Skipping.")
                    continue

                # Calculate statistics for relevant columns
                mean_values = data[columns].mean()
                std_values = data[columns].std()
                min_values = data[columns].min()
                max_values = data[columns].max()

                # Add statistics to the summary row
                for col in columns:
                    summary_row[f'{col}_mean'] = mean_values[col]
                    summary_row[f'{col}_std'] = std_values[col]
                    summary_row[f'{col}_min'] = min_values[col]
                    summary_row[f'{col}_max'] = max_values[col]

            except Exception as e:
                print(f"Error reading file {file_path}: {e}")

        # Append the summary row to the summary data
        summary_data.append(summary_row)

    # Create a dataframe from the summary data
    if summary_data:
        summary_df = pd.DataFrame(summary_data)
    else:
        print("No valid data found.")

X_dep, y_dep = summary_df.drop(['subject', 'Activity'], axis=1), summary_df['Activity']
y_dep=encoder.fit_transform(y_dep)
X_dep = pca.transform(X_dep)
```

In [36]:
```
pred = model.predict(X_dep)
y_pred = pred.argmax(axis=1)
print(y_pred)
print(y_dep)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
[0]
[0]
```

9. Additional Questions

- Explain the chosen motions you chose to be classified.

  Casual walking and walking with a phone in you hands are cosen, since they are in travel similar but also different. Drinking is chosen because we thought we thought it would be a unique and distinguishable movement.

- Which of these motions is easier/harder to classify and why?

Drinking was by far the hardest motion to train on. This is because of the orientation you have to keep your phone in to simulate drinking. Also the duration and intensity in the motion varies the most there.

- After your experience, which extra sensor data might help getting a better classifier and why?

  Since we allready used the three main sensors to measure the activities, vision would be the next addition to the dataset. This is because the drinking can be distinguished more easily.

- Explain why you think that your chosen algorithm outperforms the rest?

  We do not a clear vision on which might perform the best, due to a lacking anount of data. Because of this there is a chance that some are overfitting on the data. We could not find a way to distinguish this. To deploy a model the deeplearning was chosen, since it was the least difficult to implement on a single data point.

- While recording the same motions with the same sensor data, what do you think will help improving the performance of your models?

  A lack of data is certainly on of the most obious ones. Though generalising the raw data into more features that are usefull for the training can also help. This can be a fourier transformation to gain an understanding of the frequency sprectrum of the motion.

```
# Place your comments / conclusions / insight here
```