BIG DATA IN BUSINESS AND INDUSTRY, PRACTICAL WORK\ Taru Haimi, 0565878\ Joona Ylijoki, 0522866

TASK: Create ML model to forecast user's next exercise type, time and, possibly, duration

In [ ]:
```
########################################
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import pa_functions as pafuns
```

Loading data from json files into one dataframe.\ Source code: READ_workoutsToMoodle.py

In [ ]:
```
with open(r'folderpath.txt','r') as rfile:
    folderpath = rfile.read()
data = pd.DataFrame()
for file in os.listdir(folderpath):
    newdf = pafuns.read_file_to_df(os.path.join(folderpath, file))
    data = pd.concat([data, newdf], ignore_index=True)
data.reset_index(drop=True, inplace = True)
```

EXPLORATORY DATA ANALYSIS & DATA PREPROCESSING

In [ ]:
```
print(f'Columns (variables) in the data:', data.columns)
print(f'Amount of variables in the data:', len(data.columns))
print(f'Amount of observations in the data:', len(data))
print(f'Data is from time period from ', data['start_time'][0], ' to ', data['end_t
```

```
Columns (variables) in the data: Index(['sport', 'source', 'created_date', 'start_
time', 'end_time',
       'duration_s', 'distance_km', 'calories_kcal', 'altitude_min_m',
       'altitude_max_m', 'speed_avg_kmh', 'speed_max_kmh', 'ascend_m',
       'descend_m', 'start_lat', 'start_long', 'end_lat', 'end_long',
       'hydration_l'],
      dtype='object')
Amount of variables in the data: 19
Amount of observations in the data: 3456
Data is from time period from  2017-01-01 08:53:04.0  to  2020-12-30 19:07:14.0
```

The dataset includes 19 different variables and 3456 observations from time period 1.1.2017 - 30.12.2020.

Finding out all different types of sports

In [ ]:
```
sporttypes = data["sport"].unique()
print(sporttypes)
print("Number of different sport types: ", len(data["sport"].unique()))
```

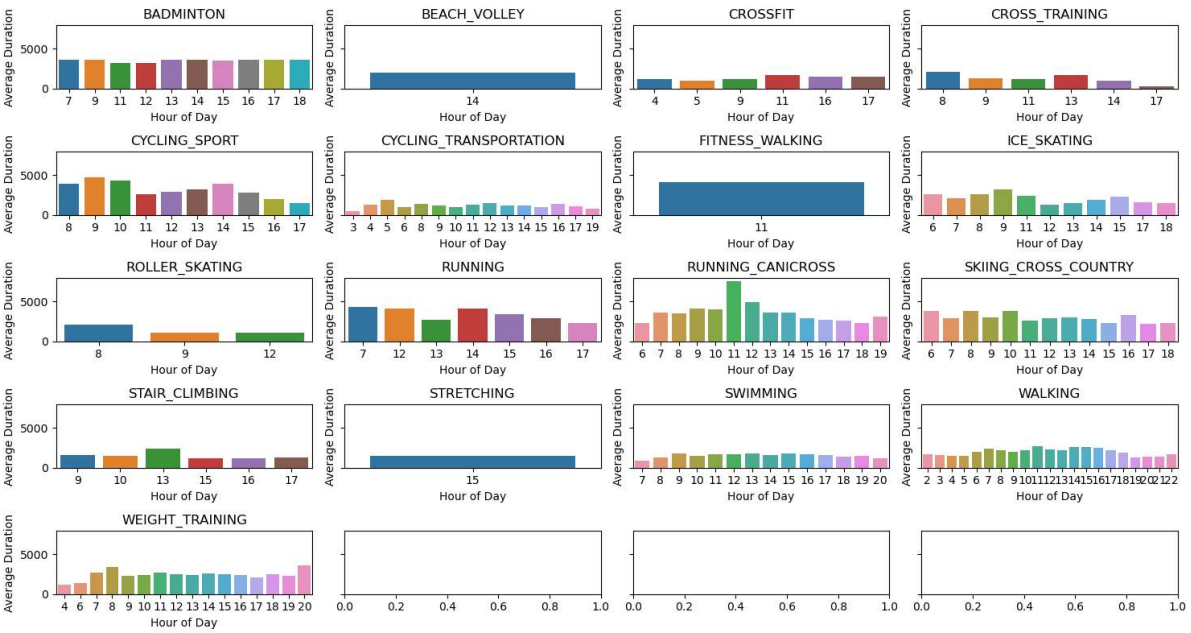
```
['WALKING' 'WEIGHT_TRAINING' 'RUNNING' 'SKIING_CROSS_COUNTRY' 'SWIMMING'
 'BADMINTON' 'ROLLER_SKATING' 'FITNESS_WALKING' 'CYCLING_SPORT' 'CROSSFIT'
 'RUNNING_CANICROSS' 'ICE_SKATING' 'BEACH_VOLLEY' 'CYCLING_TRANSPORTATION'
 'STAIR_CLIMBING' 'CROSS_TRAINING' 'STRETCHING']
Number of different sport types:  17
```

Exploring starting hours vs sports and their average duration.

These barplots describe relations between starting times and sport types, and in addition they show how the durations of activities vary between clock times and sports. It can be observed that only walking activities are started to record round the clock, 2am-10pm. Then most notable is that beach volley, fitness walking and stretching have records only in single specific hours. Roller skating has also very specific starting times, between 8am-12pm. Then many sports have pretty similar timeline starting from early morning and ending in late evening. The exact starting and ending times vary between 4-9am and 5-8pm.

Based on these notes, the starting time might be a good variable to estimate next exercise type, but certainly not alone. Some sports can occur only certain hours of the day which gives good limitations for model, but most of sports can occur same time of the day, so more features are definitely needed to estimate these hours.

In [ ]:
```
# Convert 'timestamp' column to datetime
data['start_time'] = pd.to_datetime(data['start_time'])

# Extract the hour from the timestamp
data['start_hour'] = data['start_time'].dt.hour

# Calculate the average duration for each hour and sport type
mean_durations = data.groupby(['sport', 'start_hour'])['duration_s'].mean().reset_i

# Create a figure with subplots
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(15, 8), sharey=True)

# Flatten the 2D array of subplots
axes = axes.flatten()

# Iterate over sport types
for i, (sport, hourlydata) in enumerate(mean_durations.groupby('sport')):
    sns.barplot(x='start_hour', y='duration_s', data=hourlydata, ax=axes[i])
    axes[i].set_title(sport)
    axes[i].set_xlabel('Hour of Day')
    axes[i].set_ylabel('Average Duration')

# Adjust layout
plt.tight_layout()
plt.show()
```

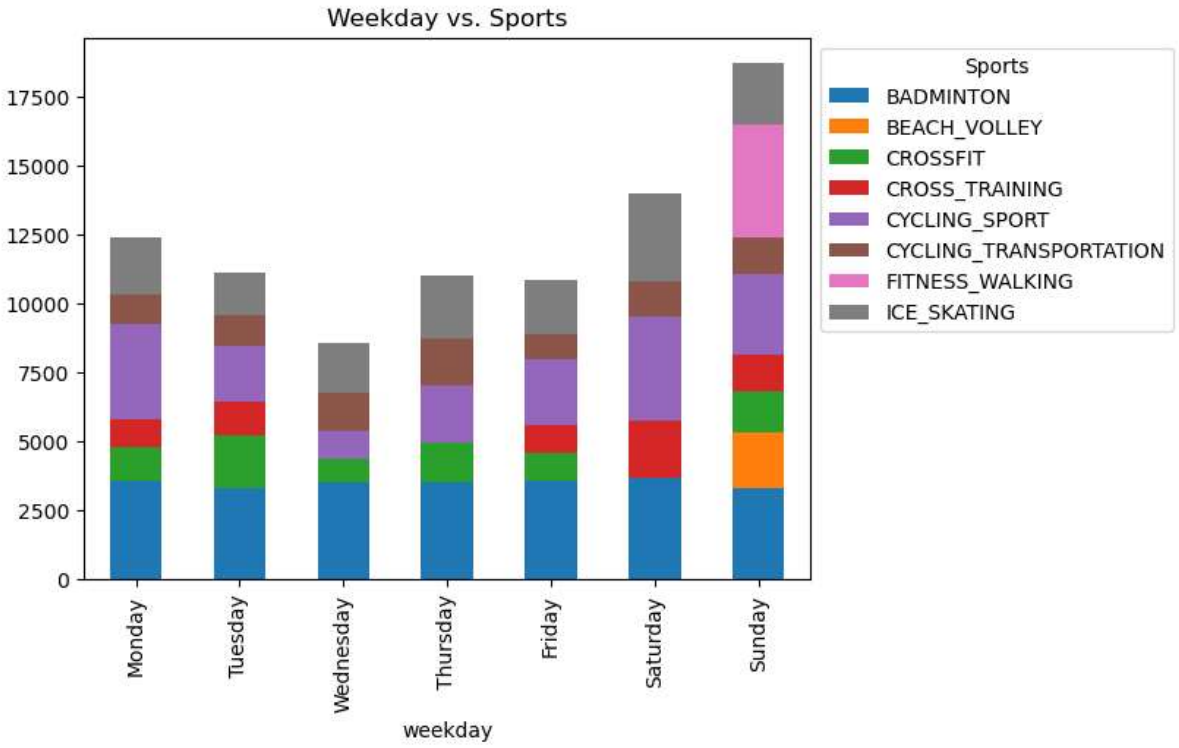


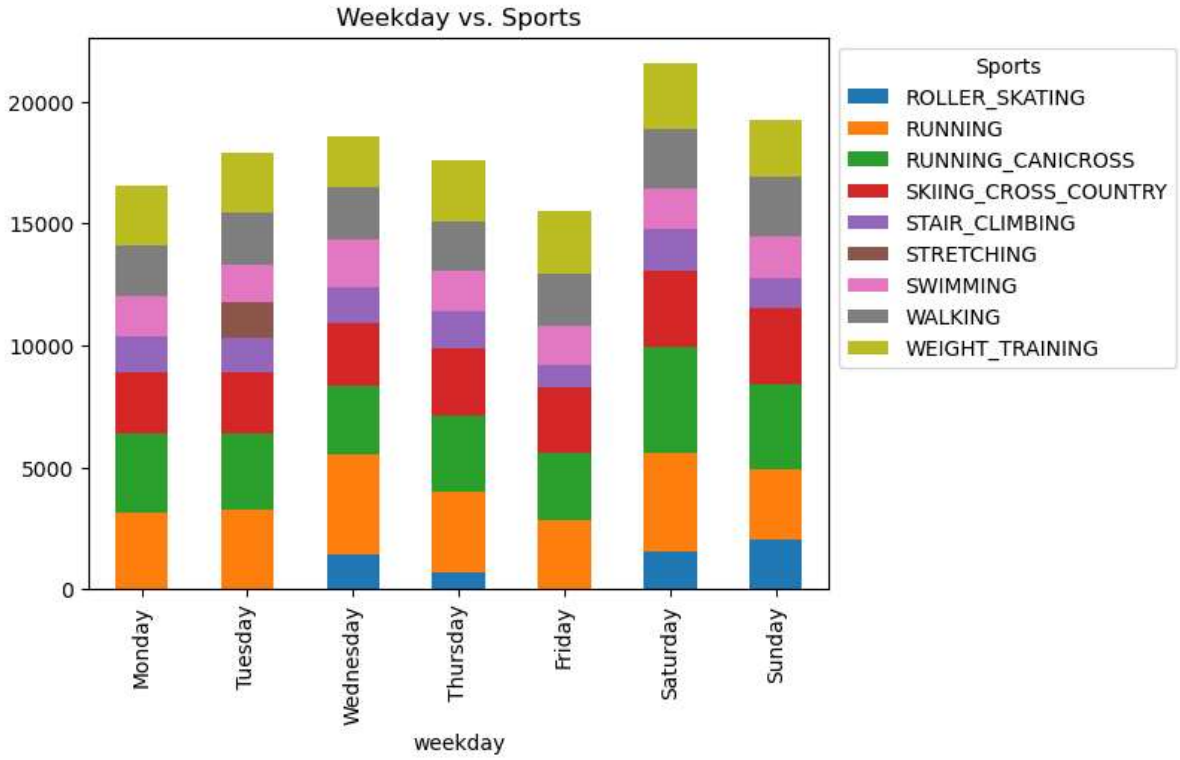
Exploring weekdays vs sports and their average duration.

These barplots tell on which weekdays certain sports have been recorded, and again also shows how the average durations are distributed between weekdays and sports. Most sports have records from every day or almost every day, but notable are those sports that have records only certain days. This kind of sports are stretching and beach volley. Additionally cross_training and roller_skating have limited weekdays to be recorded. Most of durations between weekdays in same sport type don't vary a lot, but durations between sport types in same weekday are very different from each other.

The weekday might be a good variable to be used in model, since with it some possible sports can be cut off from options. Also if the estimations showed that the duration of sport would be short, the sports with long average duration can be cut off.

In [ ]:
```
# Extract the weekday from the timestamp
data['weekday'] = data['start_time'].dt.weekday

# Calculate the average duration for each weekday and sport type
mean_durations = data.groupby(['sport', 'weekday'])['duration_s'].mean().reset_inde

# Pivot the data to have weekdays as columns
mean_durations_pivot = mean_durations.pivot(index='weekday', columns='sport', value

# Plot the average duration per sport and  per weekday in two parts:
datapart1 =  mean_durations_pivot.iloc[:,list(range(0, 9))]
datapart2 = mean_durations_pivot.iloc[:, [0] + list(range(9, 18))]
pafuns.plot_weekday_data(datapart1)
pafuns.plot_weekday_data(datapart2)
```



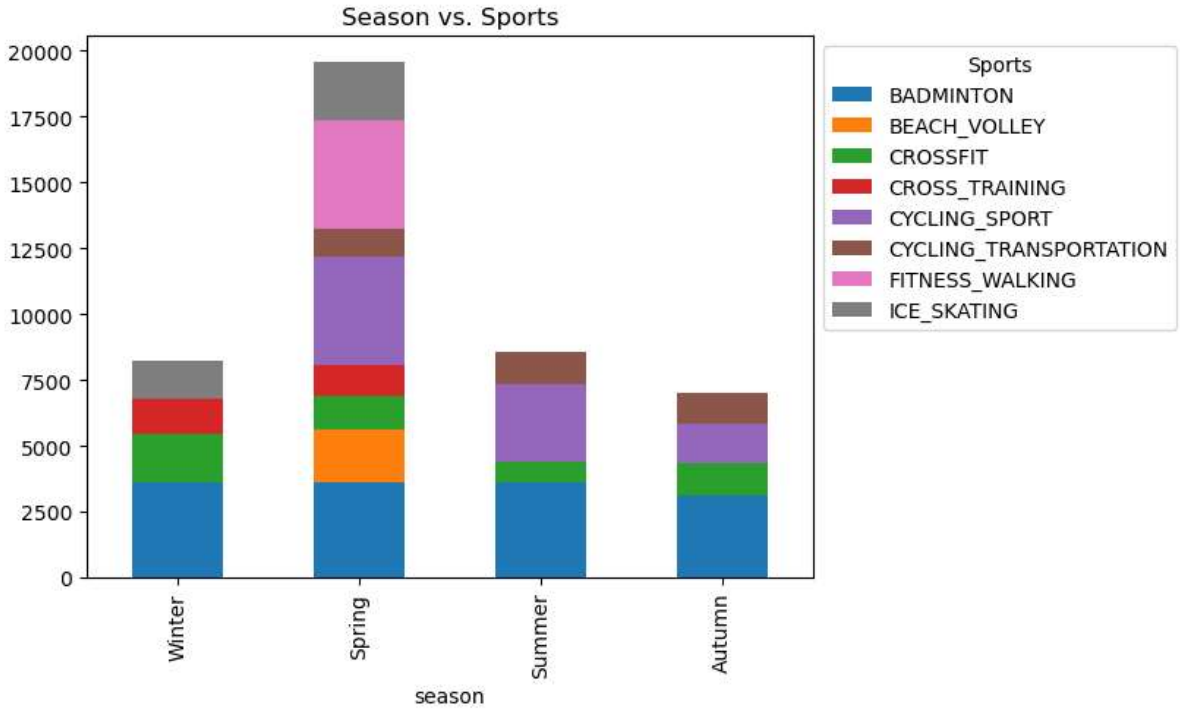


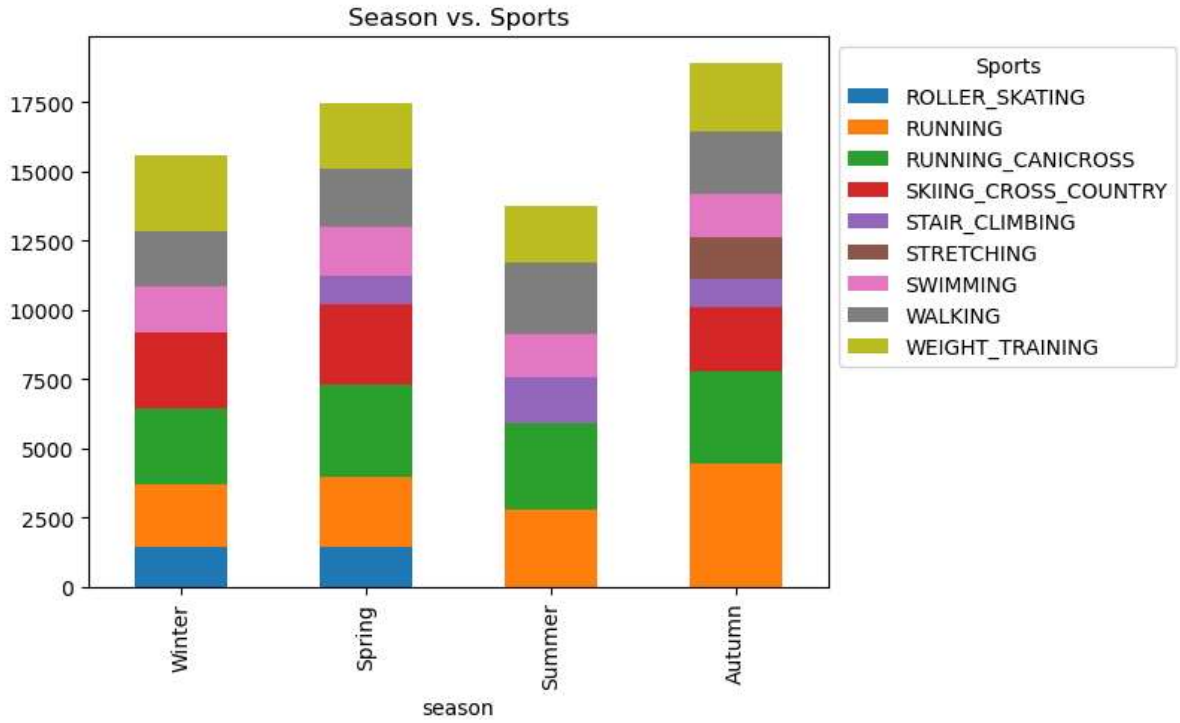
Exploring seasons vs sports and their average duration.

When comparing the barplots between sports and seasons, it can be seen that some sports like walking is done throughout the year, not depending on current season. Additionally, some sports are dependent on the season, like ice skating, skiing_cross_country and cycling_sport. However, there are not clear differences between sport durations in different seasons. These observations indicate that season measurement (or month) could be one of features to use for sport type prediction, but it might not be that good for predicting the sport duration.

In [ ]:
```
data['month'] = data['start_time'].dt.month
# Define a mapping from month to season
month_to_season = {
    12: '1',
    1: '1',
    2: '1',
    3: '2',
    4: '2',
    5: '2',
    6: '3',
    7: '3',
    8: '3',
    9: '4',
    10: '4',
    11: '4'
}
# Create a new column for seasons
data['season'] = data['month'].map(month_to_season)

# Calculate the average duration for each weekday and sport type
mean_durations = data.groupby(['sport', 'season'])['duration_s'].mean().reset_index

# Pivot the data to have weekdays as columns
mean_durations_pivot = mean_durations.pivot(index='season', columns='sport', values

# Plot the average duration per sport and  per weekday in two parts:
```

```
datapart1 =  mean_durations_pivot.iloc[:,list(range(0, 9))]
datapart2 = mean_durations_pivot.iloc[:, [0] + list(range(9, 18))]
pafuns.plot_season_data(datapart1)
pafuns.plot_season_data(datapart2)
```





Avg. Calories Burned (Kcal) w.r.t Sport

As we can see from the pie chart below, the average burned calories w.r.t. to each sport type is the smallest for stretching and highest for cross country skiing. This might give us and the model some information about the resting times after heavier exercises and/or the next exercise type. Therefore we decided to include the burned calories variable in the model.

In [ ]:
```
# Calculate the average burned calories for each sport type
mean_calories = data.groupby(['sport'])['calories_kcal'].mean().reset_index()
# Round the values to one decimal and store in a series
avg_calories = mean_calories['calories_kcal'].round(1)
```

```
# Store sport types in a series
sports = mean_calories['sport']

# Plot pie chart from average calories burned w.r.t each sport type
fig = px.pie(data_frame=mean_calories, names='sport', values=avg_calories,
             title='Average Calories Burned by Sport Type')
# Add both 'sports' and 'avg_calories' with 'kcal' in the label
fig.update_traces(textposition='outside', texttemplate='%{label}: %{value: .1f} kca
                  marker=dict(line=dict(color='#000000', width=1.5)))
# Adjust the legend position to the right
fig.update_layout(legend=dict(x=1.3, y=0.5), height=550, width=1150)
# Show the interactive pie chart
fig.show()
```

Amount of observations per sport type

As we can see from the pie chart below, the amount of observations for 'walking' is so much bigger than with observations count of any other sport type. This might affect our model because one category is so dominating. It is also notable that beach volley, fitness walking, and stretching occur only once in the data. If the person who has recorded these, has done these sports only once during the four year period, these sports can probably be excluded from the model, because their occurance is so rare and hard to predict.

In [ ]:
```
sport_counts = data['sport'].value_counts().reset_index()
sport_counts.columns = ['sport','count']

# Plot pie chart from the count of observations for each sport type
fig = px.pie(data_frame=sport_counts, names='sport', values='count',
             title='Count of Observations for each Sport Type')
# Add both 'sports' and 'avg_calories' with 'kcal' in the label
fig.update_traces(textposition='outside', texttemplate='%{label}: %{value: .1f} obs
                  marker=dict(line=dict(color='#000000', width=1.5)))
# Adjust the legend position to the right
fig.update_layout(legend=dict(x=1.3, y=0.5), height=700, width=1150)
# Show the interactive pie chart
fig.show()
```

Exploring locations of sports

There're some observations of sports that are recorded outside of Finland. Their total count is however very small when compared to the whole data amount, and they all are sport type "walking". These are probably from holiday or work trips, and the person was just "walking around in a new city". These sport activities are quite probably very anomalous from normal life, so they can be excluded from the model because they are possible outliers and could give model wrong idea.

In [ ]:
```
# Bounding box for Finland's coordinate
finland_bounding_box = {
    'min_latitude': 60,
    'max_latitude': 70,
    'min_longitude': 20,
    'max_longitude': 30,
}

# Check if each location is outside Finland
outside_finland = (
```

```
    (data['start_lat'] < finland_bounding_box['min_latitude']) |
    (data['start_lat'] > finland_bounding_box['max_latitude']) |
    (data['start_long'] < finland_bounding_box['min_longitude']) |
    (data['start_long'] > finland_bounding_box['max_longitude'])
)

# Locations outside Finland and their count
df_outside_finland = data[outside_finland]
print(f'Number of locations outside Finland: ' + str(len(df_outside_finland)))
```

```
Number of locations outside Finland: 30
```

Removing insignificant variables and observations

Sporttypes with only one observation is considered to be outliers in the data and they are therefore removed from the data. The observations outside Finland are also excluded. The variables that were thought to be important in this modeling case were achieved by exploring the barplots and pie charts in above sections. The other variables are considered more insignificant and therefore they are removed from the dataset that is used to build the model. Next, the observations that contain nan values are removed from the remaining data. Lastly, if the observation's duration is very short (5 min or less), they are also excluded from the modeling data because they might be recorded by mistake.

In [ ]:
```
# Find sports with only one occurance
single_occurrence_classes = data['sport'].value_counts()[data['sport'].value_counts

# Remove rows where 'sport' is in single_occurance_classes
data = data[~data['sport'].isin(single_occurrence_classes)]
print(f'Sports with only one occurance:', single_occurrence_classes)
# Remove the corresponding sports from labelnames
newsports = sports[sports != "STRETCHING"]
newsports = newsports[newsports != "BEACH_VOLLEY"]
newsports = newsports[newsports != "FITNESS_WALKING"]
newsports = newsports.reset_index(drop=True)

# Remove the observations outside Finland
data = data[~outside_finland]

# Selecting only the significant variables to be kept in the data
data = data[['sport','start_hour','weekday','season','duration_s','calories_kcal','
# Remove the observations with nan values
data = data.dropna().reset_index(drop=True)
# Including only the observations that have a reasonable duration
data = data[data['duration_s']>= 300].reset_index(drop=True)
```

```
Sports with only one occurance: Index(['FITNESS_WALKING', 'BEACH_VOLLEY', 'STRETCH
ING'], dtype='object')
C:\Users\joona\AppData\Local\Temp\ipykernel_2324\3340380374.py:14: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

Creating categorical dataframe for the input variables: changing the data to a format that the different categories that we will input into the model are numerical values. Doing this for the following variables: sport, duration, starting hour, kcal, weekday, month. Also creating shifted categories that represent the last exercise's type, time, kcal consumption, weekday, and month.

In [ ]:
```
categorical_df = pd.DataFrame()

# Adding categorical variables to the DataFrame
categories = ['sport_category','duration_category_text','duration_sec_category','st
              'weekday_category','month_category', 'last_exercise_category', 'last_
              'last_exercise_weekday_category', 'last_exercise_month_category', 'st

# Create a mapping dictionary for each needed variable to categories
sport_type_to_category = {sport_type: category for category, sport_type in enumerat
# Map sport types to categories and add them to the categorical DataFrame
categorical_df[categories[0]] = pd.Categorical(data['sport'].map(sport_type_to_cate
data['sport_type_category'] = categorical_df[categories[0]]

# Categories for duration as intervals
intervals = [0,11,31,61,121, float('inf')]
interval_labels = ['Max 10 mins', 'Max half an hour', 'Max one hour', 'Max two hour
data['DurationsInMinutes'] = data['duration_s'] / 60
# Corresponding text categories
categorical_df[categories[1]] = pd.cut(data['DurationsInMinutes'], bins=intervals,
# Numerical categories for duration [0, 1, 2, 3, 4]
duration_type_to_category = {duration_type: category for category, duration_type ir
categorical_df[categories[2]] = pd.Categorical(categorical_df[categories[1]].map(du

# Create a mapping dictionary for start hour variable to categories
categorical_df[categories[3]] = data['start_hour']
# Making a text category with intervals for starting hour
intervals_hours = [0, 6, 11, 14, 17, 24]
intervals_hours_labels = ['Night', 'Morning', 'Noon', 'Afternoon', 'Evening']
categorical_df[categories[13]] = pd.cut(data['start_hour'], bins=intervals_hours, l

# Categories for burned kcal as intervals
interval_kcal = [0,101,201,301,401,501,601, float('Inf')]
interval_kcal_labels = ['Max 100 kcal', 'Max 200 kcal', 'Max 300 kcal', 'Max 400 kc
# Corresponding text categories
categorical_df[categories[4]] = pd.cut(data['calories_kcal'], bins=interval_kcal, l
# Numerical categories for burned calories [0, 1, 2, 3, 4, 5, 6]
kcal_type_to_category = {kcal_type: category for category, kcal_type in enumerate(i
categorical_df[categories[5]] = pd.Categorical(categorical_df[categories[4]].map(kc

# Weekday category
categorical_df[categories[6]] = data['weekday']

# Month category
categorical_df[categories[7]] = data['month']

# Adding the last activity/exercise type to each observation
categorical_df[categories[8]] = categorical_df['sport_category'].shift(fill_value=N
# Adding the last activity/exercise timing to each observation
categorical_df[categories[9]] = categorical_df['start_hour_category'].shift(fill_va
# Adding the last activity/exercise burned kcal to each observation
categorical_df[categories[10]] = categorical_df['kcal_category'].shift(fill_value=N
# Adding the last activity/exercise weekday to each observation
categorical_df[categories[11]] = categorical_df['weekday_category'].shift(fill_valu
# Adding the last activity/exercise month to each observation
categorical_df[categories[12]] = categorical_df['month_category'].shift(fill_value=
```

Cleaning categories: changing all the categorical_df dataframe columns to type int64, so that they all have integer number types. Extracting the text formatted columns from the DataFrame and creating the numerical_df DataFrame that is used in the model.

In [ ]:
```
# Extracting the text categories from the categorical DataFrame
columns_to_remove = ['duration_category_text','kcal_category_text']
```

```
# Check if columns to remove exist in the DataFrame and remove them
for column in columns_to_remove:
    if column in categorical_df.columns:
        categorical_df.drop(column, axis=1, inplace=True)

# Convert all columns to type int64
numerical_df = pd.DataFrame()
for columns in categorical_df.columns:
    numerical_df[columns] = categorical_df[columns].astype('category').cat.codes.as
```

```
sport_category
duration_sec_category
start_hour_category
start_hour_text_category
kcal_category
weekday_category
month_category
last_exercise_category
last_exercise_timing
last_exercise_kcal_category
last_exercise_weekday_category
last_exercise_month_category
```

TESTING THE ML MODEL

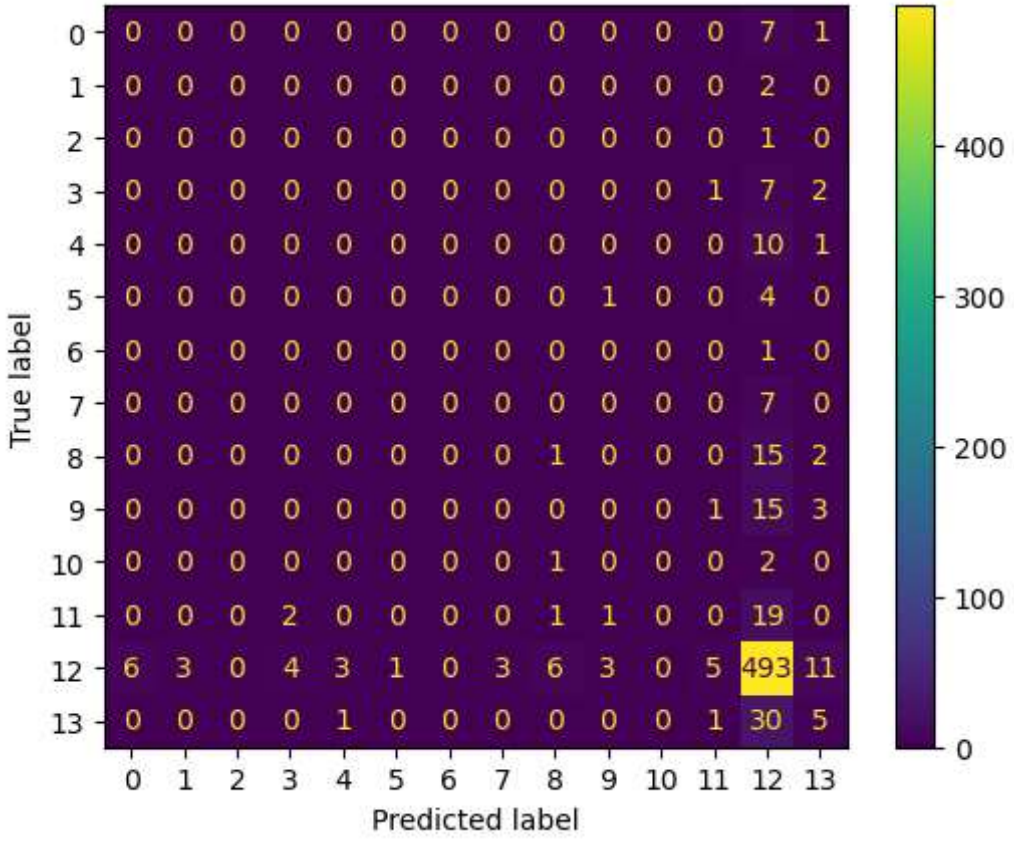
Forming the final data frame that is used to train the model. The independent variables (in X) are the type, time, burned kcal, weekday, and month of the last known exercise. The dependent variables y_type, y_time, and y_duration are the variables that we are predicting. The data is split into training and testing with training size 80% and testing size 20%, all the sport types in the data is assured to be in the training and testing partitions with the parameter 'stratify'. The type of the sport is modeled with RandomForestClassifier, and the starting time and duration of the exercise is predicted using RandomForestRegressor.

After the fitting of the models, predictions are made and the models are evaluated using the test data. The resulted accuracy for the model is 73%, MSE for the time is 15.29, and MSE for the duration is 0.46. Both of the MSEs are somewhat small so the starting time and duration is predicted quite well. Also the accuracy is okay. The confusion matrix shows that the varible 12 ('walking') is so dominating that it is also predicted to numerous other sport types.

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score, confusion_matrix, C

# Features and target variables
X = numerical_df[['last_exercise_category', 'last_exercise_timing', 'last_exercise_
y_type = numerical_df['sport_category']
y_time = numerical_df['start_hour_category']
y_duration = numerical_df['duration_sec_category']

# Split data into training and testing sets
X_train, X_test, y_type_train, y_type_test, y_time_train, y_time_test, y_duration_t
    X, y_type, y_time, y_duration, test_size=0.2, random_state=42, stratify=numeric

# Random Forest Classifier for 'type'
type_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
type_classifier.fit(X_train, y_type_train)

# Random Forest Regressor for 'time'
time_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
time_regressor.fit(X_train, y_time_train)
```

```
# Random Forest Regressor for 'duration'
duration_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
duration_regressor.fit(X_train, y_duration_train)

# Make predictions
type_predictions = type_classifier.predict(X_test)
time_predictions = time_regressor.predict(X_test)
duration_predictions = duration_regressor.predict(X_test)

# Evaluate the models
type_accuracy = accuracy_score(y_type_test, type_predictions)
time_mse = mean_squared_error(y_time_test, time_predictions)
duration_mse = mean_squared_error(y_duration_test, duration_predictions)

print(f'Type Accuracy: {type_accuracy:.2f}')
print(f'Time Mean Squared Error: {time_mse:.2f}')
print(f'Duration Mean Squared Error: {duration_mse:.2f}')

# Plotting the confusion matrix
conf_matrix = confusion_matrix(y_type_test, type_predictions)
cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix)
cm_display.plot()
plt.show()
```

```
Type Accuracy: 0.73
Time Mean Squared Error: 15.29
Duration Mean Squared Error: 0.46
```



Own input testing

The inputs 'own_exercise', 'own_exercise_start_hour', 'own_exercise_burned_kcal_category', 'own_exercise_weekday_category', and 'own_exercise_month_category' form the independent variables for the models. As an example the values are set as sport type: running, start time: 10, burned kcal: "Max 100 kcal" (category 0), weekday: Monday (category

0), month: November (category 10) respectively. The resulted predictions for the next exercise using these independent variables are as follow:

Exercise type: Walking

Exercise time: Morning

Exercise duration: Max one hour

In [ ]:
```
# Selecting the own inputs
print('Possible sport types: \n', newsports)
own_exercise = 'RUNNING'
own_exercise_category = data.set_index('sport')['sport_type_category'].get(own_exer

# The hour the last exercise was started 0-23
own_exercise_start_hour = 10

# Burned calories category: 'Max 100 kcal' = 0, 'Max 200 kcal' = 1, 'Max 300 kcal'
own_exercise_burned_kcal_category = 0

# Own exercise weekday category: 'Monday' = 0, 'Tuesday' = 1, 'Wednesday' = 2, 'Thu
own_exercise_weekday_category = 0

# Own exercise month category: 0-11
own_exercise_month_category = 10

# Forming the own X variable
own_columns = ['last_exercise_category', 'last_exercise_timing', 'last_exercise_kca
X_own = pd.DataFrame(columns=own_columns)
new_observation = {'last_exercise_category': own_exercise_category[0], 'last_exerci
                   'last_exercise_weekday_category': own_exercise_weekday_category,
X_own = pd.concat([X_own, pd.DataFrame([new_observation])], ignore_index=True)

# Make predictions
own_type_predictions = type_classifier.predict(X_own)
own_time_predictions = time_regressor.predict(X_own)
own_duration_predictions = duration_regressor.predict(X_own)

# Answers
next_sport = data.loc[data['sport_type_category'] == own_type_predictions.item(), '
next_sport_time = categorical_df.loc[categorical_df['start_hour_category'] == round
next_sport_duration = interval_labels[round(own_duration_predictions.item())]

print(f'Predicted sport type: {next_sport}')
print(f'Predicted exercise time: {next_sport_time}')
print(f'Predicted exercise duration: {next_sport_duration}')
```

```
Possible sport types:
 0                  BADMINTON
1                   CROSSFIT
2             CROSS_TRAINING
3              CYCLING_SPORT
4     CYCLING_TRANSPORTATION
5                ICE_SKATING
6             ROLLER_SKATING
7                    RUNNING
8           RUNNING_CANICROSS
9        SKIING_CROSS_COUNTRY
10            STAIR_CLIMBING
11                  SWIMMING
12                   WALKING
13           WEIGHT_TRAINING
Name: sport, dtype: object
Predicted sport type: WALKING
Predicted exercise time: Morning
Predicted exercise duration: Max one hour
```

Conclusions: ML models were created to predict the next exercise type, time and duration. The predictions seem to work just fine, but it was noticed that the type of the exercise is most of the time 'walking' possibly due to the obviously dominating amount of observations that the type 'walking' has compared to the other sport types.