TASK 1:

1) Do the CRISP-DM analysis of the project.
2) Pre-process the attribute values so that they are in the appropriate form to be given as input for an algorithm.
3) Visualize the data with different techniques and do some descriptive analysis.

CRISP ANALYSIS

- **BUSINESS UNDERSTANDING:** The goal of this project is to implement a machine learning model able to recognize Sign Language words when performed by different people, in different angles, and translates them, in real-time, into a meaningful text, using Kinect camera data to assist teachers in understanding students. There have already been some systems in the field of SLR with the same or similar aim, nonetheless, most of them failed in recognizing gestures when the signer was in a different position with respect to the camera, or when the signer performed the gesture in a different plane. It is then important to train a model able to overcome such issues and deliver more reliable and accurate results, potentially improving the communication between teachers and students, especially in the deaf schools.
  The data mining goal consists in predicting the sign language gestures based on the coordinates of 20 body joints captured by the Kinect camera.
- **DATA OVERVIEW:** Data were collected by using Kinect camera, capturing coordinates of 20 body joints. The dataset is composed of 30 different signs gestures of Indian Sign Language, performed 9 times by 10 different people, for a total of 2700 gestures. All gestures were performed in different directions, in order to increase the robustness of the model. The dataset is then composed by x, y, z coordinates for each joint, which can be found in numeric format, positive or negative depending on the data position in the 3D space.
- **DATA UNDERSTANDING & PREPARATION:** The data used in this project are sequential data as each sign is represented by multiple frames, following the movement of the body joints. Therefore, there are 60 values (20 joints x 3 coordinates) for each frame. No missing data were found in the dataset and data were visualized using 3D plots to capture data point location in the space, analyzing their sparseness and overall variance. Statistical descriptors like mean, variance, etc. were computed using different approaches, e.g., average for each column (joint), or variance of joints' coordinates for a specific gesture and for each person. This

provides us with more understanding of what kind of data we are dealing with and how we should deal with them in the further steps of the project. Implicit features were also extracted to increase the amount of information, such as the distance from each joint to the center joint (joint number 11). Given the diversity of signers, data were also normalized to remove variations.

- **MODELING:** As above mentioned, data are sequential, and specific models need to be used in order to capture the time relationship between data. Some of the options include Recurrent Neural Networks, of which LSTM is one of the best models to work with time-series. Other models will be explored like Gated Recurrent Unit, a model able to capture data dependencies and that involves less parameters than LSTMs making computationally-lighter models, and Bidirectional Long Short Term Memory, a RNN variant which processes sequences both forward and backward. Each model needs to be evaluated and then selected, taking into consideration different aspects and performance indicators described below.
- **EVALUATION:** Models will be evaluated by calculating the accuracy (percentage of correct predictions of gestures), precision and recall (a more robust metrics that can provide more insights on the model's performance), and F1-score (a balance of precision and recall). A confusion matrix will be used to visualize the true positives and negatives and false positives and negatives, highlighting potential problems with the model.

PREPROCESSING

1. Data Loading

```python
# Folder path containing the text files
folder_path = os.getcwd()+'/combined'  # Replace with your folder path

# List to hold all the data
all_data = []

# Process each text file
for file_name in os.listdir(folder_path):
    if file_name.endswith(".txt"):
        try:
            # Split the file name to extract action, name, and number of times
            file_parts = file_name.split('_')
            action = file_parts[0]
            name = file_parts[1]
            num_times = file_parts[2].replace('.txt', '')  # Remove the .txt extension

            # Read the content of the text file
            file_path = os.path.join(folder_path, file_name)
            with open(file_path, 'r') as file:
                lines = file.readlines()

            # Process each line in the text file
            for line in lines:
                try:
                    # Split the line into 60 columns of float values
                    row_data = list(map(float, line.strip().split()))
                    if len(row_data) != 60:
                        raise ValueError(f"Expected 60 values, got {len(row_data)}")

                    # Add action, name, and number of times columns
                    row_data.extend([action, name, int(num_times)])

                    # Append the row to the list
                    all_data.append(row_data)
                except Exception as e:
                    print(f"Error processing line in file {file_name}: {e}")
                    continue  # Skip to the next line
        except Exception as e:
            print(f"Error processing file {file_name}: {e}")
            continue  # Skip to the next file

# Create a DataFrame with 60 data columns + 3 additional columns for action, name, and num_times
columns = [f'col_{i+1}' for i in range(60)] + ['action', 'name', 'num_times']
df = pd.DataFrame(all_data, columns=columns)

# Export the DataFrame to an Excel file
output_file = 'combined_data.xlsx'
df.to_excel(output_file, index=False)

print(f"Data combined and saved to {output_file}")
```

2. Added Features

```
# Add new features - distance of all points from join 11. This will add 19 features.
for i in range(preprocessed_data.shape[0]):
#for i in range(10):
  for j in range(1,21):
    if j != 11:
      distance = np.sqrt((preprocessed_data.iloc[i][f'x{j}'] - preprocessed_data.iloc[i]['x11'])**2 +
                         (preprocessed_data.iloc[i][f'y{j}'] - preprocessed_data.iloc[i]['y11'])**2 +
                         (preprocessed_data.iloc[i][f'z{j}'] - preprocessed_data.iloc[i]['z11'])**2)

      preprocessed_data.at[i, f'Dist_Join{j}_Join11'] = distance

preprocessed_data.head()
```

	x1	y1	z1	x2	y2	z2	x3	y3	z3	x4	...	Dist_Join10_Join11	Dist_Join12_Join11	Dist_Join13_Join11	D
0	-0.377219	0.676463	2.477132	-0.367249	0.493018	2.566909	-0.529421	0.355840	2.576270	-0.194301	...	0.735677	0.069126	0.167921	
1	-0.376956	0.676517	2.476770	-0.366931	0.492976	2.566630	-0.529115	0.355951	2.576212	-0.194166	...	0.741351	0.069121	0.167936	
2	-0.377076	0.676532	2.476376	-0.366147	0.492745	2.566584	-0.528687	0.355584	2.575751	-0.194098	...	0.747543	0.069094	0.168020	
3	-0.377247	0.675063	2.476167	-0.366134	0.492734	2.566559	-0.528485	0.355650	2.575728	-0.194087	...	0.764618	0.069098	0.168040	
4	-0.377483	0.674604	2.476354	-0.366084	0.492718	2.566469	-0.528106	0.355623	2.575532	-0.194132	...	0.766797	0.069101	0.168014	

5 rows × 81 columns

3. Split dataset into test, train and validation sets

```
# Splitting Dataset into training, validation and testing sets
from sklearn.model_selection import train_test_split
X = preprocessed_data.iloc[:, :-2]
y = preprocessed_data.iloc[:, -2:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, random_state=42)


print(X_train.shape, X_test.shape, X_val.shape)

print(y_train.shape, y_test.shape, y_val.shape)
```

```
(140737, 79) (30158, 79) (30159, 79)
(140737, 2) (30158, 2) (30159, 2)
```

4. Normalization

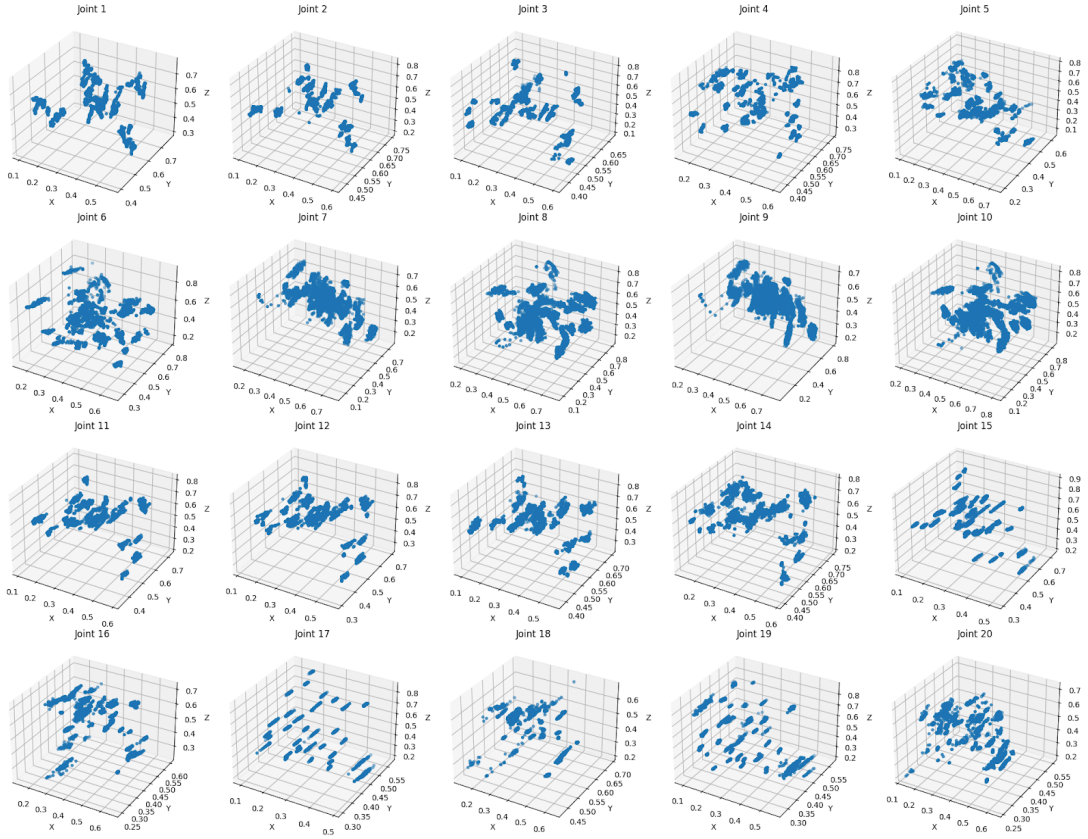
```
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)
```

5. No missing data

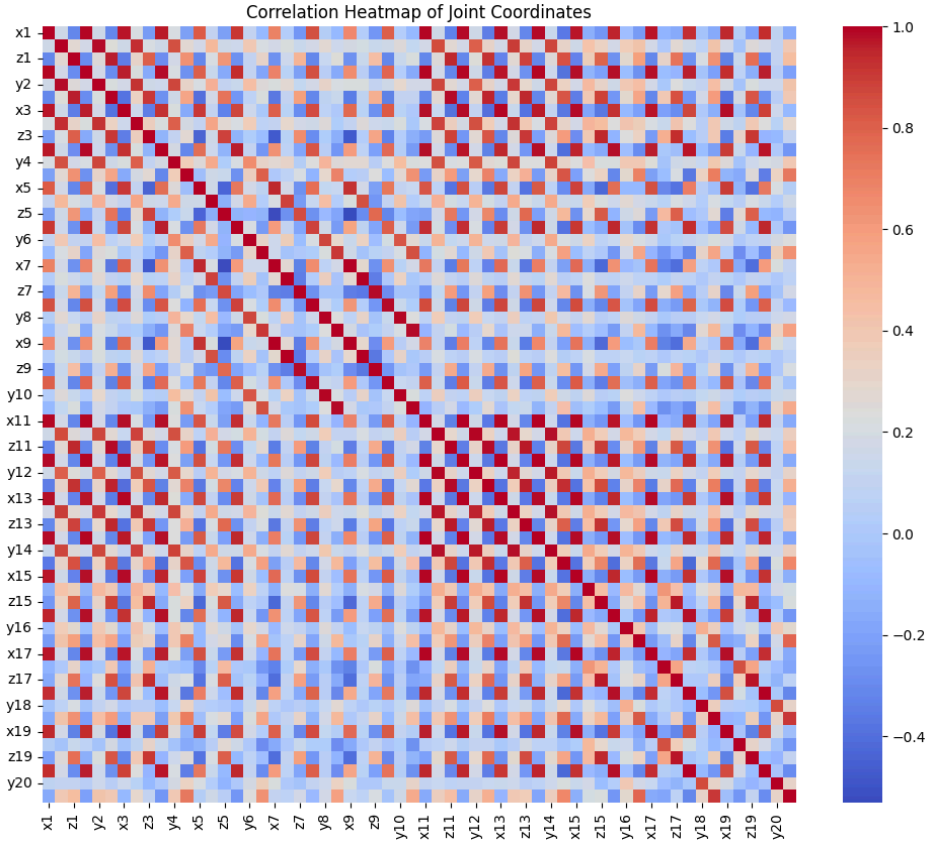
DATA VISUALIZATION

The most meaningful visualizations we performed:

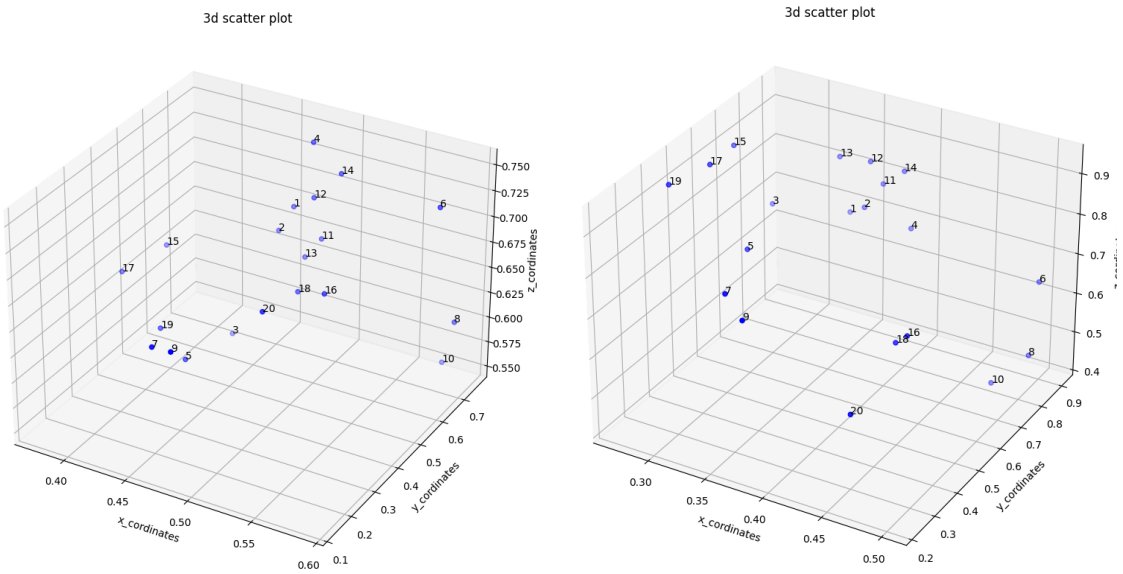
Example of 3D Scatter Plot of Joints' coordinates (of a specific Label):



Correlation Matrix:



Example of scatter Plot Joints' coordinates of Afternoon gesture for different frame: (Frame 1 and 7)



Variance Heatmap for gesture by person and Joints' coordinates ('Dark' Label):