Gesture Recognition Reading Muscle Activity

Project Phase 2 - Preprocessing Pipelines and EDA

DS5230 / Spring 2024 Semester

Team Members: Liyang Song

Mar 08, 2024

Reproducibility

All works are completed within Jupyter Notebooks and have been fully executed. To reproduce:

- Download the **environment.yml** and recreate **project_venv** virtual environment using the command:

  (base) : conda env create -f environment.yml

- Activate **project_venv** using the command:

  (base) : conda activate project_venv

- Create an ipykernel using the command:

  (project_venv) : python -m ipykernel install --user –name project_venv --display-name "Python (project_venv)"

- Open **project_phase_2.ipynb** in browser, click **restart the kernel and rerun all cells** button to reproduce all works.

Preprocessing - Establish machine learning attribute configuration

```
missing_attrs = processing.identify_missing_attrs(df, missing_threshold=MISSING_THRESHOLD)
missing_attrs
```

```
[]
```

- Identify attributes with missing value count greater than 0.20.

```
concern_list = general.check_for_complete_unique_attrs(df)
concern_list
```

```
the data frame has 11678 rows

id has 11678 unique values and is dtype int64 examine more closely
muscle reading 1 sensor 1 has 191 unique values and is dtype float64
muscle reading 1 sensor 2 has 154 unique values and is dtype float64
muscle reading 1 sensor 3 has 61 unique values and is dtype float64
muscle reading 1 sensor 4 has 102 unique values and is dtype float64
muscle reading 1 sensor 5 has 161 unique values and is dtype float64
muscle reading 1 sensor 6 has 213 unique values and is dtype float64
muscle reading 1 sensor 7 has 246 unique values and is dtype float64
muscle reading 1 sensor 8 has 177 unique values and is dtype float64
```

- Identify non machine learning attributes .

```
non_ml_attrs = ['id']
```

```
ml_drop_attrs = []
```

- No other attributes to drop from machine learning.

Preprocessing - Establish machine learning attribute configuration

```
numerical_attrs, nominal_attrs = processing.split_numerical_nominal(df)
print("numerical_attrs: ", numerical_attrs)
print("nominal_attrs: ", nominal_attrs)
```

```
numerical_attrs:  ['id', 'muscle reading 1 sensor 1', 'muscle reading 1 sensor 2', 'muscl
reading 1 sensor 7', 'muscle reading 1 sensor 8', 'muscle reading 2 sensor 1', 'muscle re
ding 2 sensor 6', 'muscle reading 2 sensor 7', 'muscle reading 2 sensor 8', 'muscle readi
g 3 sensor 5', 'muscle reading 3 sensor 6', 'muscle reading 3 sensor 7', 'muscle reading
sensor 4', 'muscle reading 4 sensor 5', 'muscle reading 4 sensor 6', 'muscle reading 4 se
sor 3', 'muscle reading 5 sensor 4', 'muscle reading 5 sensor 5', 'muscle reading 5 senso
2', 'muscle reading 6 sensor 3', 'muscle reading 6 sensor 4', 'muscle reading 6 sensor 5'
'muscle reading 7 sensor 2', 'muscle reading 7 sensor 3', 'muscle reading 7 sensor 4', 'm
scle reading 8 sensor 1', 'muscle reading 8 sensor 2', 'muscle reading 8 sensor 3', 'musc
e reading 8 sensor 8']
nominal_attrs:  []
```

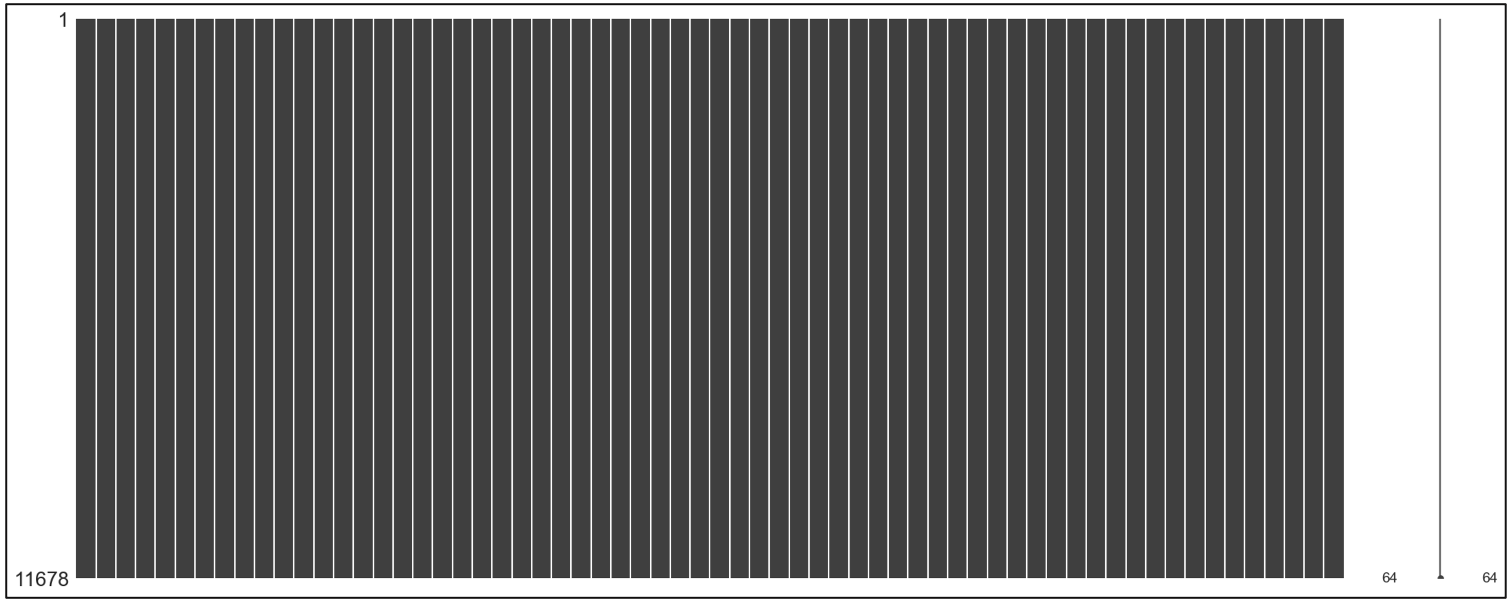
- Establish numerical and nominal attributes.

```
ml_ignore_list = missing_attrs + non_ml_attrs + ml_drop_attrs
print(f'\nml_ignore_list: {ml_ignore_list}')

numerical_attrs = [attr for attr in numerical_attrs if attr not in ml_ignore_list]
nominal_attrs = [attr for attr in nominal_attrs if attr not in ml_ignore_list]

assert(df.shape[1] == len(ml_ignore_list) + len(numerical_attrs) + len(nominal_attrs))
```

```
ml_ignore_list: ['id']
```

- Machine learning attribute configuration.

Preprocessing - Check out the missingness of attributes



- There is no missing value in all attributes, which is also shown in the *missingno* matrix.

Preprocessing - Checkout the types of attributes

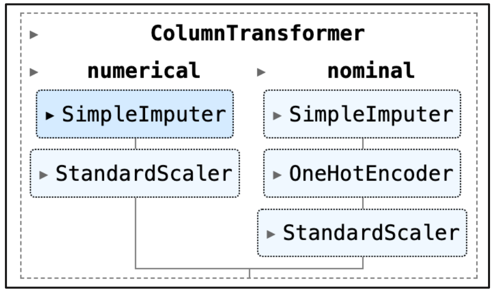
```
df[numerical_attrs + nominal_attrs].dtypes
```

```
muscle reading 1 sensor 1    float64
muscle reading 1 sensor 2    float64
muscle reading 1 sensor 3    float64
muscle reading 1 sensor 4    float64
muscle reading 1 sensor 5    float64
                               ...
muscle reading 8 sensor 4    float64
muscle reading 8 sensor 5    float64
muscle reading 8 sensor 6    float64
muscle reading 8 sensor 7    float64
muscle reading 8 sensor 8    float64
Length: 64, dtype: object
```

- All 64 machine learning attributes are *float64* type.

Preprocessing - Build a preprocessing pipeline

```python
def get_default_preprocessor(numerical_attrs: list, nominal_attrs: list)
    numerical_transformer = Pipeline([
        ('simple_imputer', SimpleImputer()),
        ('standard_scaler', StandardScaler())
    ])

    # Use OneHot Encoder to transform nominal attrs.
    # Target Encoder cannot be used as this is unsupervised.
    nominal_transformer = Pipeline([
        ('simple_imputer', SimpleImputer(strategy='most_frequent')),
        ('oneHot_encoder', OneHotEncoder()),
        ('standard_scaler', StandardScaler())
    ])

    preprocessor = ColumnTransformer([
        ("numerical", numerical_transformer, numerical_attrs),
        ("nominal", nominal_transformer, nominal_attrs)
    ])

    return preprocessor
```



- As this is an unsupervised machine learning task, *TargetEncoder* cannot be used and *OneHotEncoder* was applied.
- Although there is no missing value and no nominal attributes, *SimpleImputer* and *nominal_transformer* were applied in the pipeline for the new data in the future.

Preprocessing - Apply the preprocessing pipeline to the data frame

```
df_transformed = pd.DataFrame(
    data=preprocessor.fit_transform(df),
    columns=numerical_attrs + nominal_attrs
)
print(df_transformed.shape)
df_transformed.head()
```

```
(11678, 64)
```

	muscle reading 1 sensor 1	muscle reading 1 sensor 2	muscle reading 1 sensor 3	muscle reading 1 sensor 4	muscle reading 1 sensor 5	muscle reading 1 sensor 6	muscle reading 1 sensor 7	muscle reading 1 sensor 8	muscle reading 2 sensor 1	muscle reading 2 sensor 2	...
0	1.428445	0.401724	1.150179	1.173139	-0.047110	-0.482211	-4.293826	-4.240470	-0.459858	0.224160	...
1	-2.503492	-0.448155	-0.853937	-0.842622	0.737220	-0.017247	1.445765	-0.606050	0.588529	-0.282565	...
2	-0.995352	-0.618131	-1.455172	-0.977006	-1.167581	-0.210982	-3.098077	0.821758	0.036746	0.477522	...
3	0.135753	0.316736	0.148121	0.366835	0.008913	0.873935	4.275702	-0.865651	-0.846106	-0.113656	...
4	0.351202	0.061772	0.148121	-0.170702	-0.775416	0.408971	-1.982046	0.367456	0.422994	0.055252	...

5 rows × 64 columns

EDA – Non-graphical analysis

```
df_transformed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11678 entries, 0 to 11677
Data columns (total 64 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   muscle reading 1 sensor 1  11678 non-null  float64
 1   muscle reading 1 sensor 2  11678 non-null  float64
 2   muscle reading 1 sensor 3  11678 non-null  float64
 3   muscle reading 1 sensor 4  11678 non-null  float64
 4   muscle reading 1 sensor 5  11678 non-null  float64
 5   muscle reading 1 sensor 6  11678 non-null  float64
 6   muscle reading 1 sensor 7  11678 non-null  float64
 7   muscle reading 1 sensor 8  11678 non-null  float64
 8   muscle reading 2 sensor 1  11678 non-null  float64
 9   muscle reading 2 sensor 2  11678 non-null  float64
 10  muscle reading 2 sensor 3  11678 non-null  float64
 11  muscle reading 2 sensor 4  11678 non-null  float64
 12  muscle reading 2 sensor 5  11678 non-null  float64
 13  muscle reading 2 sensor 6  11678 non-null  float64
 14  muscle reading 2 sensor 7  11678 non-null  float64
 15  muscle reading 2 sensor 8  11678 non-null  float64
 16  muscle reading 3 sensor 1  11678 non-null  float64
 17  muscle reading 3 sensor 2  11678 non-null  float64
 18  muscle reading 3 sensor 3  11678 non-null  float64
 19  muscle reading 3 sensor 4  11678 non-null  float64
 20  muscle reading 3 sensor 5  11678 non-null  float64
 21  muscle reading 3 sensor 6  11678 non-null  float64
 22  muscle reading 3 sensor 7  11678 non-null  float64
 23  muscle reading 3 sensor 8  11678 non-null  float64
 24  muscle reading 4 sensor 1  11678 non-null  float64
```

```
df_transformed.describe()
```

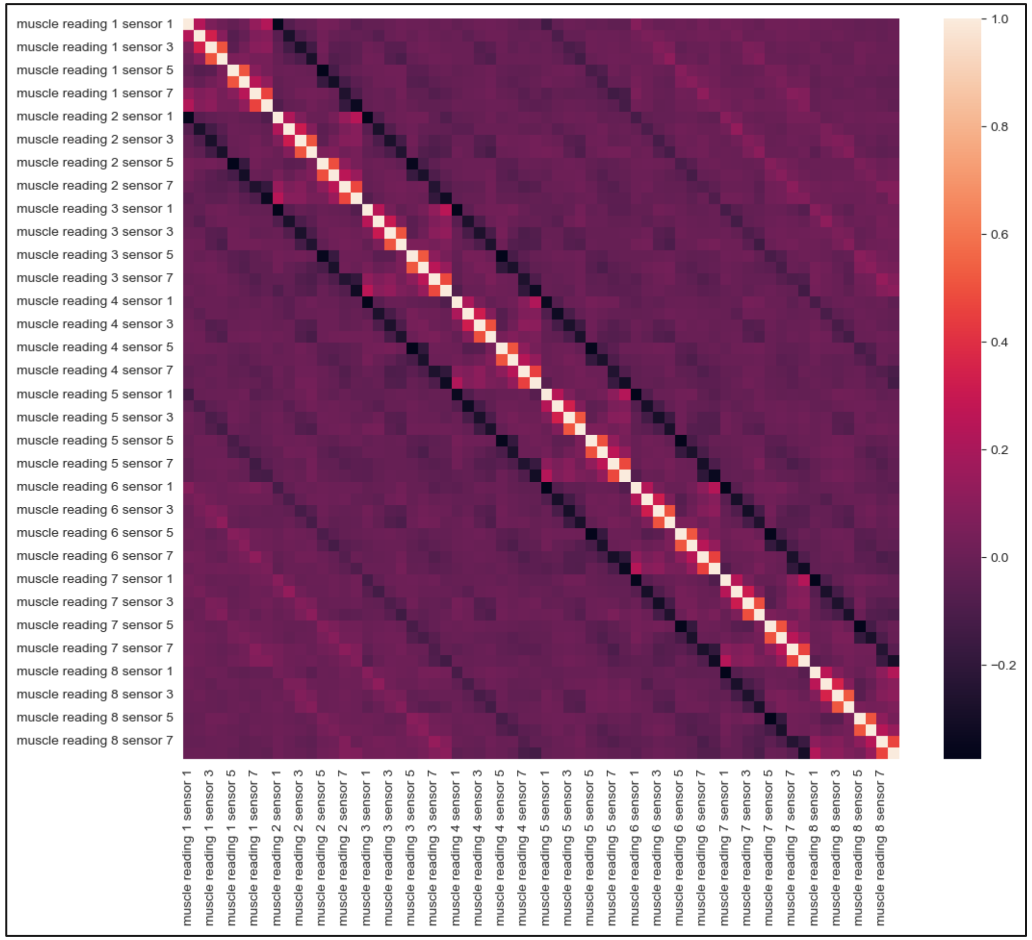
	muscle reading 1 sensor 1	muscle reading 1 sensor 2	muscle reading 1 sensor 3	muscle reading 1 sensor 4	muscle reading 1 sensor 5	muscle reading 1 sensor 6	muscle reading 1 sensor 7	muscle reading 1 sensor 8	muscle reading 2 sensor 1	muscle reading 2 sensor 2	...
count	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	1.167800e+04	...
mean	-5.323899e-18	4.487286e-18	-4.411230e-18	7.301347e-18	-2.068715e-17	7.301347e-18	-1.216891e-18	1.916604e-17	-1.041963e-17	4.563342e-18	...
std	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	1.000043e+00	...
min	-6.219980e+00	-8.776974e+00	-6.465463e+00	-9.980741e+00	-6.769936e+00	-4.705639e+00	-5.051133e+00	-8.264292e+00	-6.032863e+00	-1.075487e+01	...
25%	-4.567305e-01	-2.781794e-01	-4.531142e-01	-4.394698e-01	-5.513220e-01	-5.597053e-01	-1.884240e-01	-4.762490e-01	-4.598582e-01	-2.825647e-01	...
50%	-2.583335e-02	-2.321558e-02	-5.229099e-02	-3.631754e-02	8.913491e-03	-1.724671e-02	1.086733e-02	-2.194652e-02	-1.843213e-02	-2.920240e-02	...
75%	4.050638e-01	3.167362e-01	5.489439e-01	5.012189e-01	5.691490e-01	5.252118e-01	2.101587e-01	4.323560e-01	3.678157e-01	3.086139e-01	...
max	6.006727e+00	7.710688e+00	6.962116e+00	7.489192e+00	5.163080e+00	4.942374e+00	5.112726e+00	8.220399e+00	7.044386e+00	9.007385e+00	...

8 rows × 64 columns

```
df_transformed.isnull().sum().sum()
```

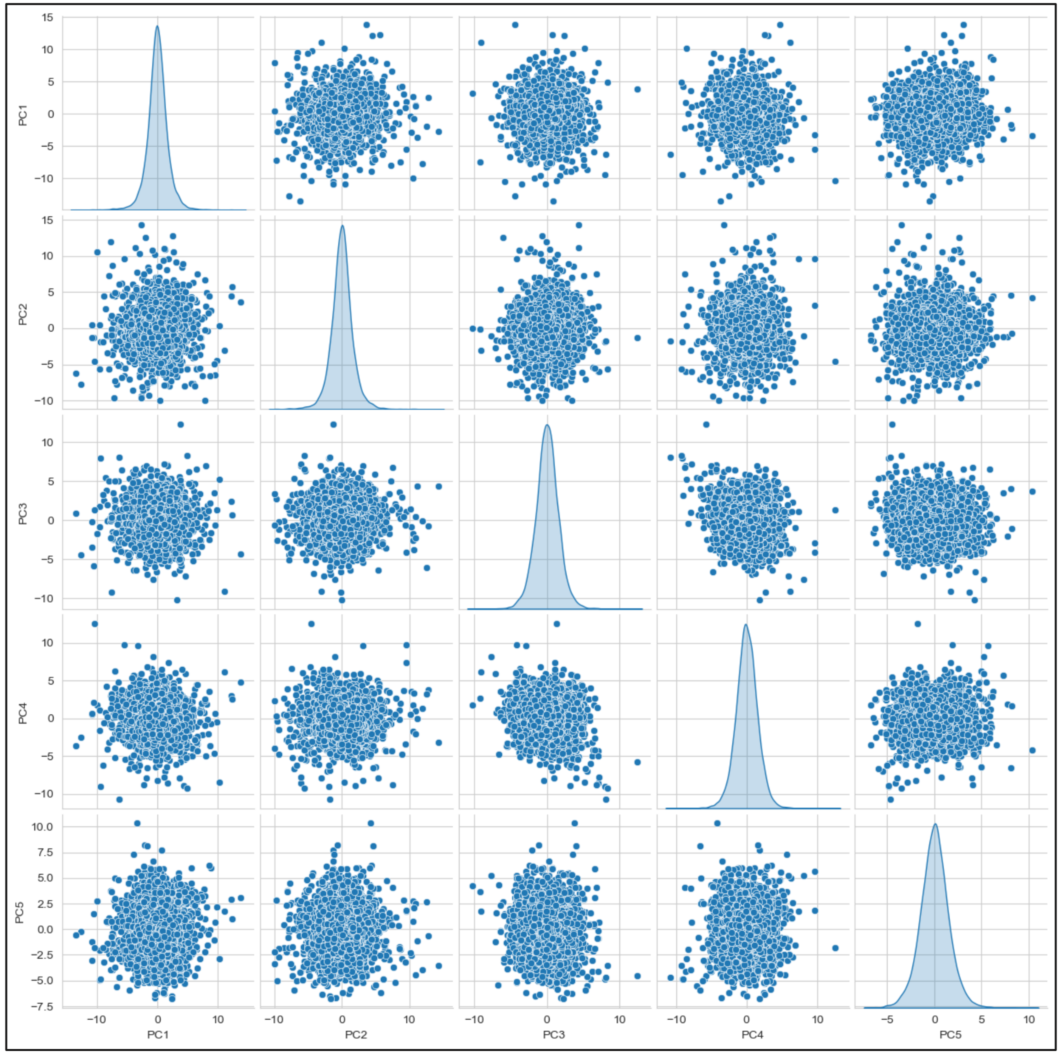
```
0
```

EDA - Pair plots to look for possible clusters



- It's not practical to draw pair plots for all 64 columns, which will generate 64 * 64 / 2 = 2048 scatter plots.
- We tried using correlation heatmap to check the correlations between attrs, and drawing pair plots for principal components.
- The heatmap does not reveal extremely high correlations. Some relatively high correlations show between adjacent sensors in the same reading (e.g. sensor 3 and 4 in reading 1). This may be due to the muscle movement affecting adjacent sensors.
- Most correlations between attrs are relatively low, which is normal for a high-dimension dataset.

EDA - Pair plots to look for possible clusters

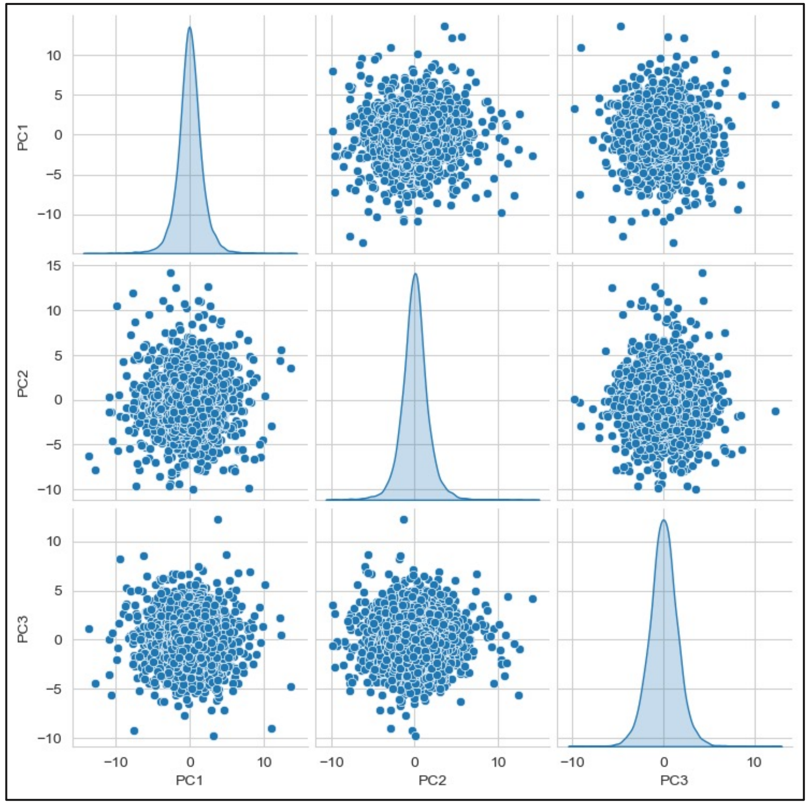


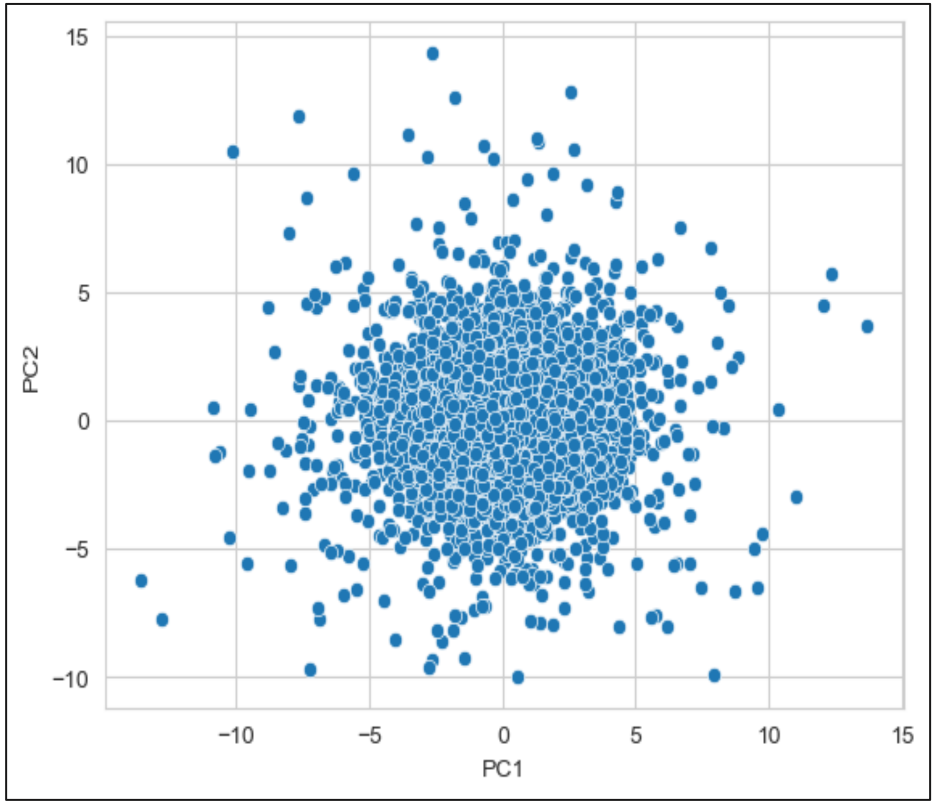

```
pca_pipeline = Pipeline([
    ('standard_scaler', StandardScaler()),
    ('PCA', PCA(n_components=5))
])

df_copy = df_transformed[numerical_attrs].copy()
pca_components = pca_pipeline.fit_transform(df_copy)

pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5'])
```

```
sns.pairplot(pca_df, diag_kind = 'kde')
plt.show()
```

- Use PCA to reduce dimensionality to 5 and then draw pair plots for principal components.
- The plots do not show very high correlations between pairs, whether linear or non-linear.

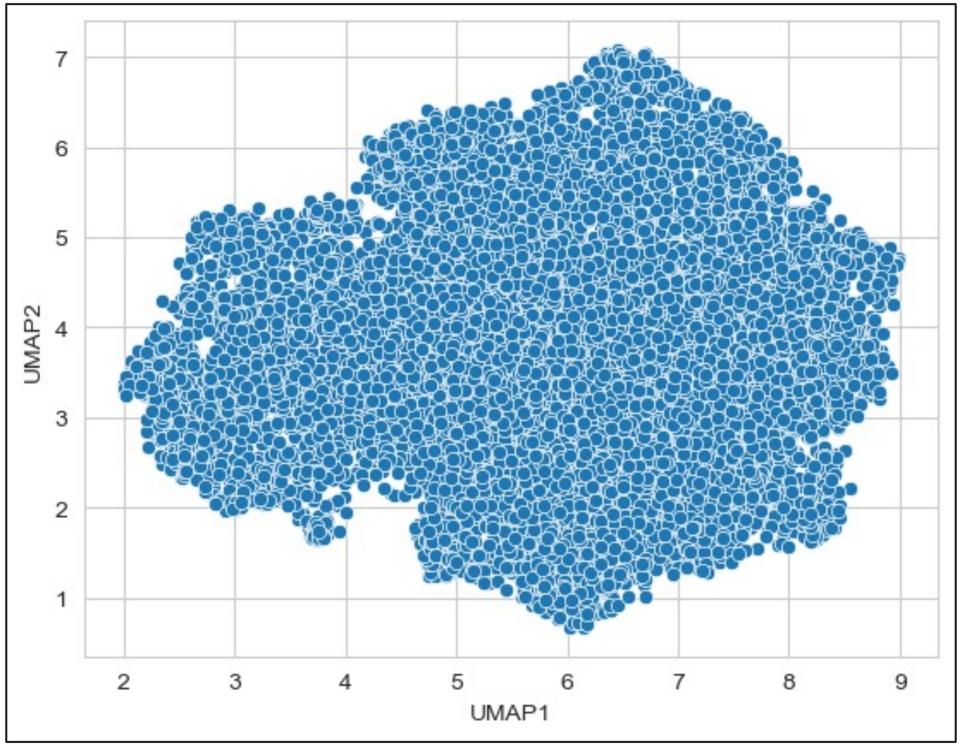
EDA - Pair plots to look for possible clusters





- We also tried PCA to reduce dimensionality to 3 and 2 and then draw pair plots for them. Still, no distinct clusters are shown in the plots.

EDA - Pair plots to look for possible clusters

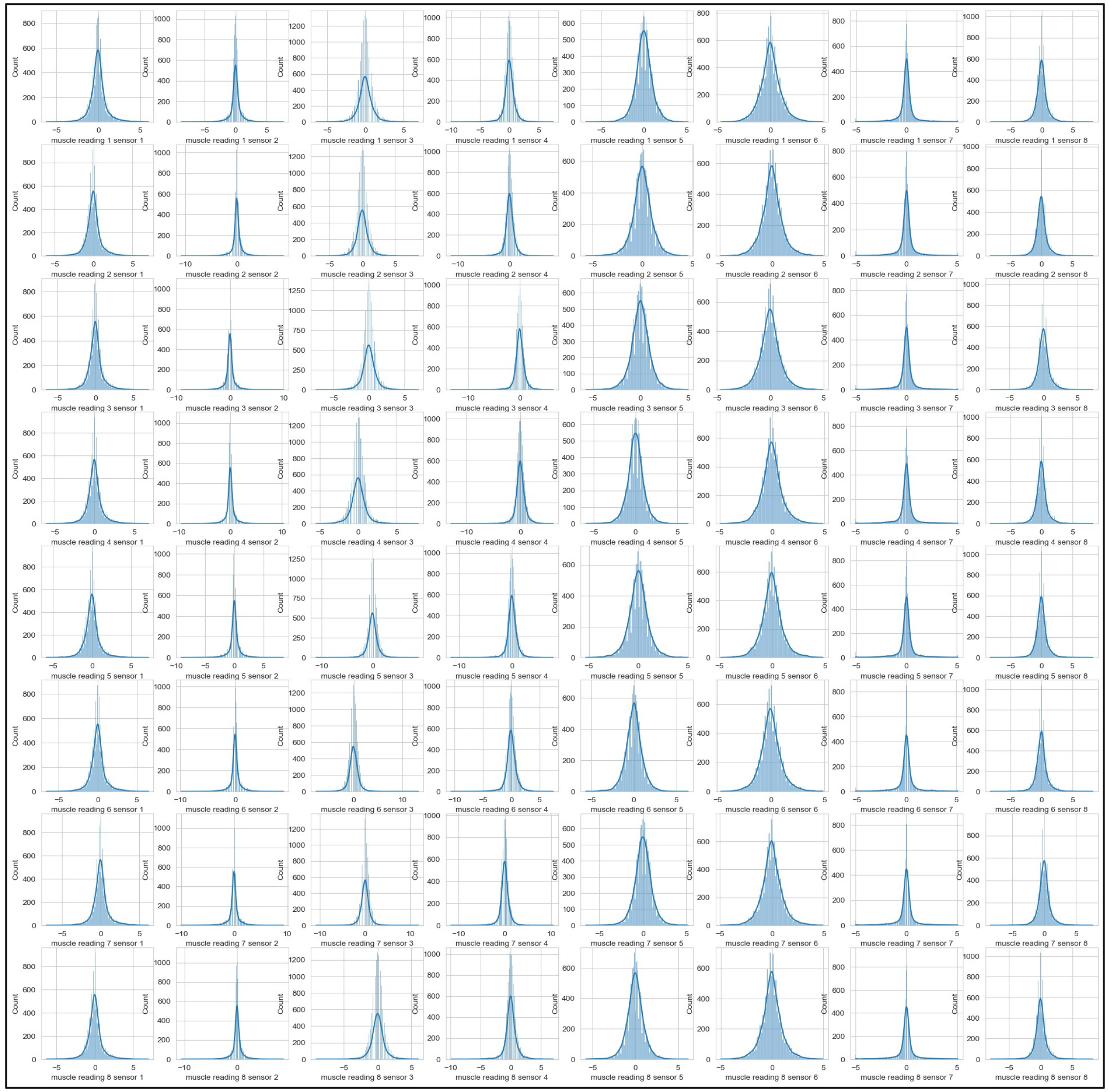



```
from umap import UMAP

umap_pipeline = Pipeline([
    ('standard_scaler', StandardScaler()),
    ('UMAP', UMAP(n_components=2))
])

df_copy = df_transformed[numerical_attrs].copy()
umap_components = umap_pipeline.fit_transform(df_copy)

umap_df = pd.DataFrame(data=umap_components, columns=['UMAP1', 'UMAP2'])
```

```
sns.scatterplot(umap_df, x='UMAP1', y='UMAP2')
plt.show()
```

- As PCA is linear combination of attrs, we tried using UMAP as the non-linear dimension reduction method. Drawing plot between two UMAP principal components also did not revear distinct clusters.

EDA - Histograms to look for possible clusters



- The histograms show some similar distributions. Most attrs follow a Gaussian distribution pattern.
- The variance among attrs are relatively uniform, which may add the difficulty to find distinct clusters.
- In the next phase, we will try tuning the UMAP hyperparameters, or try more different algorithms like K-Means to find the clusters.