ACasaccioDSC630 - 4.2 Clustering Exercise

April 4, 2024

DSC630 - 4.4: Clustering Exercise

Author: Alysen Casaccio

Due Date: 4/7/2024

[36]:
```
#import statements
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
import warnings
import os
```

[32]:
```
# setting the number of threads as suggested in the KMeans algorithm to avoid␣
 ↪memory leakage
os.environ["OMP_NUM_THREADS"] = "9"

# suppressing warnings to prevent cluttered output related to KMeans memory␣
 ↪leakage/multiple threads
warnings.filterwarnings(action='ignore', category=UserWarning, module='sklearn')
```

[2]:
```
# loading and reading
file_path = 'C:/Users/alyse/OneDrive/Documents/Bellevue University/DSC 630 -␣
 ↪Predictive Analytics/Data Sources/als_data.csv'
alspatients = pd.read_csv(file_path)
```

[6]:
```
# checking the column list
print(alspatients.columns.tolist())
```

```
['ID', 'Age_mean', 'Albumin_max', 'Albumin_median', 'Albumin_min',
'Albumin_range', 'ALSFRS_slope', 'ALSFRS_Total_max', 'ALSFRS_Total_median',
'ALSFRS_Total_min', 'ALSFRS_Total_range', 'ALT.SGPT._max', 'ALT.SGPT._median',
'ALT.SGPT._min', 'ALT.SGPT._range', 'AST.SGOT._max', 'AST.SGOT._median',
'AST.SGOT._min', 'AST.SGOT._range', 'Bicarbonate_max', 'Bicarbonate_median',
```

```
'Bicarbonate_min', 'Bicarbonate_range', 'Blood.Urea.Nitrogen..BUN._max',
'Blood.Urea.Nitrogen..BUN._median', 'Blood.Urea.Nitrogen..BUN._min',
'Blood.Urea.Nitrogen..BUN._range', 'bp_diastolic_max', 'bp_diastolic_median',
'bp_diastolic_min', 'bp_diastolic_range', 'bp_systolic_max',
'bp_systolic_median', 'bp_systolic_min', 'bp_systolic_range', 'Calcium_max',
'Calcium_median', 'Calcium_min', 'Calcium_range', 'Chloride_max',
'Chloride_median', 'Chloride_min', 'Chloride_range', 'Creatinine_max',
'Creatinine_median', 'Creatinine_min', 'Creatinine_range', 'Gender_mean',
'Glucose_max', 'Glucose_median', 'Glucose_min', 'Glucose_range', 'hands_max',
'hands_median', 'hands_min', 'hands_range', 'Hematocrit_max',
'Hematocrit_median', 'Hematocrit_min', 'Hematocrit_range', 'Hemoglobin_max',
'Hemoglobin_median', 'Hemoglobin_min', 'Hemoglobin_range', 'leg_max',
'leg_median', 'leg_min', 'leg_range', 'mouth_max', 'mouth_median', 'mouth_min',
'mouth_range', 'onset_delta_mean', 'onset_site_mean', 'Platelets_max',
'Platelets_median', 'Platelets_min', 'Potassium_max', 'Potassium_median',
'Potassium_min', 'Potassium_range', 'pulse_max', 'pulse_median', 'pulse_min',
'pulse_range', 'respiratory_max', 'respiratory_median', 'respiratory_min',
'respiratory_range', 'Sodium_max', 'Sodium_median', 'Sodium_min',
'Sodium_range', 'SubjectID', 'trunk_max', 'trunk_median', 'trunk_min',
'trunk_range', 'Urine.Ph_max', 'Urine.Ph_median', 'Urine.Ph_min']
```

[8]:
```
# listing columns to drop, starting with ID, generic blood work, and vital␣
 ↪assessments, excluding respiratory-related columns.
columns_to_drop = ['Albumin_max', 'Albumin_median', 'Albumin_min',␣
 ↪'Albumin_range',
                   'ALT.SGPT._max', 'ALT.SGPT._median', 'ALT.SGPT._min', 'ALT.
 ↪SGPT._range',
                   'AST.SGOT._max', 'AST.SGOT._median', 'AST.SGOT._min', 'AST.
 ↪SGOT._range',
                   'Blood.Urea.Nitrogen..BUN._max', 'Blood.Urea.Nitrogen..BUN.
 ↪_median',
                   'Blood.Urea.Nitrogen..BUN._min', 'Blood.Urea.Nitrogen..BUN.
 ↪_range',
                   'bp_diastolic_max', 'bp_diastolic_median',␣
 ↪'bp_diastolic_min', 'bp_diastolic_range',
                   'bp_systolic_max', 'bp_systolic_median', 'bp_systolic_min',␣
 ↪'bp_systolic_range',
                   'Calcium_max', 'Calcium_median', 'Calcium_min',␣
 ↪'Calcium_range',
                   'Chloride_max', 'Chloride_median', 'Chloride_min',␣
 ↪'Chloride_range',
                   'Creatinine_max', 'Creatinine_median', 'Creatinine_min',␣
 ↪'Creatinine_range',
                   'Glucose_max', 'Glucose_median', 'Glucose_min',␣
 ↪'Glucose_range',
```

```
                  'Hematocrit_max', 'Hematocrit_median', 'Hematocrit_min',␣
 ↪'Hematocrit_range',
                  'Hemoglobin_max', 'Hemoglobin_median', 'Hemoglobin_min',␣
 ↪'Hemoglobin_range',
                  'Platelets_max', 'Platelets_median', 'Platelets_min',
                  'Potassium_max', 'Potassium_median', 'Potassium_min',␣
 ↪'Potassium_range',
                  'Sodium_max', 'Sodium_median', 'Sodium_min', 'Sodium_range',
                  'Urine.Ph_max', 'Urine.Ph_median', 'Urine.Ph_min', 'ID']

# dropping the columns from the dataframe
alspatients = alspatients.drop(columns=columns_to_drop)
```

[10]:
```
#rechecking the columns in the DF
print(alspatients.columns.tolist())
```

```
['Age_mean', 'ALSFRS_slope', 'ALSFRS_Total_max', 'ALSFRS_Total_median',
'ALSFRS_Total_min', 'ALSFRS_Total_range', 'Bicarbonate_max',
'Bicarbonate_median', 'Bicarbonate_min', 'Bicarbonate_range', 'Gender_mean',
'hands_max', 'hands_median', 'hands_min', 'hands_range', 'leg_max',
'leg_median', 'leg_min', 'leg_range', 'mouth_max', 'mouth_median', 'mouth_min',
'mouth_range', 'onset_delta_mean', 'onset_site_mean', 'pulse_max',
'pulse_median', 'pulse_min', 'pulse_range', 'respiratory_max',
'respiratory_median', 'respiratory_min', 'respiratory_range', 'SubjectID',
'trunk_max', 'trunk_median', 'trunk_min', 'trunk_range']
```

Reducing Redundant Variables with a Correlation Matrix Assessment (Abbot, 2014)

[12]:
```
# generating a correlation matrix
corr_matrix = alspatients.corr()

# identifying the highly correlated pairs
high_corr_var=np.where(np.abs(corr_matrix) > 0.8)
high_corr_var=[(corr_matrix.columns[x], corr_matrix.columns[y]) for x,y in␣
 ↪zip(*high_corr_var) if x!=y and x<y]

# displaying the highly correlated pairings
for var_pair in high_corr_var:
    print(f"Highly correlated pair: {var_pair}")
```

```
Highly correlated pair: ('ALSFRS_slope', 'ALSFRS_Total_range')
Highly correlated pair: ('ALSFRS_Total_max', 'ALSFRS_Total_median')
Highly correlated pair: ('ALSFRS_Total_max', 'trunk_max')
Highly correlated pair: ('ALSFRS_Total_median', 'ALSFRS_Total_min')
Highly correlated pair: ('ALSFRS_Total_median', 'trunk_median')
Highly correlated pair: ('ALSFRS_Total_min', 'trunk_min')
Highly correlated pair: ('ALSFRS_Total_range', 'trunk_range')
Highly correlated pair: ('hands_max', 'hands_median')
Highly correlated pair: ('hands_median', 'hands_min')
```

```
Highly correlated pair: ('hands_min', 'trunk_min')
Highly correlated pair: ('leg_max', 'leg_median')
Highly correlated pair: ('leg_median', 'leg_min')
Highly correlated pair: ('mouth_max', 'mouth_median')
Highly correlated pair: ('mouth_median', 'mouth_min')
Highly correlated pair: ('mouth_min', 'mouth_range')
Highly correlated pair: ('respiratory_min', 'respiratory_range')
Highly correlated pair: ('trunk_max', 'trunk_median')
```

[14]:
```
# listing more columns to drop
columns_to_drop = [
    'ALSFRS_Total_range', 'ALSFRS_Total_max', 'ALSFRS_Total_min',
    'trunk_range', 'hands_max', 'hands_min', 'trunk_min',
    'leg_max', 'leg_min', 'mouth_max', 'mouth_min',
    'mouth_range', 'respiratory_range', 'trunk_max']

# dropping the columns
alspatients = alspatients.drop(columns=columns_to_drop)
```

[15]:
```
# final column check
print(alspatients.columns.tolist())
```

```
['Age_mean', 'ALSFRS_slope', 'ALSFRS_Total_median', 'Bicarbonate_max',
'Bicarbonate_median', 'Bicarbonate_min', 'Bicarbonate_range', 'Gender_mean',
'hands_median', 'hands_range', 'leg_median', 'leg_range', 'mouth_median',
'onset_delta_mean', 'onset_site_mean', 'pulse_max', 'pulse_median', 'pulse_min',
'pulse_range', 'respiratory_max', 'respiratory_median', 'respiratory_min',
'SubjectID', 'trunk_median']
```

[21]:
```
# dataframe review
print(alspatients.info())
alspatients.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2223 entries, 0 to 2222
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age_mean             2223 non-null   int64
 1   ALSFRS_slope         2223 non-null   float64
 2   ALSFRS_Total_median  2223 non-null   float64
 3   Bicarbonate_max      2223 non-null   float64
 4   Bicarbonate_median   2223 non-null   float64
 5   Bicarbonate_min      2223 non-null   float64
 6   Bicarbonate_range    2223 non-null   float64
 7   Gender_mean          2223 non-null   int64
 8   hands_median         2223 non-null   float64
 9   hands_range          2223 non-null   float64
 10  leg_median           2223 non-null   float64
```

```
 11  leg_range            2223 non-null   float64
 12  mouth_median         2223 non-null   float64
 13  onset_delta_mean     2223 non-null   int64
 14  onset_site_mean      2223 non-null   int64
 15  pulse_max            2223 non-null   int64
 16  pulse_median         2223 non-null   float64
 17  pulse_min            2223 non-null   int64
 18  pulse_range          2223 non-null   float64
 19  respiratory_max      2223 non-null   int64
 20  respiratory_median   2223 non-null   float64
 21  respiratory_min      2223 non-null   int64
 22  SubjectID            2223 non-null   int64
 23  trunk_median         2223 non-null   float64
dtypes: float64(15), int64(9)
memory usage: 416.9 KB
None
```

[21]:

	Age_mean	ALSFRS_slope	ALSFRS_Total_median	Bicarbonate_max	\
0	65	-0.965608	28.0	30.0	
1	48	-0.921717	33.0	32.0	
2	38	-0.914787	14.0	35.0	
3	63	-0.598361	29.0	23.0	
4	63	-0.444039	27.5	32.0	
5	36	-0.118353	34.5	29.0	
6	55	-1.225580	24.0	32.0	
7	55	-0.760417	27.5	29.0	
8	37	-1.010148	28.5	36.0	
9	72	-0.107861	25.5	32.0	

	Bicarbonate_median	Bicarbonate_min	Bicarbonate_range	Gender_mean	\
0	28.0	25.0	0.017422	1	
1	28.0	25.0	0.012195	1	
2	29.0	24.0	0.019643	2	
3	20.0	20.0	0.007481	2	
4	28.0	23.0	0.014925	1	
5	26.0	22.0	0.012681	2	
6	27.5	23.0	0.016275	2	
7	28.0	25.0	0.010582	1	
8	29.0	20.0	0.028986	2	
9	29.5	27.0	0.008929	1	

	hands_median	hands_range	…	onset_site_mean	pulse_max	pulse_median	\
0	7.5	0.005291	…	1	79	68.0	
1	6.0	0.003591	…	1	90	76.0	
2	1.0	0.007143	…	1	82	73.0	
3	5.5	0.004988	…	2	84	72.0	
4	6.5	0.008489	…	2	101	96.0	

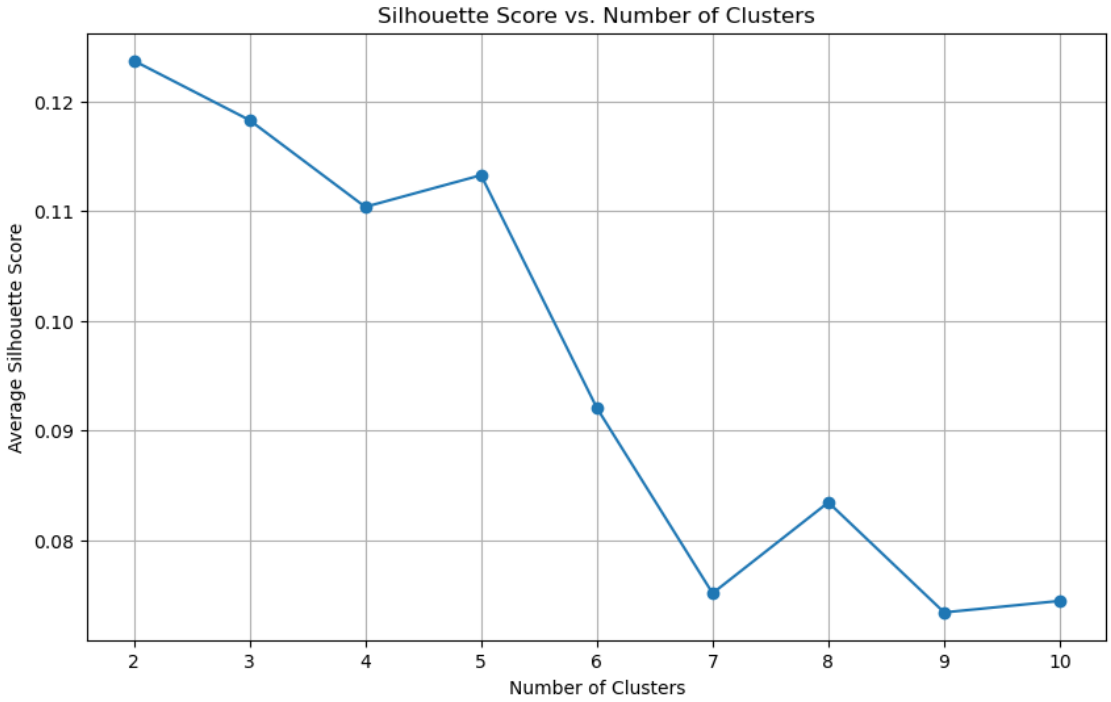
```
5           7.0     0.005435  …                1         88          66.0
6           4.0     0.009042  …                2         96          80.0
7           8.0     0.007937  …                2        100          80.0
8           1.5     0.010870  …                2         84          68.0
9           7.0     0.003571  …                2        100         100.0

   pulse_min  pulse_range  respiratory_max  respiratory_median  \
0         61     0.047619                4                 3.0
1         64     0.046679                4                 4.0
2         60     0.039286                4                 4.0
3         68     0.039900                3                 3.0
4         74     0.044776                4                 4.0
5         60     0.050725                4                 4.0
6         66     0.053191                4                 3.0
7         64     0.095238                4                 4.0
8         59     0.045290                4                 4.0
9         80     0.035714                4                 4.0

   respiratory_min  SubjectID  trunk_median
0                3        533           7.0
1                3        649           7.0
2                4       1234           0.0
3                3       2492           5.0
4                3       2956           4.0
5                3       3085           8.0
6                2       3551           5.0
7                1       3971           3.0
8                4       4390           3.0
9                4       4772           3.0

[10 rows x 24 columns]
```

[23]:
```
# excluding subject id from scaling
features_to_scale = alspatients.drop(columns=['SubjectID'])

# initializing the scaler
scaler = StandardScaler()

# fitting the scaler to the data/transform
scaled_features = scaler.fit_transform(features_to_scale)

# creating a new DF for the scaled data
als_scaled = pd.DataFrame(scaled_features, columns=features_to_scale.columns,␣
 ↪index=features_to_scale.index)
```

[33]:
```
# establishing the range of clusters to evaluate
range_n_clusters = list(range(2, 11))
```

```
# creating an empty list to store the average silhouette scores for n_clusters
silhouette_avg_scores = []

# iterating over the range of cluster numbers
for n_clusters in range_n_clusters:
    # initializing the KMeans model
    kmeans = KMeans(n_clusters=n_clusters, random_state=10)

    # fitting the model and predicting the cluster labels
    cluster_labels = kmeans.fit_predict(als_scaled)

    # calculating the silhouette score and appending to the list
    silhouette_avg = silhouette_score(als_scaled, cluster_labels)
    silhouette_avg_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is :␣
 ↪{silhouette_avg}")

# plotting the silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(range_n_clusters, silhouette_avg_scores, marker='o')
plt.title('Silhouette Score vs. Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Average Silhouette Score')
plt.xticks(range_n_clusters)
plt.grid(True)
plt.show()
```

```
For n_clusters = 2, the average silhouette_score is : 0.12371636450911432
For n_clusters = 3, the average silhouette_score is : 0.11829827260148983
For n_clusters = 4, the average silhouette_score is : 0.11038424073914906
For n_clusters = 5, the average silhouette_score is : 0.11327412713117564
For n_clusters = 6, the average silhouette_score is : 0.09206597736427868
For n_clusters = 7, the average silhouette_score is : 0.07516293284431551
For n_clusters = 8, the average silhouette_score is : 0.0834499506372277
For n_clusters = 9, the average silhouette_score is : 0.07341662119908254
For n_clusters = 10, the average silhouette_score is : 0.07446673455942485
```



Cluster Selection Understanding the silhouette scores is crucial for selecting the optimal number of clusters for K-Means clustering using the silhouette scoring method.
The silhouette score measures how similar an object is to its own cluster compared to other clusters. Higher values indicate better-defined clusters. In this case, I would want to choose 2 clusters, as they have the highest score and, therefore, the most distinct grouping.

If I wanted to cross-check this decision, I could also use the elbow curve method, which leverages the total within-cluster sum of squares (WSS) and measures the compactness of the clustering. Using both methods allows for more confident decision-making.

Fitting the K-Means

[34]:
```
# initializing the KMeans model with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=10)

# fitting the model to the data
kmeans.fit(als_scaled)

# predicting cluster labels for the data
cluster_labels = kmeans.predict(als_scaled)
```

Principal Component Analysis (PCA)

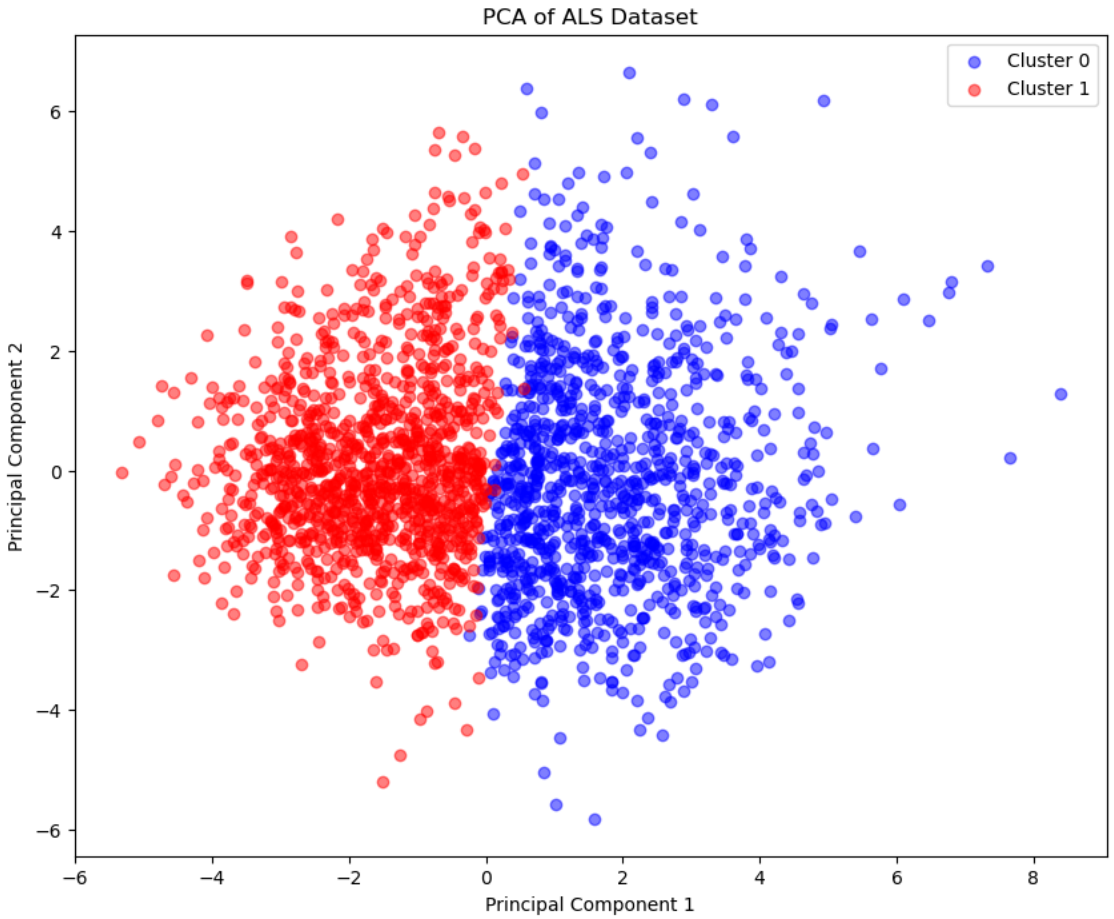
[38]:
```
# initializing PCA to reduce the data to 2 components
pca = PCA(n_components=2)
```

```
# fitting the PCA model to the scaled data/transforming
pca_features = pca.fit_transform(als_scaled)

# creating a new DF for the PCA-transformed data
pca_als = pd.DataFrame(pca_features, columns=['PCA1', 'PCA2'])
```

Scatter Plot

[41]:
```
# preparing the data for plotting
pca_als = pd.DataFrame(pca_features, columns=['PCA1', 'PCA2'])
pca_als['Cluster'] = cluster_labels  # adding the cluster labels to the PCA␣
 ↪DataFrame
```

[44]:
```
# two clusters, two colors
colors = ['blue', 'red']

# creating the scatter plot
plt.figure(figsize=(10, 8))
for i, color in enumerate(colors):
    # filtering data points based on their cluster label
    cluster_data = pca_als[pca_als['Cluster'] == i]
    plt.scatter(cluster_data['PCA1'], cluster_data['PCA2'], color=color,␣
 ↪alpha=0.5, label=f'Cluster {i}')

plt.title('PCA of ALS Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



Interpretation The PCA and clustering analysis have identified two main subgroups within the ALS dataset. These are differentiated by patterns captured in the first two principal components. The distinct separation between PC1 and PC2 suggests that these two components capture important aspects of variability within the ALS patient data. This could correlate with different stages of the disease process or other interpretations, depending on the specific features represented by these components.

I felt that the PCA process helped reduce the dimensionality of the data and certainly helped with visualization, but understanding the principal components themselves is challenging for me personally. Since the principal components must be combinations of the original features, further interpretation would require examining the PCA loadings and determining which of the original features contributed most to them.
Especially in healthcare, where I have a lot of domain knowledge, I think doing unsupervised learning techniques can lead to interventions or further research that I perhaps wouldn't have examined previously.

This analysis seems to have meaningful heterogeneity within the ALS patient population, at least for this dataset.
The obvious next steps are to develop deeper insights into the nature of these differences and how

they could be utilized in a clinical setting.

References: Dean Abbot, (2014). Applied Predictive Analytics: Principles and Techniques for the Professional Data Analyst. Indianapolis, IN: Wiley.

Wes McKinney, (2022) Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter, 3rd ed. Sebastopol, CA: O'Reilly

Python Software Foundation. Python Language Reference, version 3.9. Available at [1][1].

Cedarbaum JM, Stambler N, Malta E, et al. The ALSFRS-R: a revised ALS functional rating scale that incorporates assessments of respiratory function. J Neurol Sci. 1999;169(1-2):13–21

Štětkářová I, Ehler E. Diagnostics of Amyotrophic Lateral Sclerosis: Up to Date. Diagnostics (Basel). 2021 Feb 3;11(2):231. doi: 10.3390/diagnostics11020231. PMID: 33546386; PMCID: PMC7913557.