**Uitleg bij deze File:**  
Even nog over de StandardScaler functie, die zou ik nog altijd weglaten bij elk model. Als je kijkt naar de 10 random boxplots generated van de merged\_dataset:

Afbeelding met diagram, lijn, Parallel, tekst

Automatisch gegenereerde beschrijving

Dan zie je dat de data zich wel goed tusse -1 en 1 bevinden, dichtbij 0 vaak (wat betekent dat de data reeds gescaled is). Als je de boxplot van de oefeningenreeks bekijkt in kaggle voor standaardisatie dan ziet die er similair uit als onze hier, waardoor je zou denken dat we nog eens moeten scalen, maar op de y schaal daar varieert de data zeer ver van 0, wat erop wijst dat de data niet gestandaardiseerd is. Hier is de data wel aant schommelen rond 0 over het algemeen. (de boxplots schommelen als het waren wel rond dezelfde waarde, maar omdat de Y as zo klein is lijkt het denk ik gwn alsof de data niet gescaled is?)

Dan vervolgens heb ik Random forest models + logistic regression models gegenereerd (met en zonder PCA eerst toegepast en steeds de accurracy van het model in de title geschreven). Bij de randomforest models is het model zonder PCA accurater, maar bij logistic regression is de accuracy hetzelfde (0.94)…. Ik weet niet goed wat ik hiermee aan moet vangen?

Vervolgens heb ik ook een tSNE analyse nog gedaan op de data

# RANDOM FOREST

**1.1 RANDOM FOREST WITH PCA (Slightly Worse Accuracy (0.87))**

# Importing Required Libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

import shap

# Load the Data

mergeddata\_filepath = '../input/merged-data/merged\_data.csv'

mergeddata = pd.read\_csv(mergeddata\_filepath)

# Select Only Proteins Starting From 'O43704'

protein\_data = mergeddata.loc[:, 'O43704':]

# Define the Target (Tumor Subtype)

target = mergeddata['Proteomic\_Subtype']

# Split the Data into Training and Test Sets (70% Train, 30% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(protein\_data, target, test\_size=0.3, random\_state=1)

# Apply PCA for Dimensionality Reduction (Retain 95% Variance)

pca = PCA(n\_components=0.95) # Retain 95% of the variance

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

# Train a Random Forest Classifier

rf\_model = RandomForestClassifier(random\_state=1)

rf\_model.fit(X\_train\_pca, y\_train)

# Cross-Validation of the Random Forest (5-Fold Cross-Validation)

cv\_scores = cross\_val\_score(rf\_model, X\_train\_pca, y\_train, cv=5)

print(f"Random Forest Cross-validation Scores: {cv\_scores}")

print(f"Mean Cross-validation Accuracy: {cv\_scores.mean()}")

# Make Predictions

y\_pred = rf\_model.predict(X\_test\_pca)

# Evaluate the Model

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print the Evaluation Results

print(f"Accuracy: {accuracy}")

print(f"Classification Report:\n{classification\_rep}")

# Plot the Confusion Matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=rf\_model.classes\_)

disp.plot(cmap='Blues')

plt.show()

# Feature Importance After PCA

importances = rf\_model.feature\_importances\_

# Create a DataFrame of Important Proteins (Components)

important\_proteins = pd.DataFrame({

'Protein': [f'PC{i+1}' for i in range(X\_train\_pca.shape[1])],

'Importance': importances

}).sort\_values(by='Importance', ascending=False)

# Show the Top 10 Most Important Components

print("\nTop 10 Most Important Components:")

print(important\_proteins.head(10))

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijving

**1.2 Random Forest WITHOUT PCA (Accuracy 0.90)**

# Importing Required Libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

import shap

# Load the Data

mergeddata\_filepath = '../input/merged-data/merged\_data.csv'

mergeddata = pd.read\_csv(mergeddata\_filepath)

# Select Only Proteins Starting From 'O43704'

protein\_data = mergeddata.loc[:, 'O43704':]

# Define the Target (Tumor Subtype)

target = mergeddata['Proteomic\_Subtype']

# Split the Data into Training and Test Sets (70% Train, 30% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(protein\_data, target, test\_size=0.3, random\_state=1)

# Convert Scaled Data to DataFrame with Correct Column Names

X\_test\_scaled = pd.DataFrame(X\_test, columns=X\_test.columns)

# Train a Random Forest Classifier

rf\_model = RandomForestClassifier(random\_state=1)

rf\_model.fit(X\_train, y\_train)

# Cross-Validation of Random Forest (5-Fold Cross-Validation)

cv\_scores = cross\_val\_score(rf\_model, X\_train, y\_train, cv=5)

print(f"Random Forest Cross-validation Scores: {cv\_scores}")

print(f"Mean Cross-validation Accuracy: {cv\_scores.mean()}")

# Make Predictions

y\_pred = rf\_model.predict(X\_test)

# Evaluate the Model

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print the Evaluation Results

print(f"Accuracy: {accuracy}")

print(f"Classification Report:\n{classification\_rep}")

# Plot the Confusion Matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=rf\_model.classes\_)

disp.plot(cmap='Blues')

plt.show()

# Feature Importance

importances = rf\_model.feature\_importances\_

# Create a DataFrame of Important Proteins

important\_proteins = pd.DataFrame({

'Protein': protein\_data.columns,

'Importance': importances

}).sort\_values(by='Importance', ascending=False)

# Show the Top 10 Most Important Proteins

print("\nTop 10 Most Important Proteins:")

print(important\_proteins.head(10))

# Importing additional libraries for plotting

import matplotlib.pyplot as plt

import seaborn as sns

# Select the Top 10 Most Important Proteins

top\_10\_proteins = important\_proteins.head(10)

# Plot the Top 10 Most Important Proteins

plt.figure(figsize=(12, 8))

sns.barplot(

x='Importance',

y='Protein',

data=top\_10\_proteins,

palette='coolwarm',

edgecolor='black'

)

# Add labels and a title to the plot

plt.title('Top 10 Most Important Proteins in Random Forest Model', fontsize=16, pad=15)

plt.xlabel('Importance Score', fontsize=12)

plt.ylabel('Protein', fontsize=12)

# Add gridlines for better readability

plt.grid(axis='x', linestyle='--', alpha=0.6)

# Show the plot

plt.show()

Afbeelding met tekst, schermopname, Parallel, lijn

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijving

## LOGISTIC REGRESSION

* 1. **Logistic regression WITHOUT PCA (0.94 Accuracy**

**# Importing required libraries**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split, cross\_val\_score**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay**

**import matplotlib.pyplot as plt**

**import shap**

**# Load the dataset**

**mergeddata\_filepath = '../input/merged-data/merged\_data.csv'**

**merged\_data = pd.read\_csv(mergeddata\_filepath)**

**# Select only proteins starting from 'O43704'**

**protein\_data = merged\_data.loc[:, 'O43704':]**

**# Target (tumor subtype)**

**target = merged\_data['Proteomic\_Subtype']**

**# Split the data into train and test sets (70% train, 30% test)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(protein\_data, target, test\_size=0.3, random\_state=42)**

**# Convert scaled data back to DataFrames**

**X\_train = pd.DataFrame(X\_train, columns=protein\_data.columns, index=X\_train.index)**

**X\_test = pd.DataFrame(X\_test, columns=protein\_data.columns, index=X\_test.index)**

**# Train a Logistic Regression Model**

**log\_model = LogisticRegression(max\_iter=1000, random\_state=42)**

**# Perform cross-validation to evaluate the model's performance**

**cv\_scores = cross\_val\_score(log\_model, X\_train, y\_train, cv=5, scoring='accuracy')**

**print(f"Logistic Regression Cross-validation Scores: {cv\_scores}")**

**print(f"Mean Cross-validation Accuracy: {cv\_scores.mean():.2f}")**

**# Train the model**

**log\_model.fit(X\_train, y\_train)**

**# Make predictions**

**y\_pred = log\_model.predict(X\_test)**

**# Evaluate the model**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**classification\_rep = classification\_report(y\_test, y\_pred)**

**conf\_matrix = confusion\_matrix(y\_test, y\_pred)**

**# Print the evaluation results**

**print(f"Accuracy: {accuracy:.2f}")**

**print(f"\nClassification Report:\n{classification\_rep}")**

**# Plot the Confusion Matrix**

**disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=log\_model.classes\_)**

**disp.plot(cmap='Blues')**

**plt.title("Confusion Matrix - Logistic Regression")**

**plt.show()**

**# View the important proteins (feature importance)**

**coefficients = pd.DataFrame({**

**'Protein': protein\_data.columns,**

**'Coefficient': log\_model.coef\_[0]**

**}).sort\_values(by='Coefficient', ascending=False)**

**# Show the top 10 most important proteins (positive influence)**

**print("\nTop 10 Most Important Proteins (Positive Influence):")**

**print(coefficients.head(10))**

**# Show the 10 proteins with the strongest negative influence**

**print("\nTop 10 Most Important Proteins (Negative Influence):")**

**print(coefficients.tail(10))**

**# View the model parameters**

**model\_parameters = pd.DataFrame({**

**'attribute': X\_train.columns,**

**'parameter\_value': log\_model.coef\_[0]**

**})**

**# Sort by absolute value**

**model\_parameters['parameter\_abs\_value'] = model\_parameters['parameter\_value'].abs()**

**model\_parameters = model\_parameters.sort\_values(by="parameter\_abs\_value", ascending=False)**

**# Select the most important attributes**

**selected\_attributes = list(model\_parameters.iloc[:10]['attribute'])**

**print(f"Selected Attributes: {selected\_attributes}")**

**# Fit the model again with selected attributes**

**cls\_std = LogisticRegression(max\_iter=1000, random\_state=42)**

**cls\_std.fit(X\_train[selected\_attributes], y\_train)**

**print(f"Accuracy: {cls\_std.score(X\_train[selected\_attributes], y\_train):.2f}")**

**# Search for optimal attribute selection**

**for s in range(2, 50):**

**top\_s\_attributes = model\_parameters['attribute'][:s]**

**X\_train\_selected = X\_train[top\_s\_attributes]**

**X\_test\_selected = X\_test[top\_s\_attributes]**

**cls\_selected = LogisticRegression(max\_iter=1000, random\_state=42)**

**cls\_selected.fit(X\_train\_selected, y\_train)**

**accuracy = cls\_selected.score(X\_test\_selected, y\_test)**

**print(f"Accuracy for top {s} proteins: {accuracy:.2f}")**

**# Importing additional libraries for plotting**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**# Combine positive and negative coefficients**

**top\_positive = coefficients.head(10)**

**top\_negative = coefficients.tail(10)**

**# Combine data for visualization**

**top\_features = pd.concat([top\_negative, top\_positive])**

**# Reset the index of the combined DataFrame**

**top\_features = top\_features.reset\_index(drop=True)**

**# Create a bar plot**

**plt.figure(figsize=(14, 8))**

**sns.barplot(**

**x='Coefficient', y='Protein', data=top\_features,**

**palette='coolwarm', edgecolor='black'**

**)**

**# Add labels and title**

**plt.title('Most Important Proteins in Logistic Regression Model', fontsize=16, pad=15)**

**plt.xlabel('Coefficient Value (Effect Size)', fontsize=12)**

**plt.ylabel('Proteins', fontsize=12)**

**plt.grid(axis='x', linestyle='--', alpha=0.6)**

**# Show the plot**

**plt.show()**

Afbeelding met tekst, schermopname, software, scherm

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, Parallel, lijn

Automatisch gegenereerde beschrijving

* 1. **Logistic Regression WITH PCA (Same Accuracy of 0.94)**

# Importing Required Libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

import shap

# Load the Dataset

mergeddata\_filepath = '../input/merged-data/merged\_data.csv'

merged\_data = pd.read\_csv(mergeddata\_filepath)

# Select Only Proteins Starting From 'O43704'

protein\_data = merged\_data.loc[:, 'O43704':]

# Define the Target (Tumor Subtype)

target = merged\_data['Proteomic\_Subtype']

# Split the Data into Training and Test Sets (70% Train, 30% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(protein\_data, target, test\_size=0.3, random\_state=1)

# Apply PCA to Reduce Dimensionality (Keeping 95% of Variance)

pca = PCA(n\_components=0.95) # Keep 95% of the variance

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

# Train a Logistic Regression Model

log\_model = LogisticRegression(max\_iter=1000, random\_state=1)

# Perform Cross-Validation to Evaluate the Model's Performance

cv\_scores = cross\_val\_score(log\_model, X\_train\_pca, y\_train, cv=5, scoring='accuracy')

print(f"Logistic Regression Cross-validation Scores: {cv\_scores}")

print(f"Mean Cross-validation Accuracy: {cv\_scores.mean():.2f}")

# Train the Model

log\_model.fit(X\_train\_pca, y\_train)

# Make Predictions

y\_pred = log\_model.predict(X\_test\_pca)

# Evaluate the Model

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print the Evaluation Results

print(f"Accuracy: {accuracy:.2f}")

print(f"\nClassification Report:\n{classification\_rep}")

# Plot the Confusion Matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=log\_model.classes\_)

disp.plot(cmap='Blues')

plt.title("Confusion Matrix - Logistic Regression")

plt.show()

# Feature Importance

coefficients = pd.DataFrame({

'Protein': [f'PC{i+1}' for i in range(X\_train\_pca.shape[1])], # Use the components instead of protein names

'Coefficient': log\_model.coef\_[0]

}).sort\_values(by='Coefficient', ascending=False)

# Show the Top 10 Most Important Components (Positive Influence)

print("\nTop 10 Most Important Components (Positive Influence):")

print(coefficients.head(10))

# Show the 10 Most Negatively Influential Components

print("\nTop 10 Most Important Components (Negative Influence):")

print(coefficients.tail(10))

# View the Model Parameters

model\_parameters = pd.DataFrame({

'attribute': [f'PC{i+1}' for i in range(X\_train\_pca.shape[1])],

'parameter\_value': log\_model.coef\_[0]

})

# Sort by Absolute Value

model\_parameters['parameter\_abs\_value'] = model\_parameters['parameter\_value'].abs()

model\_parameters = model\_parameters.sort\_values(by="parameter\_abs\_value", ascending=False)

# Select the Most Important Attributes

selected\_attributes = list(model\_parameters.iloc[:10]['attribute'])

print(f"Selected Attributes: {selected\_attributes}")

# Fit the Model Again with Selected Attributes

cls\_std = LogisticRegression(max\_iter=1000, random\_state=42)

cls\_std.fit(X\_train\_pca[:, :10], y\_train) # Train again with top 10 components

print(f"Accuracy: {cls\_std.score(X\_train\_pca[:, :10], y\_train):.2f}")

# Search for Optimal Attribute Selection

for s in range(2, X\_train\_pca.shape[1]):

top\_s\_attributes = model\_parameters['attribute'][:s]

X\_train\_selected = X\_train\_pca[:, :s]

X\_test\_selected = X\_test\_pca[:, :s]

cls\_selected = LogisticRegression(max\_iter=1000, random\_state=42)

cls\_selected.fit(X\_train\_selected, y\_train)

accuracy = cls\_selected.score(X\_test\_selected, y\_test)

print(f"Accuracy for top {s} components: {accuracy:.2f}")

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijving

1. **tSNE analyse**

# Perform t-SNE on the training data

print("\nPerforming t-SNE on the training data...")

from sklearn.manifold import TSNE

import seaborn as sns

tsne = TSNE(n\_components=2, random\_state=1, perplexity=30)

X\_train\_tsne = tsne.fit\_transform(X\_train)

# Convert the t-SNE results into a DataFrame

tsne\_df = pd.DataFrame(X\_train\_tsne, columns=['Component 1', 'Component 2'])

tsne\_df['Subtype'] = y\_train.values

# Plot the t-SNE results

plt.figure(figsize=(10, 7))

sns.scatterplot(data=tsne\_df, x='Component 1', y='Component 2', hue='Subtype', palette='Set1', alpha=0.7)

plt.title("t-SNE Visualization of Training Data")

plt.xlabel("Component 1")

plt.ylabel("Component 2")

plt.legend(title='Subtype')

plt.show()

Afbeelding met schermopname, tekst, diagram

Automatisch gegenereerde beschrijving

This t-SNE shows that the protein profiles of all subgroups are relatively consistent (especially S1 and S3 are consistent). However, it is noticeable that the protein profile of S2 is more varied, so the identification of S2 subgroups based on protein abundances will be less accurate compared to S1 and S3.