Student ID: 11164605
DATA70132

Supervised & Unsupervised Classification Analysis of The Vertebral Dataset

OVERVIEW

Vertebral dataset is a biomedical dataset built by Dr. Henrique da Mota, designed for a classification task to predict either Normal or Abnormal condition, with the latter covering Disk Hernia and Spondylolisthesis conditions. This paper details the application of both supervised and unsupervised classification to the dataset.

UNSUPERVISED CLASSIFICATION

The objective of the unsupervised classification task is to identify clusters of given data points based on their similarities and assign each data point to its respective cluster. Some common techniques include centroid-based, hierarchical, and model-based clustering.

This paper focuses on model-based clustering, specifically Expectation-Maximization, which assumes data comes from different distributions and uses soft assignments, allowing for overlapping cluster members, as is the case in this dataset. However, its effectiveness can be limited by initialisation conditions, possibly leading to local rather than global optima.

After the initialisation, the algorithm proceeds through two steps: Expectation and Maximization. The Expectation step is described by Rogers & Girolami (2016):

$$z_{ij} = \frac{\pi_j \, f(x_i \mid \mu_j, \sum_j)}{\sum_{r=1}^{k} \pi_r \, f(x_i \mid \mu_r, \sum_r)}$$

While the Maximisation steps computes the parameters given the posterior z_{ij}.

$$\pi_j = \frac{1}{n} \sum_{i=1}^{n} z_{ij} \qquad \mu_j = \frac{\sum_{i=1}^{n} z_{ij} \, x_i}{\sum_{i=1}^{n} z_{ij}} \qquad \sum_j = \frac{\sum_{i=1}^{n} z_{ij} \, (x_i - \mu_j)^T \, (x_i - \mu_j)}{\sum_{i=1}^{n} z_{ij}}$$

where

- z_{ij} : cluster assignment probabilities of data point-*i* in cluster-*j*
- x_i : data point-*i*
- π_j : mixture weights of cluster-*j*
- μ_j : means of cluster-*j*
- $\sum_j$: covariance of cluster-*j*
- k : number of clusters

SUPERVISED CLASSIFICATION

In contrast to unsupervised, supervised classification operates with labelled data, learning from the relationship between labels and features to predict accurate labels for unseen data. One of the algorithms is SVM (Support Vector Machine), which classifies data by identifying a hyperplane that separates the classes while maximising the margin between them.

The margin is the distance between the hyperplane and the nearest points from each class, which are called support vectors. The objective function and the given constraints are given as follows (Rogers & Girolami, 2016).

$$\underset{w}{\text{argmin}} \quad \frac{1}{2}\|w\|^2$$
$$\text{subject to} \quad t_n(w^T x_n + b) \geq 1, \text{ for all } n$$

where

- w : perpendicular vector to the hyperplane that separates the different classes
- t_n : label of the nth training example
- x_n : nth data point
- b : bias

To enable the soft margin, which allows some data points to be incorrectly classified, the corresponding equation becomes:

$$\underset{w}{\text{argmin}} \quad \frac{1}{2}w^T w + C\sum_{n=1}^{N}\xi_n$$
$$\text{subject to } \xi_n \geq 0 \quad t_n(w^T x_n + b) \geq 1 - \xi_n, \text{ for all } n$$

The slack variable, ξ_n, is added to allow a certain degree of misclassification, and the regularisation parameter *C* controls the trade-off between maximizing the margin and minimising misclassification. Finally, an unseen data point will be labelled according to this equation.

$$t_{new} = sign(w^T x_{new} + b)$$

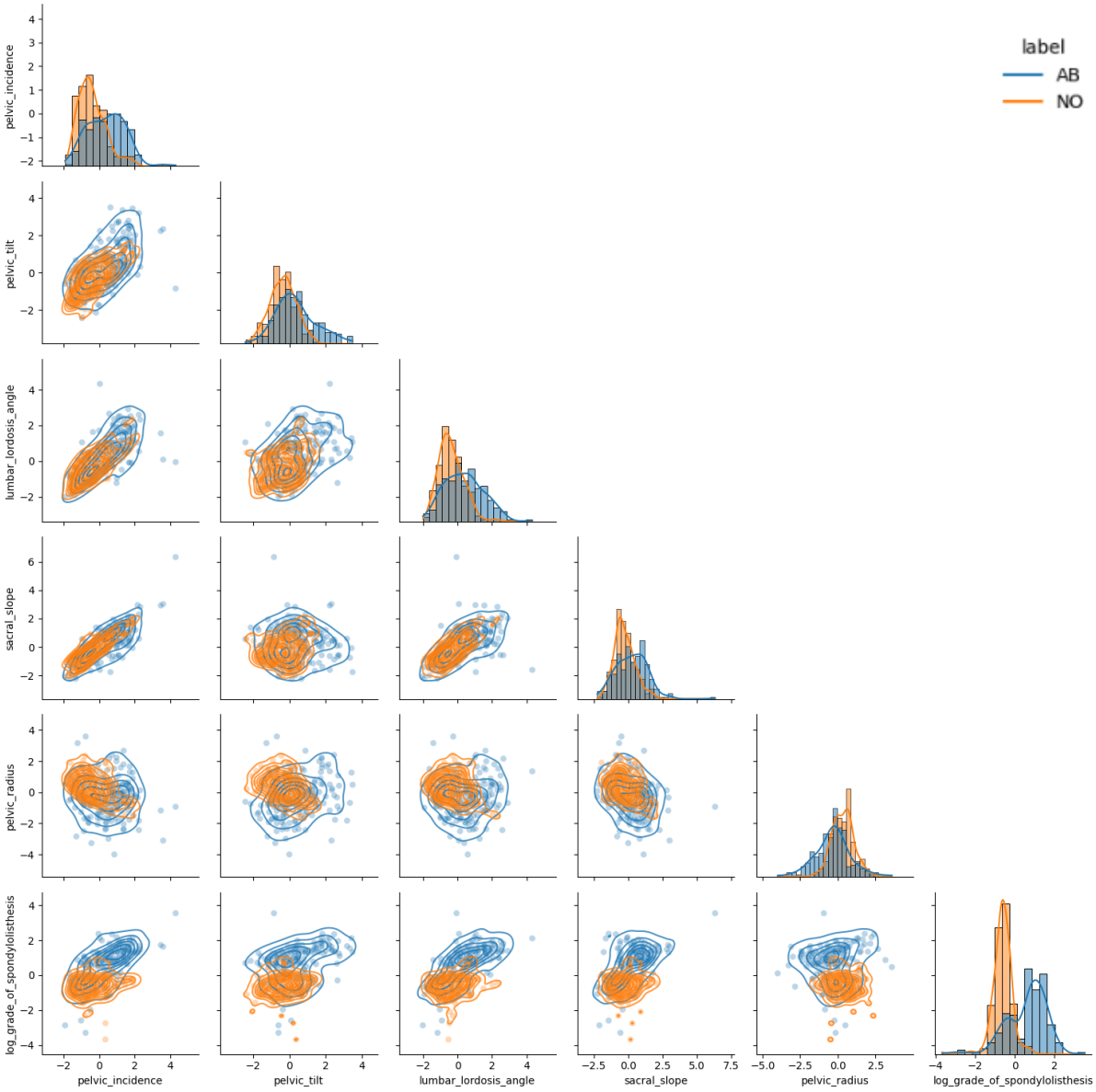
The SVM is also capable of handling non-linear classification problems by applying the Kernel trick, such as Polynomial and Radial Basis Function. Nevertheless, incorrect choices might lead to underfitting or overfitting. Moreover, SVM is sensitive to imbalanced and the scale of the data.

EXPLORATORY DATA ANALYSIS & PRE-PROCESSING

The dataset consists of 210 abnormal and 100 normal classes, creating an imbalance. Since SVM and GMM could be affected by this imbalance, the SMOTE (Synthetic Minority Over-sampling Technique) method (Chawla et al., 2002) was applied to balance the Normal class. SMOTE generates synthetic data that is similar to existing samples rather than duplicating them, which reduces the risk of overfitting.

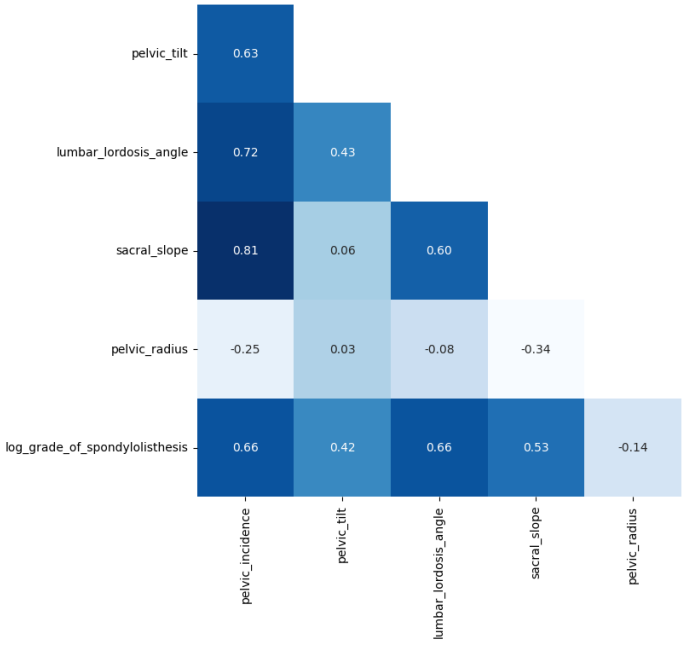
In addition, the *grade_of_spondylolisthesis* has been log transformed to reduce the skewness. Following this, the features were scaled by standard scaler. The interaction between features, coloured by the original label, is shown in Figure 1.

Figure 1: Multivariate EDA (After Transformation & Pre-Processing)



The figure is more effective at identifying the clusters than before the transformation (*see Appendix 1*). However, some of the features such as *pelvic_tilt* and *sacral_slope* may still not be useful due to their predominantly overlapped clusters. Conversely, *log_grade_of_spondylolisthesis* seems to distinguish the clusters sufficiently well, which is intuitive considering that one of the abnormal conditions is spondylolisthesis. Furthermore, the correlation matrix in Figure 2 suggests potential multicollinearity among variables, possibly distorting the classification.

Figure 2: Correlation Matrix



The feature selection was applied by running every possible combination of features (*see Appendix 2*). The best subset of features includes *pelvic_incidence*, *lumbar_lordosis_angle*, *pelvic_radius*, and *log_grade_of_spondylolisthesis*.

RESULT & ANALYSIS

▪ Unsupervised

Figure 1 suggests that the clusters are irregular in shape and mostly overlapping. Therefore, one suitable clustering algorithm could be the model-based, Expectation-Maximisation algorithm. EM requires pre-defined number of clusters, which can be determined through the elbow plot. Additionally, a dendrogram was added to provide more information on cluster formation, helping in the identification of natural clusters.

Figure 3: Elbow Plot

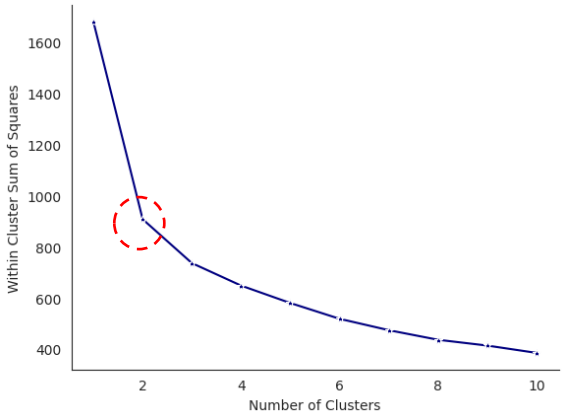
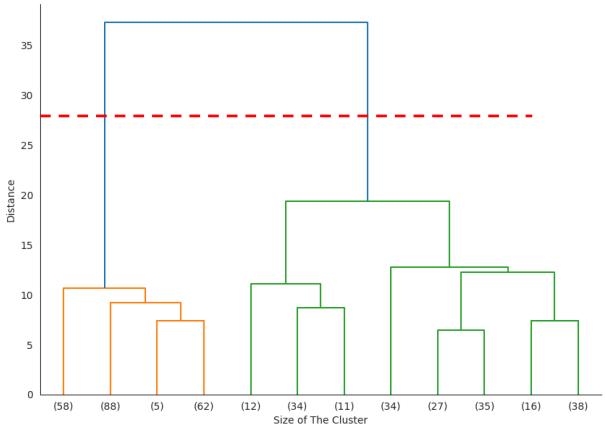


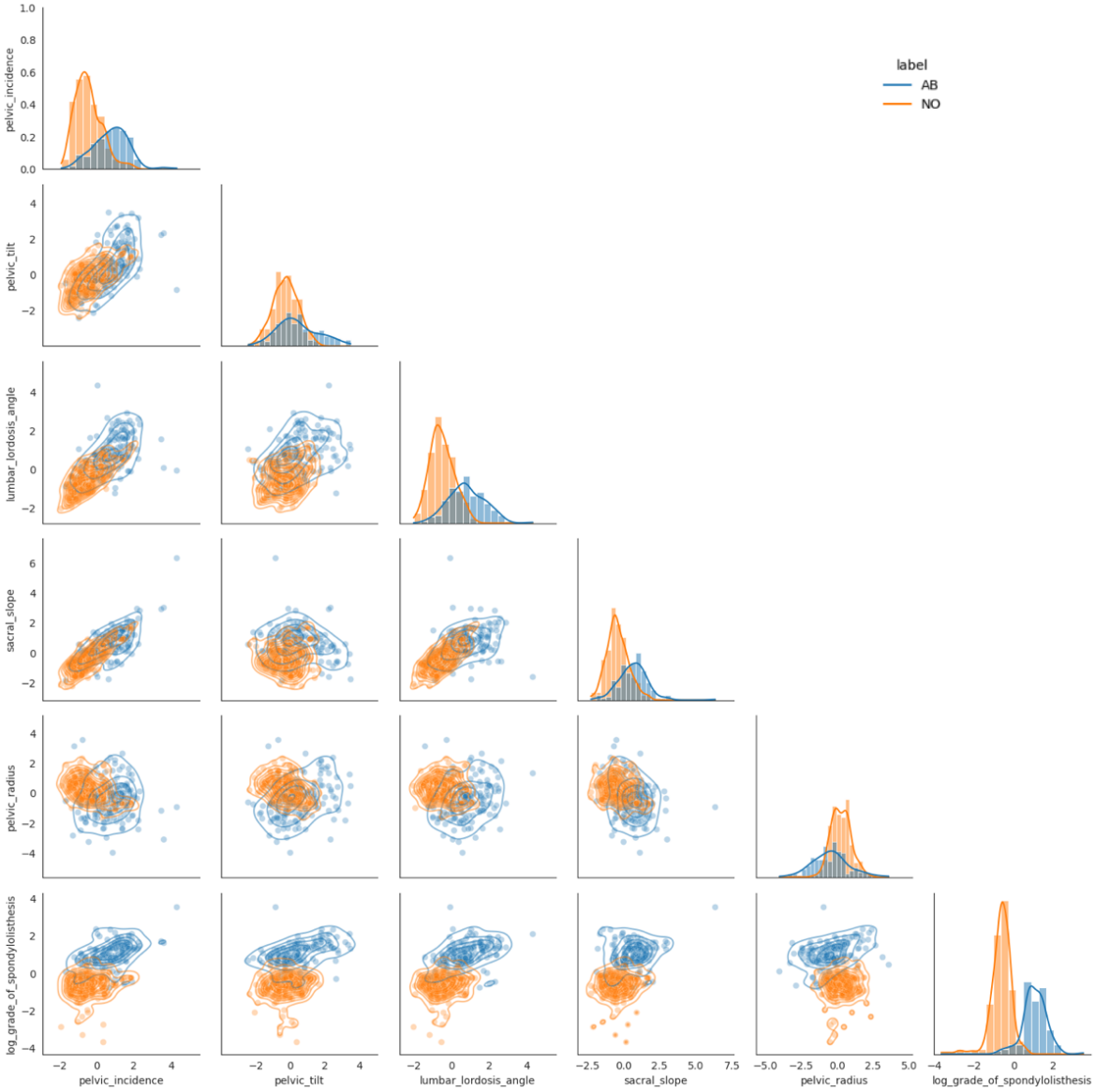
Figure 4: Dendrogram



Both plots indicate an optimal two clusters in the dataset, based on the elbow plot's slowdown in the Within-Cluster Sum of Squares and the dendrogram's longest vertical gap. Furthermore, the dendrogram suggests that three clusters might also exist, seemingly corresponding to Normal, Spondylolisthesis, and Disk Hernia classes. However, the analysis will focus on two clusters as they are more distinguishable.

The EM algorithm, specifically the Gaussian Mixture Model, was applied using Python's scikit-learn default settings, which initialises parameters with K-Means to prevent local optima. The result is shown in Figure 5.

Figure 5: Clustering Result



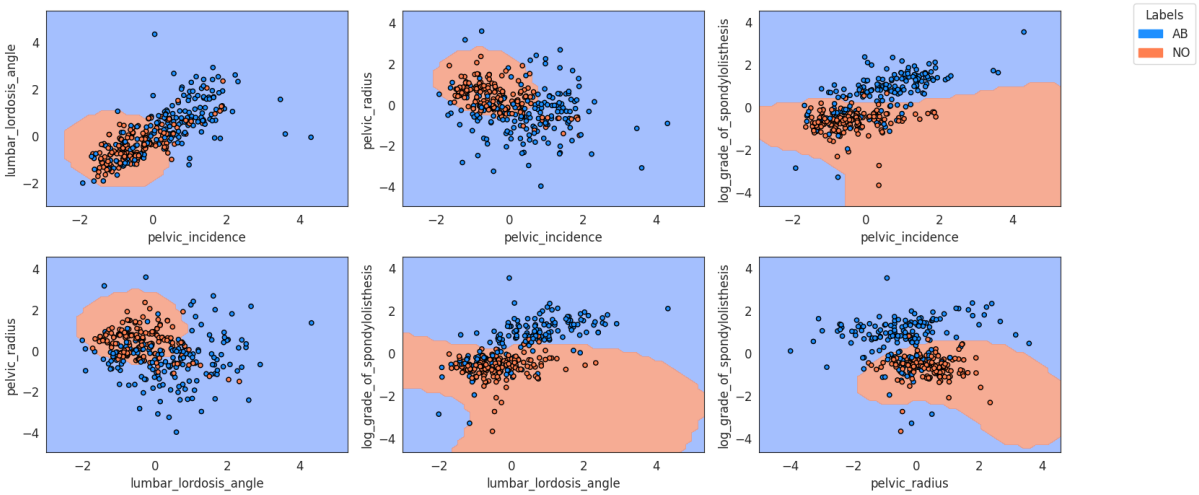
The clusters seem similar to the original cluster. However, many abnormal classes were incorrectly classified as normal, especially in overlapping areas observed in *log_grade_spondylolisthesis*. This could be caused by the outliers appeared in Normal class, shifting the optimum covariance of Normal label to be higher. Although cleaning the outliers might help, it may require the judgment of a domain expert.

Furthermore, defining the metric to evaluate the clustering is important. Additional experiments showed that the silhouette score or distance-based metric might not be suitable due to the many overlapping members between clusters (*see Appendix 3*).

- **Supervised**

Regarding supervised classification, SVM outperformed Random Forest and Naïve Bayes (*see Appendix 4*). To assess generalisation on unseen data, the dataset was split into training and testing set. Additionally, hyperparameter tuning, incorporating cross validation, was also performed. The best hyperparameter combination found includes radial basis function (RBF) kernel with gamma γ 0.1 and regularisation C 10. The results of the training set classification are shown in Figure 6.

Figure 6: Classification on Training Set



The figure shows the non-linear clusters identified by the RBF kernel. It can be observed that the interactions with *log_grade_of_spondylolisthesis* might be the best for classifying the class. The predictions on the test set are shown by the confusion matrix in Table 1, with detailed scores provided in Table 2.

Table 1: Confusion Matrix

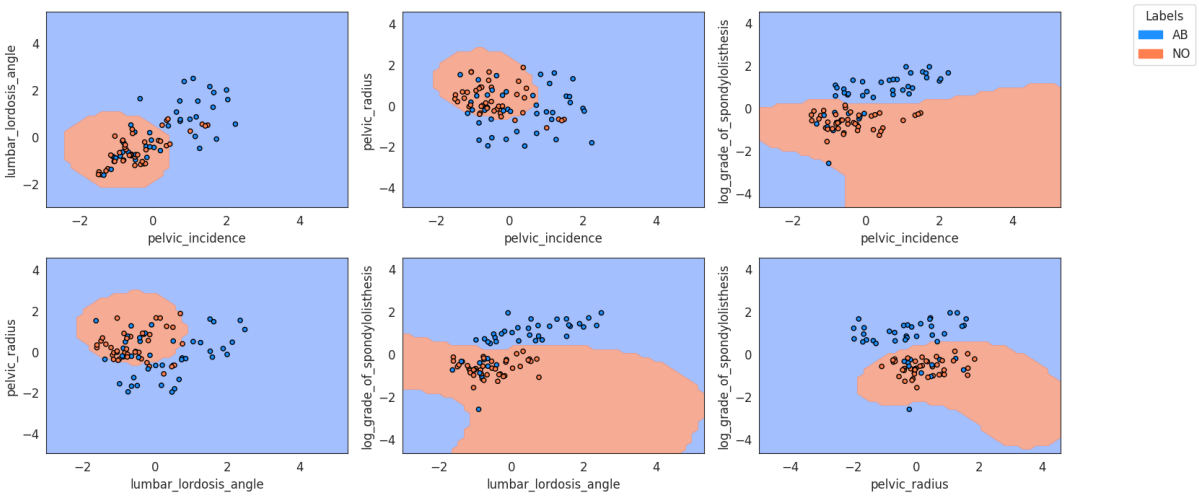
	Predicted AB	Predicted NO
True: AB	33	9
True: NO	0	42

Table 2: Supervised Classification Scores

Metrics	Score
Precision	0.91
Recall	0.89
F1	0.89

Moreover, Figure 7 displays the data points of the test set against the decision boundaries of the SVM.

Figure 7: Classification on Test Set



The prediction appears to yield good results, as judged by the score and visualisation, although some classes are still misclassified. Improvements could be made in data pre-processing, such as outlier removal, to enable the SVM to construct a more optimal hyperplane. Moreover, the margin might also suggest a potential overfitting. Addressing this issue could involve adding more data and tuning the regularisation hyperparameter.

In general, unsupervised learning aids in understanding data distributions and can assist in creating labelled data for semi-supervised classification and imputing missing values for features. In this experiment, supervised learning was utilised for feature selection, which helped to construct better clusters by reducing noise from irrelevant features. Given that labels are available, the performance of supervised models can also serve as a benchmark to enhance clustering algorithms.

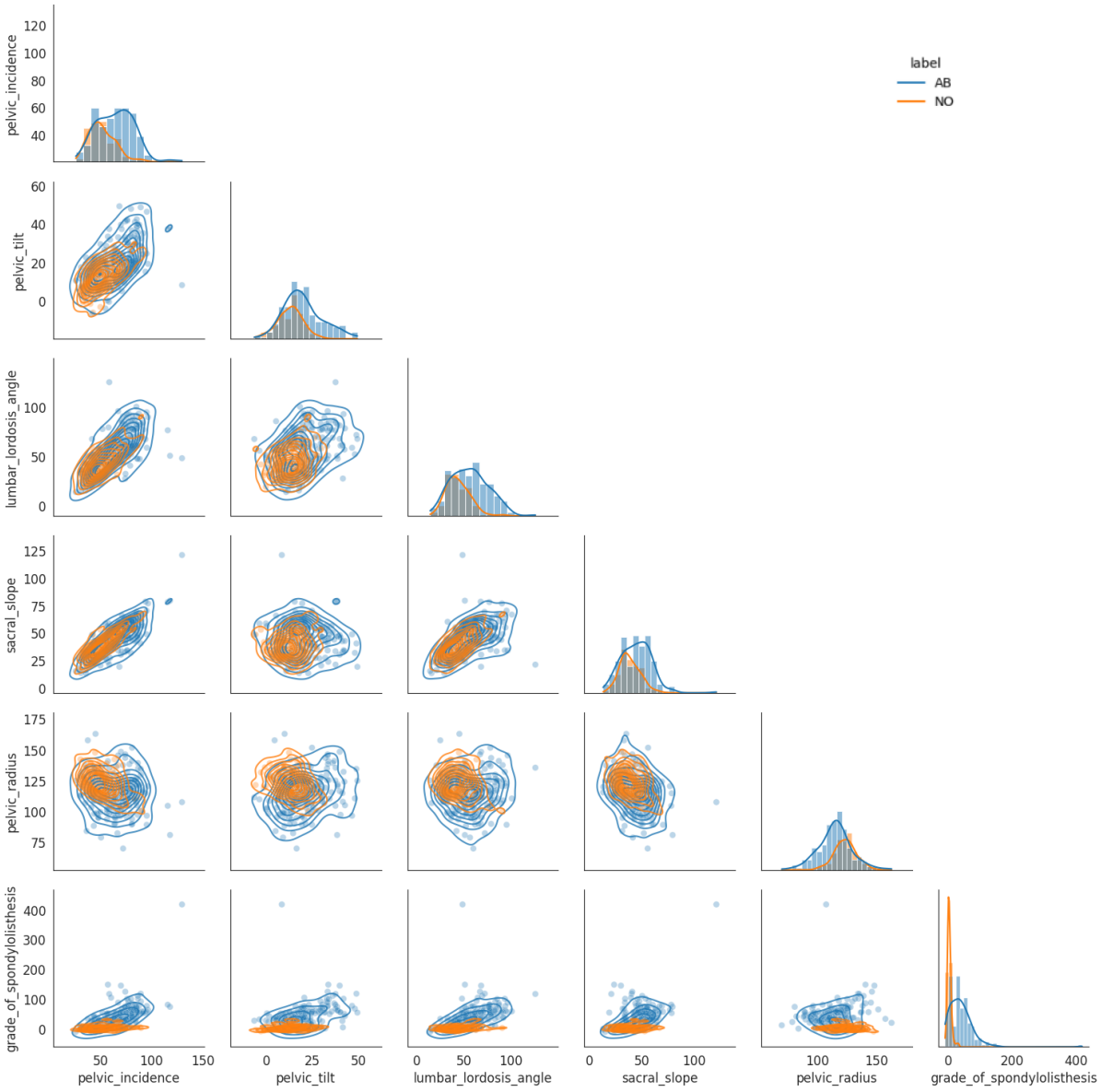
REFERENCES

Rogers, S. & Girolami, M., 2016. A First Course in Machine Learning. 2nd ed. Boca Raton: Chapman & Hall/CRC. ISBN: 9781498738484.

Chawla, N.V., Bowyer, K.W., Hall, L.O. & Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, pp.321-357.

APPENDIX

Appendix 1: Multivariate EDA (Before Transformation & Pre-Processing)



Appendix 2: Feature Selection Results Sorted by F1 Score (Top 5 and Bottom 5)

Feature Combination	F1 Score	Silhoutte Score
pelvic_incidence,lumbar_lordosis_angle,pelvic_radius,log_grade_of_spondylolisthesis	0.876	0.422
pelvic_incidence,pelvic_tilt,pelvic_radius,log_grade_of_spondylolisthesis	0.871	0.362
pelvic_incidence,pelvic_tilt,lumbar_lordosis_angle,pelvic_radius,log_grade_of_spondylolisthesis	0.871	0.388
pelvic_tilt,lumbar_lordosis_angle,pelvic_radius,log_grade_of_spondylolisthesis	0.857	0.368
pelvic_incidence,lumbar_lordosis_angle,sacral_slope,pelvic_radius,log_grade_of_spondylolisthesis	0.857	0.416
...	...	...
pelvic_incidence,pelvic_tilt,sacral_slope,log_grade_of_spondylolisthesis	0.814	0.427
pelvic_tilt,lumbar_lordosis_angle,sacral_slope,log_grade_of_spondylolisthesis	0.813	0.409
pelvic_tilt,lumbar_lordosis_angle,log_grade_of_spondylolisthesis	0.807	0.437
pelvic_tilt,pelvic_radius,log_grade_of_spondylolisthesis	0.807	0.346
pelvic_incidence,pelvic_tilt,lumbar_lordosis_angle,pelvic_radius	0.807	0.378

The selected features were decided based on the F1 Score since the Silhouette Score might not be suitable for this dataset.

Appendix 3: Clustering Algorithm Comparison

Algorithms	Silhouette Score	ARI*	F1
K-Means	0.432	0.273	0.725
EM (GMM)	0.387	0.469	0.820
Hierarchical	0.363	0.288	0.767

*ARI: Adjusted Rand Index, a metric used to measure the similarity between two clustering assignments, including the true labels and the labels predicted by a clustering algorithm.

The Silhouette Score might not be the appropriate metric for this dataset due to the presence of many overlapping cluster members. Despite exhibiting a poor Silhouette Score, the EM algorithm demonstrated the best performance in terms of ARI and F1 Score, two metrics for comparison with the true labels.

Appendix 4: Supervised Classification Comparison on Test Data (Before Hyperparameters Tuning)

Algorithms	Precision	Recall	F1-Score
SVM	0.87	0.86	0.86
RF	0.84	0.83	0.83
NB	0.81	0.80	0.80

Appendix 5: Hyperparameters Option

Hyperparameters	Options
C	[0.1, 1, 10, 100]
gamma	[1, 0.1, 0.01, 0.001]
kernel	['rbf', 'poly']

Appendix 6: Python Code

1. Set up environments and data

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from itertools import combinations
from collections import Counter
from scipy.stats import mode
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.kernel_approximation import RBFSampler
from sklearn.mixture import GaussianMixture
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder, PolynomialFeatures
from sklearn.metrics import classification_report, make_scorer, f1_score, confusion_matrix,
silhouette_score, ConfusionMatrixDisplay, adjusted_rand_score
from matplotlib.patches import Patch
```

```
#load dataset
column_names = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope',
'pelvic_radius', 'grade_of_spondylolisthesis', 'label']
vertebral = pd.read_csv('vertebral_column_data.txt', delimiter=' ', header=None,
names=column_names)
```

2. Data Pre-Processing

```
#transform grade_of_spondylolisthesis to log_grade_of_spondylolisthesis

#add offset to avoid 0 for log_grade_of_spondylolisthesis
offset = np.abs(np.min(vertebral['grade_of_spondylolisthesis'])) + 1
vertebral_df = vertebral.copy()
#add new column log_grade_of_spondylolisthesis
vertebral_df['log_grade_of_spondylolisthesis'] =
np.round(np.log(vertebral['grade_of_spondylolisthesis'] + offset),2)
#drop grade_of_spondylolisthesis
vertebral_df = vertebral_df.drop('grade_of_spondylolisthesis', axis=1)
```

```
y = vertebral_df['label']
X = vertebral_df.drop('label', axis=1)
```

```
#oversampling dataset using SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
```

```
#scaling dataset
numeric_cols = X_res.columns.tolist() #add feature names
scaler = StandardScaler() #initialise scaler
X_scaled = scaler.fit_transform(X_res) #scaling
X_scaled = pd.DataFrame(X_scaled, columns = numeric_cols) #convert to data frame
scaled_vertebral = pd.concat([X_scaled, y_res], axis=1) #combine with label
```

```
#feature selection by F1-Score

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_res, test_size=0.3,
random_state=42)

#set up best score and feature combination storage
max_score = 0
best_combination = None
all_scores = []

#evaluate all possible combinations of features
for L in range(1, len(X_scaled.columns) + 1):
    for subset in itertools.combinations(X_scaled.columns, L):
        #select only the columns for the current features combination
        X_subset_train = X_train[list(subset)]
        X_subset_test = X_test[list(subset)]

        #train the SVM classifier
        classifier = SVC(kernel='rbf')
        classifier.fit(X_subset_train, y_train)

        #make predictions and evaluate using F1 score
        y_pred = classifier.predict(X_subset_test)
        score = f1_score(y_test, y_pred, pos_label='AB')

        #save the score and combination
        all_scores.append((subset, score))

        #update the best score and combination if score is higher
        if score > max_score:
            max_score = score
            best_combination = subset

#print all feature combinations with F1 scores
for combination, score in all_scores:
    print(f'Feature combination: {combination}, F1 score: {score}')

#print the best feature combination and F1 score
print(f'Best feature combination: {best_combination}, Best F1 score: {max_score}')
```

```
#feature selection (Silhouette Score)


#set up max score and feature combination storage
max_score = -1
best_combination = None
all_scores = []

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_res, test_size=0.3,
random_state=42)

#evaluate all possible combinations of features
for L in range(2, len(X_scaled.columns) + 1):  #start from 2 since silhouette requires
multiple clusters to compute
    for subset in itertools.combinations(X_scaled.columns, L):
        ##select only the columns for the current features combination
        X_subset = X_train[list(subset)]

        #initialise kmeans
        kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)
        cluster_labels = kmeans.fit_predict(X_subset)

        #compute the silhouette score
        score = silhouette_score(X_subset, cluster_labels)

        #Save the score and features combination
        all_scores.append((subset, score))

        #update the best score and combination if score is higher
        if score > max_score:
            max_score = score
            best_combination = subset

#print all feature combinations with silhouette scores
for combination, score in all_scores:
    print(f'Feature combination: {combination}, Silhouette score: {score}')

#print the best feature combination and silhouette score
print(f'Best feature combination: {best_combination}, Best Silhouette score: {max_score}')
```

3. Exploratory Data Analysis

```
#multivariate EDA
#initialise the pairgrid
grid = sns.PairGrid(scaled_vertebral, hue='label')

#draw kde and scatter plot to the lower diagonal
grid.map_lower(sns.kdeplot, alpha=0.8)
grid.map_lower(sns.scatterplot, alpha=0.3)

#draw histogram to the diagonal
grid.map_diag(sns.histplot, kde=True)

#add legend
grid.add_legend(title='label', bbox_to_anchor=(1, 1), loc='upper left')

#remove upper diagonal plot
for i, j in zip(*np.tril_indices_from(grid.axes, -1)):
    grid.axes[j, i].set_visible(False)

plt.show()
```

```
#draw correlation matrix
#calculate correlation
corr = vertebral_df.corr(numeric_only=True)

#generate a mask to hide the upper diagonal
mask = np.triu(np.ones_like(corr, dtype=bool))

#initialise matplotlib figure
fig, ax = plt.subplots(figsize=(11, 9))
```

```
#draw the heatmap
sns.heatmap(corr, mask=mask, cmap='Blues', cbar_kws={"shrink": .5}, annot=True, fmt='.2f')
```

4. Unsupervised Classification

- K-Means

```
#draw elbow plot

max_k = 10 #set maximum number of clusters to be checked
sum_of_squares = [] #initialise empty list
#run k-means iteratively for each k number of clusters
for k in range(1,max_k+1):
    kmeans = KMeans(n_clusters = k, n_init = 10) #initialise K-Means for each k
    kmeans.fit(X_subset) #fit to scaled data
    sum_of_squares.append(kmeans.inertia_) #store the within cluster sum of squares in the
list

#plotting the elbow plot
sns.set_style('white')
sns.lineplot(x=list(range(1,max_k+1)), y=sum_of_squares,  marker = '*', color='navy')
plt.xlabel('Number of Clusters')
plt.ylabel('Within Cluster Sum of Squares')
sns.despine() #remove the top and right border line
plt.show()
```

```
#kmeans with 2 clusters
kmeans = KMeans(n_clusters=2, n_init=4)
kmeans.fit(X_subset)
kmeans_labels = kmeans.labels_
```

```
#calculate Silhoutte Score
score_silhoutte = silhouette_score(X_subset, kmeans_labels)
print('K-Means Silhouette Score:', score_silhoutte)

#calculate ARI
ari_score = adjusted_rand_score(y_res, kmeans_labels)
print('K-Means ARI:', ari_score)

#calculate F1
true_label = np.array(y_res)
labels_con = np.where(kmeans_labels == 1, 'AB', 'NO')
f1 = f1_score(true_label, labels_con, pos_label='AB')
print('K-Means F1:', f1)
```

- Hierarchical

```
#draw dendrogram

#generate the linkage matrix
#use ward to minimise within cluster sum of squares
Z = linkage(X_subset, 'ward')

#plot the dendrogram
plt.figure(figsize=(10, 7))
plt.ylabel('Distance')
plt.xlabel('Size of The Cluster')

#create dendrogram
dendrogram(
    Z,
    truncate_mode='lastp',  #truncate to display only the last p merged clusters
    p=12,  #show only the last 12 merged clusters
    leaf_font_size=10,  #font size for the x axis labels
)
sns.despine() #remove spine
plt.show()
```

```
#Agglomerative Clustering with 2 clusters
agg = AgglomerativeClustering(n_clusters=2, metric='euclidean', linkage='ward')
agg_labels = agg.fit_predict(X_scaled)
```

```
#calculate Silhoutte Score
score_silhoutte = silhouette_score(X_subset, agg_labels)
print('Hierarchical Silhouette Score:', score_silhoutte)
```

```
#calculate ARI
ari_score = adjusted_rand_score(y_res, agg_labels)
print('Hierarchical ARI:', ari_score)

#calculate F1
true_label = np.array(y_res)
labels_con = np.where(agg_labels == 0, 'AB', 'NO')
f1 = f1_score(true_label, labels_con, pos_label='AB')
print('Hierarchical F1:', f1)
```

- **Expectation-Maximisation**

```
#initialise the Gaussian Mixture Model
gmm = GaussianMixture(n_components=2, random_state=0)

#fit the model and make prediction for the cluster
gmm.fit(X_subset)
gmm_labels = gmm.predict(X_subset)
```

```
#calculate Silhoutte Score
score_silhoutte = silhouette_score(X_subset, gmm_labels)
print('EM Silhouette Score:', score_silhoutte)

#calculate ARI
ari_score = adjusted_rand_score(y_res, gmm_labels)
print('EM ARI:', ari_score)

#calculate F1
true_label = np.array(y_res)
labels_con = np.where(gmm_labels == 0, 'AB', 'NO')
f1 = f1_score(true_label, labels_con, pos_label='AB')
print('EM F1:', f1)
```

- **Plot Clusters Result**

```
#store cluster results
cluster_results = scaled_vertebral.copy()
cluster_results['cluster'] = gmm_labels
```

```
#initialise the pairgrid
grid = sns.PairGrid(cluster_results, hue='cluster')

#draw kde and scatter plot to the lower diagonal
grid.map_lower(sns.kdeplot, alpha=0.8)
grid.map_lower(sns.scatterplot, alpha=0.3)

#draw histogram to the diagonal
grid.map_diag(sns.histplot, kde=True)

#add legend
grid.add_legend(title='label', bbox_to_anchor=(1, 1), loc='upper left')

#remove upper diagonal plot
for i, j in zip(*np.tril_indices_from(grid.axes, -1)):
    grid.axes[j, i].set_visible(False)

plt.show()
```

5. Supervised Classification

```
#splitting train and test data
X_train, X_test, y_train, y_test = train_test_split(X_subset, y_res, test_size=0.2,
stratify=y_res, random_state=42)
```

- **Random Forest**

```
#initialise random forest classifier
rf = RandomForestClassifier()
#fit training data
rf.fit(X_train, y_train)
```

```
#making prediction on training set
y_pred_trainrf = rf.predict(X_train)
print(classification_report(y_train, y_pred_trainrf))
```

```
#making prediction on test set
```

```
y_pred_testrf = rf.predict(X_test)
print(classification_report(y_test, y_pred_testrf))
```

- **Naïve Bayes**

```
#initialise naive bayes
nb = GaussianNB()
#fit training data
nb.fit(X_train, y_train)

#making prediction for training set
y_pred_trainnb = nb.predict(X_train)
print(classification_report(y_train, y_pred_trainnb))

#making prediction for test set
y_pred_testnb = nb.predict(X_test)
print(classification_report(y_test, y_pred_testnb))
```

- **SVM**

```
#initialise SVM classifier
svm = SVC(kernel='linear', probability=True)
#fit the training data
svm.fit(X_train, y_train)

#making prediction on training set
y_pred_train = svm.predict(X_train)
print(classification_report(y_train, y_pred_train))

#making prediction on test set
y_pred_test = svm.predict(X_test)
print(classification_report(y_test, y_pred_test))
```

- **Hyperparameters Tuning**

```
#initialise SVM classifier
svm = SVC(probability=True)

#hyperparameters option
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001],
              'kernel': ['rbf', 'poly']}

#set custom f1 score for grid search
f1_scorer = make_scorer(f1_score, pos_label='AB')
```

```
#initialise SVM classifier
svm = SVC(probability=True)

#hyperparameters option
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001],
              'kernel': ['rbf', 'poly']}

#set custom f1 score for grid search
f1_scorer = make_scorer(f1_score, pos_label='AB')

#fit the training data
gs.fit(X_train, y_train)

#print best parameters
print("Best parameters found: ", gs.best_params_)

#fit the best model & make prediction on training set
best_model = gs.best_estimator_
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_train)
print(classification_report(y_train, y_pred))


#make prediction on test set
y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))

#compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
#use ConfusionMatrixDisplay to plot
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Abnormal', 'Normal'])

#plot the confusion matrix
plt.figure(figsize=(3,3))
disp.plot(cmap='Blues')

plt.title('Confusion Matrix')
plt.show()
```

- **Draw Results**

```
#draw results for training data

#encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_train)

#get unique labels from the training data
unique_labels = np.unique(y_train)
#set label colors
label_colors = {label: color for label, color in zip(unique_labels, ['dodgerblue', 'coral'])}

#plotting for each feature combinations
feature_pairs = combinations(X_subset.columns, 2)
plt.figure(figsize=(15, 7))

for i, (feature1, feature2) in enumerate(feature_pairs, start=1):
    #get the specific pair of features for the current plot
    idx1, idx2 = X_train.columns.get_loc(feature1), X_train.columns.get_loc(feature2)
    X_pair = X_train[[feature1, feature2]]

    # SVM model
    model = SVC(kernel='rbf', gamma=0.1, C=10)
    model.fit(X_pair, y_encoded)

    #plot decision boundaries
    plt.subplot(2, 3, i)
    x_min, x_max = X_pair.iloc[:, 0].min() - 1, X_pair.iloc[:, 0].max() + 1
    y_min, y_max = X_pair.iloc[:, 1].min() - 1, X_pair.iloc[:, 1].max() + 1

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 50),
                         np.linspace(y_min, y_max, 50))
    grid_points = pd.DataFrame(np.c_[xx.ravel(), yy.ravel()], columns=[feature1, feature2])
    Z = model.predict(grid_points)
    Z = Z.reshape(xx.shape)

    #plot the contour
    plt.contourf(xx, np.array(yy), Z, alpha=0.8, levels=[-1, 0, 1], cmap= 'coolwarm' )
    #plot scatter for training data
    plt.scatter(X_pair[feature1], X_pair[feature2], c=[label_colors[label] for label in
y_train], s=20, edgecolors='k')

    plt.xlabel(feature1)
    plt.ylabel(feature2)

#add legend to the plot outside of the subplots
legend= [Patch(color=color, label=label) for label, color in label_colors.items()]

plt.tight_layout()
plt.show()
```

```
#draw results for test data

#encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_train)

#get unique labels from the training data
unique_labels = np.unique(y_test)
#set label colors
label_colors = {label: color for label, color in zip(unique_labels, ['dodgerblue', 'coral'])}

#plotting for each feature combinations
feature_pairs = combinations(X_subset.columns, 2)
```

```
plt.figure(figsize=(15, 7))

for i, (feature1, feature2) in enumerate(feature_pairs, start=1):
    #get the specific pair of features for the current plot
    idx1, idx2 = X_train.columns.get_loc(feature1), X_train.columns.get_loc(feature2)
    X_pair = X_train[[feature1, feature2]]
    X_pair_test = X_test[[feature1, feature2]]

    #initialise SVM model
    model = SVC(kernel='rbf', gamma=0.1, C=10)
    model.fit(X_pair, y_encoded)

    #plot decision boundaries
    plt.subplot(2, 3, i)
    x_min, x_max = X_pair.iloc[:, 0].min() - 1, X_pair.iloc[:, 0].max() + 1
    y_min, y_max = X_pair.iloc[:, 1].min() - 1, X_pair.iloc[:, 1].max() + 1

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 50),
                         np.linspace(y_min, y_max, 50))
    grid_points = pd.DataFrame(np.c_[xx.ravel(), yy.ravel()], columns=[feature1, feature2])
    Z = model.predict(grid_points)
    Z = Z.reshape(xx.shape)

    #plot the contour
    plt.contourf(xx, np.array(yy), Z, alpha=0.8, levels=[-1, 0, 1], cmap= 'coolwarm' )
    #plot scatter for test data
    plt.scatter(X_pair_test[feature1], X_pair_test[feature2], c=[label_colors[label] for label
in y_test], s=20, edgecolors='k')

    plt.xlabel(feature1)
    plt.ylabel(feature2)

# Create legend handles manually
legend_handles = [Patch(color=color, label=label) for label, color in label_colors.items()]

#add legend to the plot outside of the subplots
plt.figlegend(handles=legend_handles, title="Labels", bbox_to_anchor=(1.05, 1), loc='upper
left')

plt.tight_layout()
plt.show()
```