AE2 COM725

MEHRDAD MADADI

Student number :102607756

Breast Cancer Wisconsin

SOLENT UNIVERSITY SOUTHAMPTON

Apr 2025

Introduction

AI algorithms can analyze medical images (e.g., X-rays, MRIs, ultrasounds, CT scans, and DXAs) and assist healthcare providers in identifying and diagnosing diseases more accurately and quickly (Khan & Islam, 2023). One key area of application is the analysis of microscopic images. Nuclear segmentation in digital microscopic tissue images can enable extraction of high-quality features for nuclear morphometrics and other analysis in computational pathology (Kumar et al., 2017). Studies such as Radhakrishnan et al. (2017) demonstrated that machine learning can be used for cancer diagnosis through nuclear biomarkers. Accurate prediction of cancer can play a crucial role in its treatment. The procedure of cancer detection is incumbent upon the doctor, which at times can be subjected to human error and therefore leading to erroneous decisions, using machine learning techniques for the same can prove to be beneficial. (Sharma, Kulshrestha & Daniel, 2018). For this study, I used the Breast Cancer Wisconsin dataset, originally introduced by Street, Wolberg, and Mangasarian in 1993, which has since become a benchmark in cancer classification research. Numerous studies have applied different methods to this dataset, yielding varying results. For instance, Sharma et al. used Support Vector Machines (SVM) and Artificial Neural Networks (ANN) for classification of this dataset and showed better results using SVM. In contrast, Ćwiklińska-Jurkowska (2013) explored resampling techniques, such as bagging and boosting, to improve model accuracy. Wolberg, Street, and Mangasarian (1995) employed 10-fold cross-validation for model evaluation, while in 2019, Sidey-Gibbons and Sidey-Gibbons experimented with regularized General Linear Models, SVM, and ANN to assess classification performance.

One observation I made about this dataset is that it is slightly imbalanced. Studies have shown that imbalanced datasets can lead to poor model performance, as models may become biased toward the majority class, resulting in lower accuracy for the minority class (e.g., He & Garcia, 2009). To address this, I plan to employ two techniques: XGBoost, which has a built-in method for handling imbalanced datasets, as demonstrated by Chen et al. (2016), and resampling (oversampling) for the minority class, a technique that has been shown to improve performance in binary classification tasks (e.g., Chawla et al., 2002). I will compare the performance of these two approaches and use cross-validation to ensure the results are reliable, as cross-validation is widely recognized for its ability to reduce overfitting and provide a more robust estimate of model performance (e.g., Kohavi, 1995). Additionally, I will apply feature selection using XGBoost's feature importance, a technique that has been shown to effectively identify the most relevant predictors and improve model interpretability (e.g., Chen & Guestrin, 2016).

Methods

Dataset

The dataset was obtained from the UCI Machine Learning Repository, a well-known and reliable source for open-access datasets. The repository not only provides the dataset itself but also includes an introductory paper and numerous other studies that have cited the dataset. Additionally, it offers valuable insights into the data, its features, and the problem at hand, as well as the results achieved by other researchers using various machine learning models.

The dataset is titled "Breast Cancer Wisconsin (Diagnostic)" and was introduced by Street, Wolberg, and Mangasarian in 1993. It contains 30 features, with 10 real-valued attributes computed for the cell nuclei of each patient, the standard error and worst value for each of these columns. The dataset consists of 569 rows and includes a target column, "Diagnosis," where the values are labelled as "M" for malignant and "B" for benign, as well as a column containing the patient ID numbers.

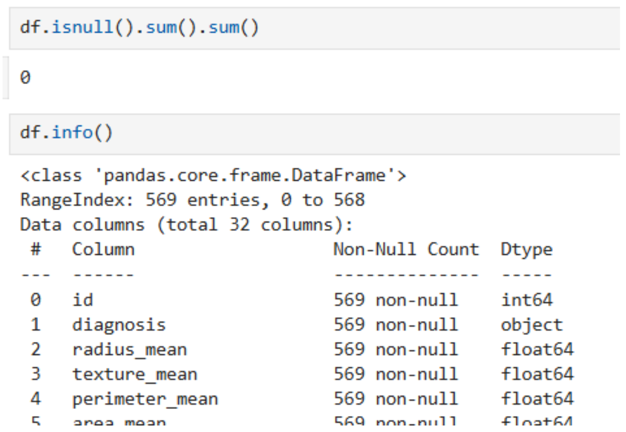
Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. The 10 measurements are as follows:

1. Radius (mean of distances from center to points on the perimeter)
2. Texture (standard deviation of gray-scale values)
3. Perimeter
4. Area
5. Smoothness (local variation in radius lengths)
6. Compactness (perimeter^2 / area - 1.0)
7. Concavity (severity of concave portions of the contour)
8. Concave points (number of concave portions of the contour)
9. Symmetry
10. Fractal dimension ("coastline approximation" - 1)

Analysis and Results

Dataset Preparation

The dataset was mostly clean, with no missing values or duplicates, and the data types were correct. I performed checks for missing values, duplicates, and ensured the data types were appropriate. The only modification I made was to create a new column by converting the "Diagnosis" column, replacing the labels "malignant" and "benign" with 1 and 0, respectively, and to drop the "ID" column, as it was not relevant to the analysis. These changes ensure that the data is properly formatted for feeding into machine learning models, where numeric labels and relevant features are required for classification tasks.

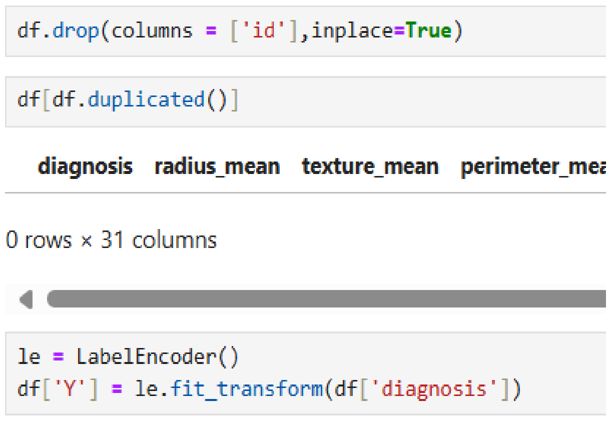


```
df.isnull().sum().sum()
```

```
0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
```



```
df.drop(columns = ['id'],inplace=True)
```

```
df[df.duplicated()]
```

diagnosis	radius_mean	texture_mean	perimeter_mea

0 rows × 31 columns

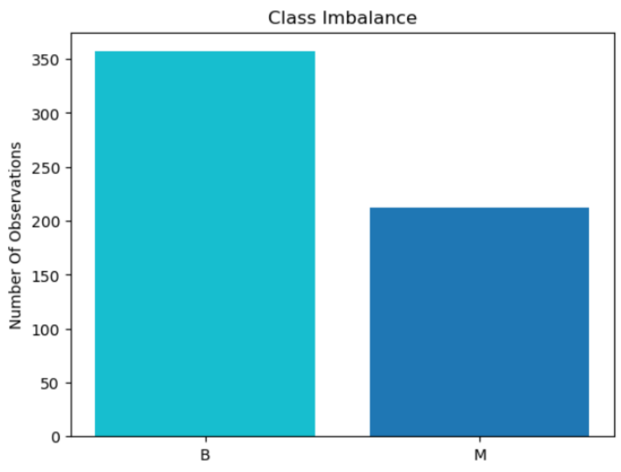
```
le = LabelEncoder()
df['Y'] = le.fit_transform(df['diagnosis'])
```

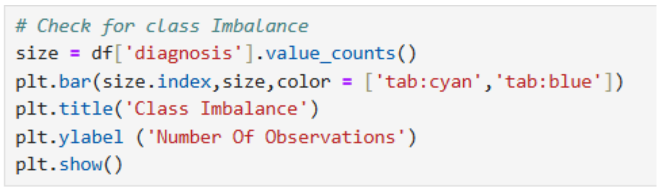
Figure 1 Data cleaning

Exploratory Data Analysis

1. Class Imbalance Detection

To assess the class imbalance, I first counted the instances of each target group. Using a bar chart from Matplotlib, I visualized the distribution of the target variable. The dataset contains 357 benign cases, making up 63% of the data, while 37% of the data consists of malignant cases (shown in Figure 2).





```
# Check for class Imbalance
size = df['diagnosis'].value_counts()
plt.bar(size.index,size,color = ['tab:cyan','tab:blue'])
plt.title('Class Imbalance')
plt.ylabel ('Number Of Observations')
plt.show()
```

Figure 2 Class imbalanced bar chart and code

2. Feature Distribution Across Diagnoses

Since this is a binary classification problem, I wanted to compare the distribution of each feature between the two diagnostic groups (benign and malignant). The boxplots help us see where the two groups differ, identify outliers, and understand the overall distribution of the features. This gives a quick visual overview of how the features relate to the diagnosis and highlights any key differences or similarities.

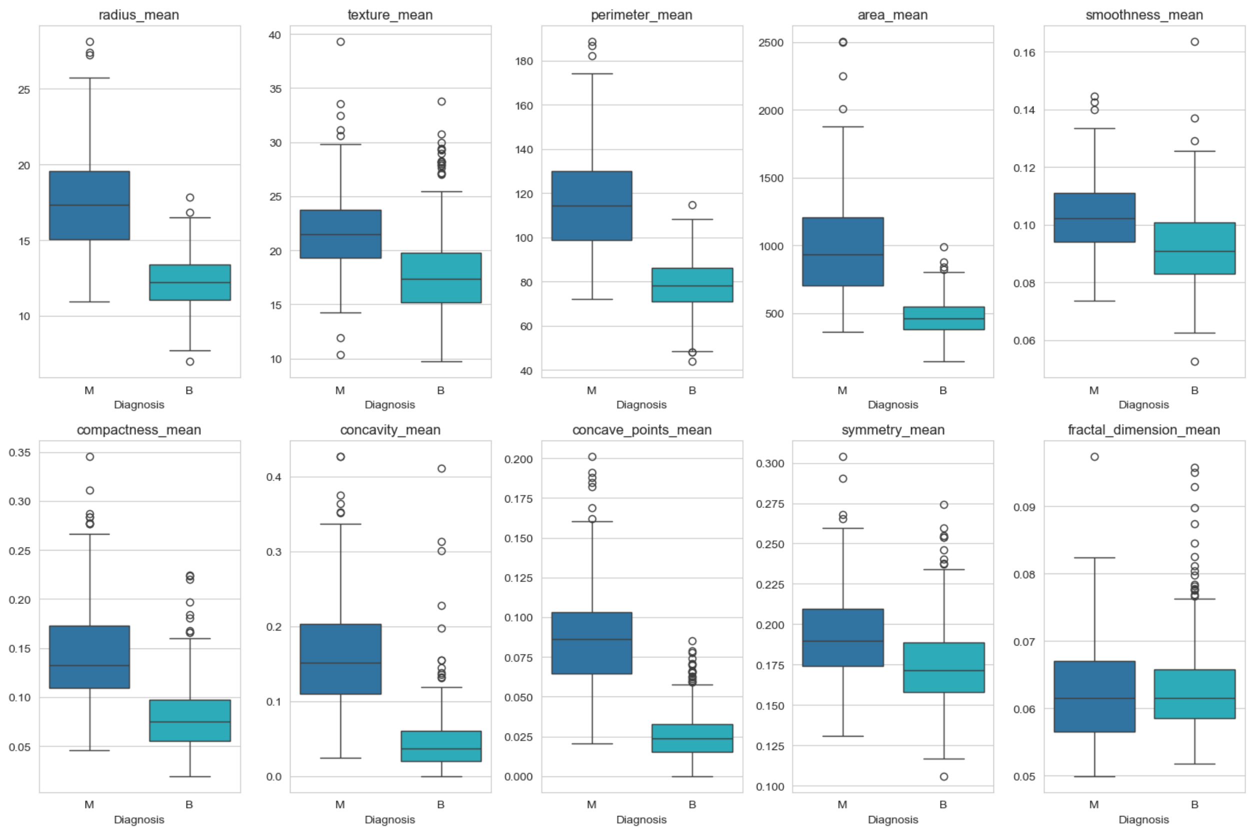
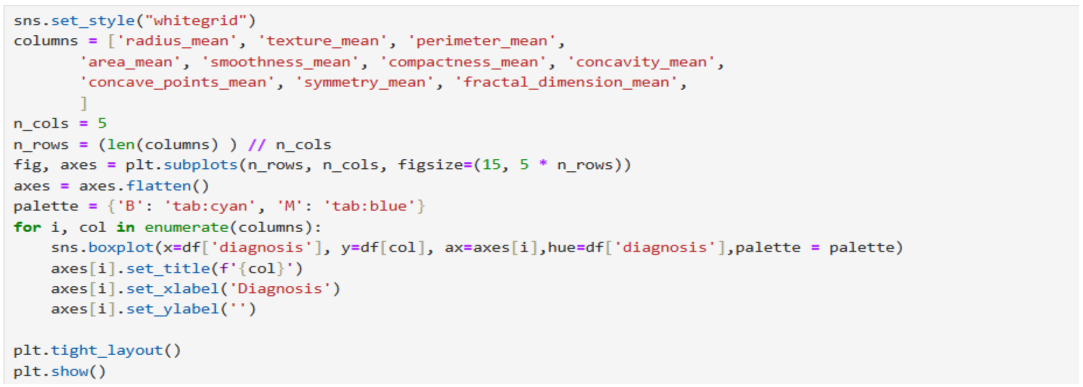


Figure 3 Boxplot across diagnoses

I created this plot using the Seaborn library in Python, applying a white grid style for clarity. To maintain consistency with the previous plot, I generated ten subplots using the same color scheme with box plots.



```python
sns.set_style("whitegrid")
columns = ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       ]
n_cols = 5
n_rows = (len(columns) ) // n_cols
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))
axes = axes.flatten()
palette = {'B': 'tab:cyan', 'M': 'tab:blue'}
for i, col in enumerate(columns):
    sns.boxplot(x=df['diagnosis'], y=df[col], ax=axes[i],hue=df['diagnosis'],palette = palette)
    axes[i].set_title(f'{col}')
    axes[i].set_xlabel('Diagnosis')
    axes[i].set_ylabel('')

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

Figure 4 Code for box plot

3. Feature Distributions with KDE Overlay

Here, I visualize the distribution of each feature using histograms and KDE curves. This helps us check for skewness, kurtosis, and overall shape of the distributions. By overlaying the KDE, we can better understand the density and smoothness of each feature's distribution.

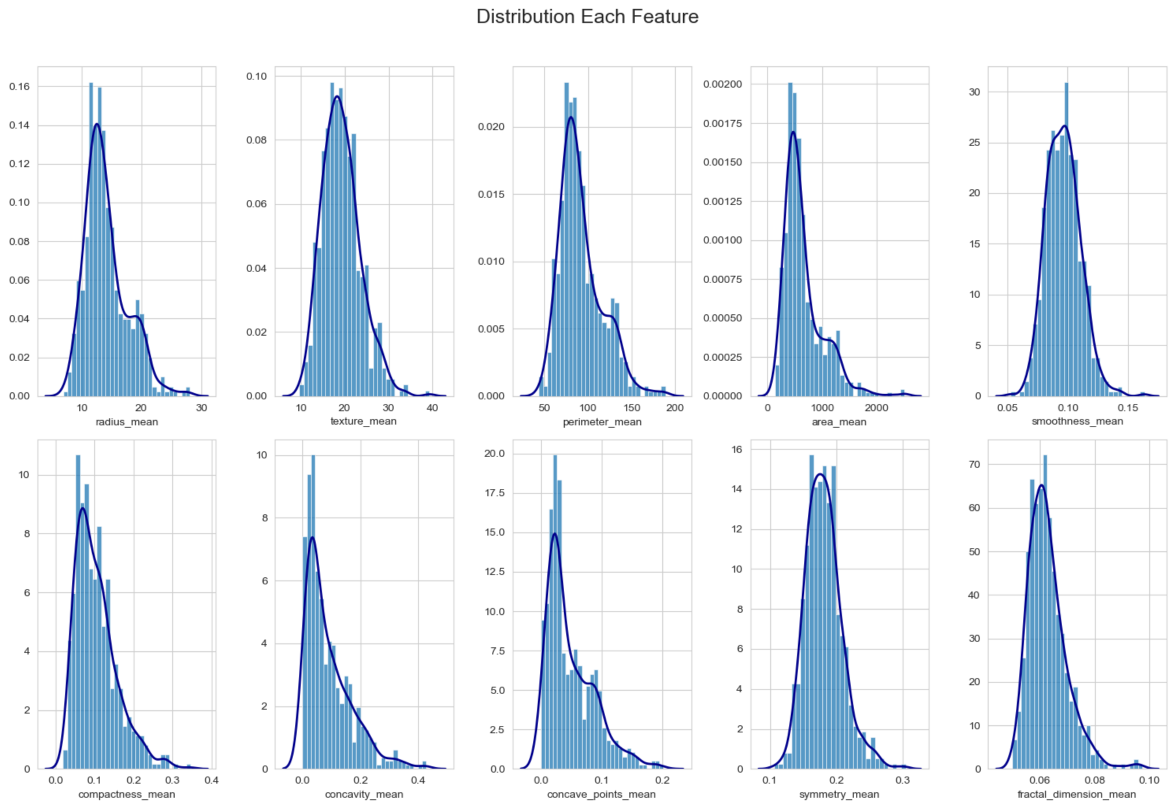
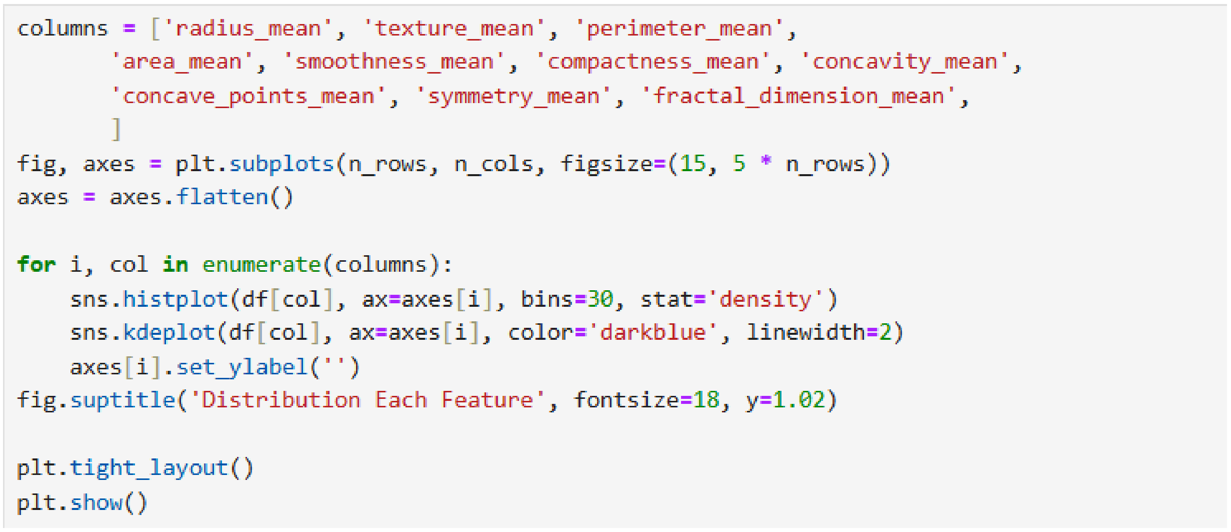


Figure 5 Histogram and KDE curve

I created this plot using the Seaborn library in Python like the previous one, applying a white grid style for clarity. I generated ten subplots using seaborn histogram plots and added KDE curve and titles.

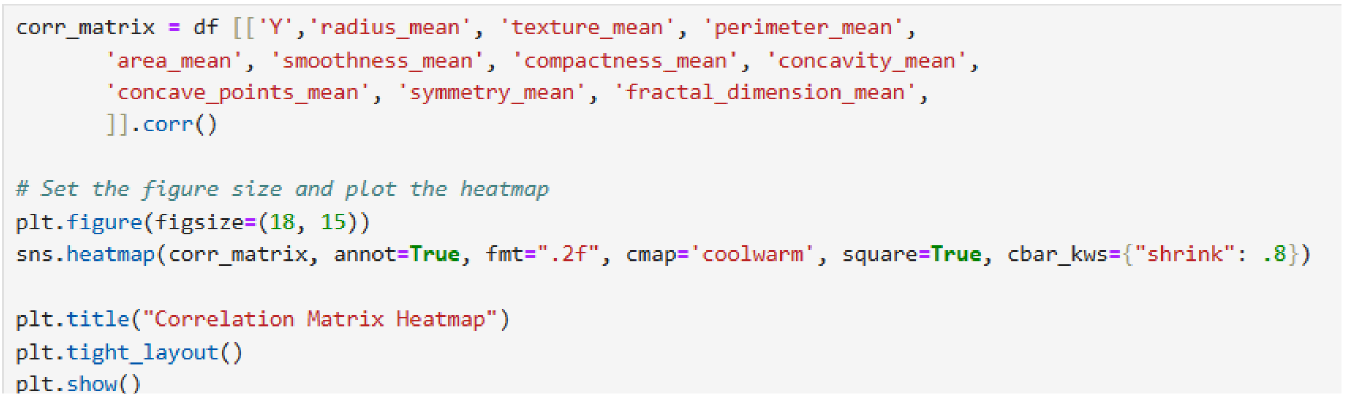


```
columns = ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       ]
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))
axes = axes.flatten()

for i, col in enumerate(columns):
    sns.histplot(df[col], ax=axes[i], bins=30, stat='density')
    sns.kdeplot(df[col], ax=axes[i], color='darkblue', linewidth=2)
    axes[i].set_ylabel('')
fig.suptitle('Distribution Each Feature', fontsize=18, y=1.02)

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

Figure 6 Code for histogram and KDE

4. Correlation Matrix Heatmap

I used a correlation matrix to calculate the pairwise correlations between the 10 features and the target variable. Then, I visualized these correlations using a heatmap with seaborn. This allows us to easily spot positive and negative relationships between the features and the target, helping us to understand how each feature is related to one another and to the diagnosis.



```
corr_matrix = df [['Y','radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       ]].corr()

# Set the figure size and plot the heatmap
plt.figure(figsize=(18, 15))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, cbar_kws={"shrink": .8})

plt.title("Correlation Matrix Heatmap")
plt.tight_layout()
plt.show()
```

Figure 7 Code for heat map

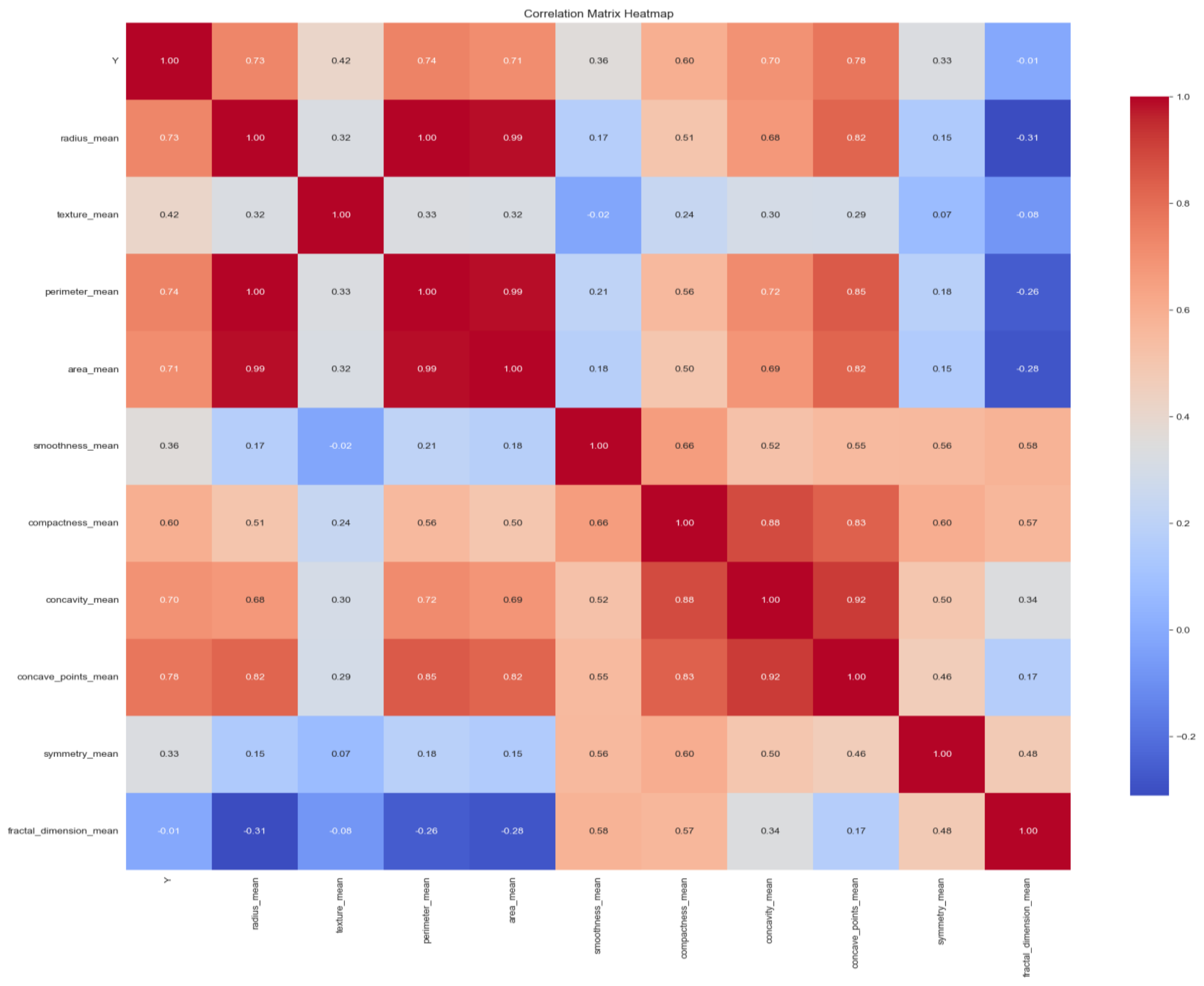
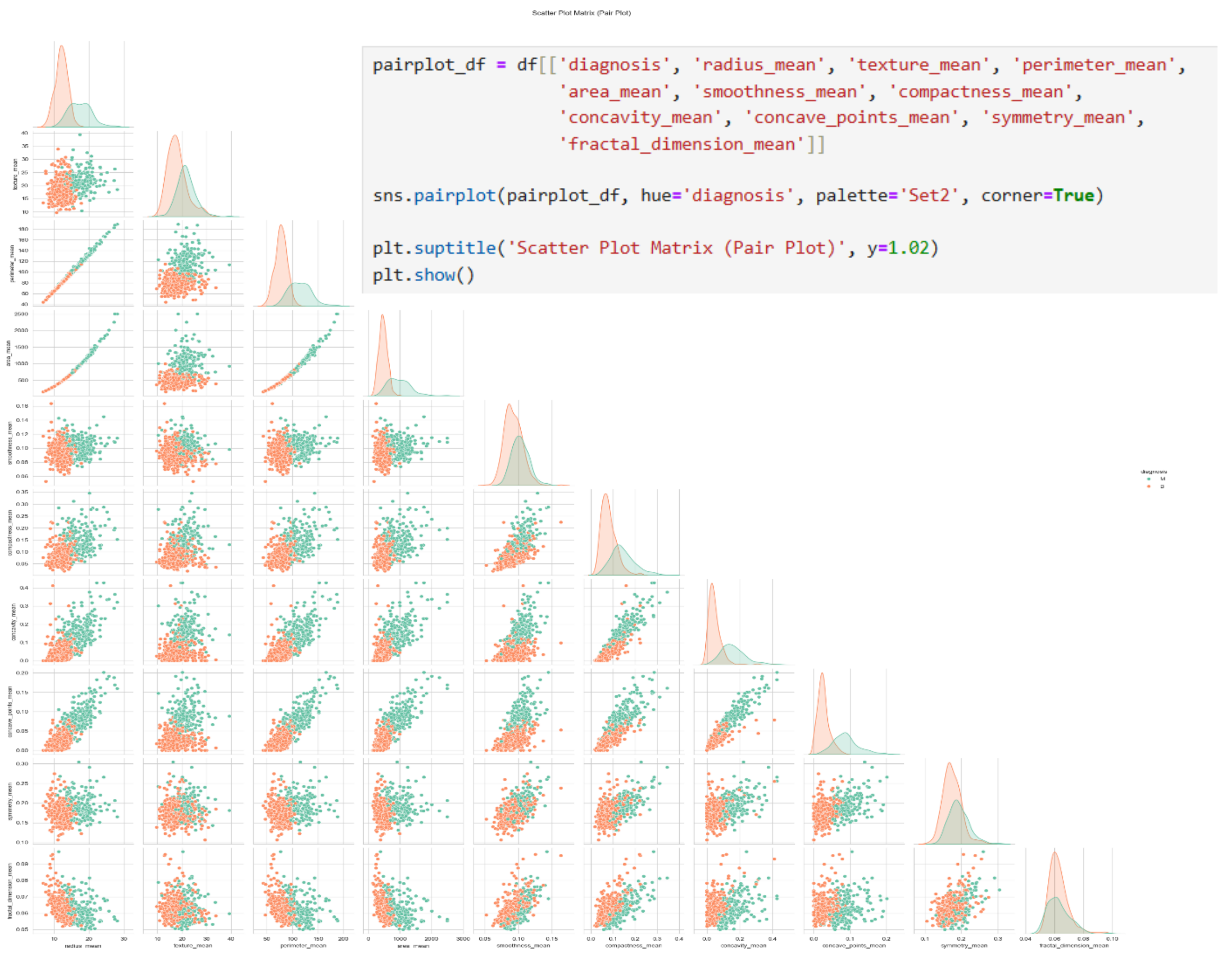


Figure 8 Correlation Matrix Heatmap

Modelling and Visualisation

I used a scatter plot matrix (pair plot) to explore the relationships between features, with points colored by diagnosis group. This helps us visually assess how features interact with each other and whether certain combinations can separate the two classes. It also highlights patterns, clusters, and potential correlations across the dataset.



```
pairplot_df = df[['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
                  'area_mean', 'smoothness_mean', 'compactness_mean',
                  'concavity_mean', 'concave_points_mean', 'symmetry_mean',
                  'fractal_dimension_mean']]

sns.pairplot(pairplot_df, hue='diagnosis', palette='Set2', corner=True)

plt.suptitle('Scatter Plot Matrix (Pair Plot)', y=1.02)
plt.show()
```

Figure 9 Scatter plot with code

To visualise the data in a lower-dimensional space, I first applied PCA to reduce the original 10 features to 2 principal components. This helped show how the data points cluster based on these components, with colours indicating the diagnosis groups. I then extended the PCA analysis by including all 30 features to capture more variance and uncover richer patterns in the data. After standardising the features, I again reduced them to 2 components, and the resulting scatter plot provided a clearer view of how well the diagnosis groups separate, offering deeper insight into the dataset's structure.

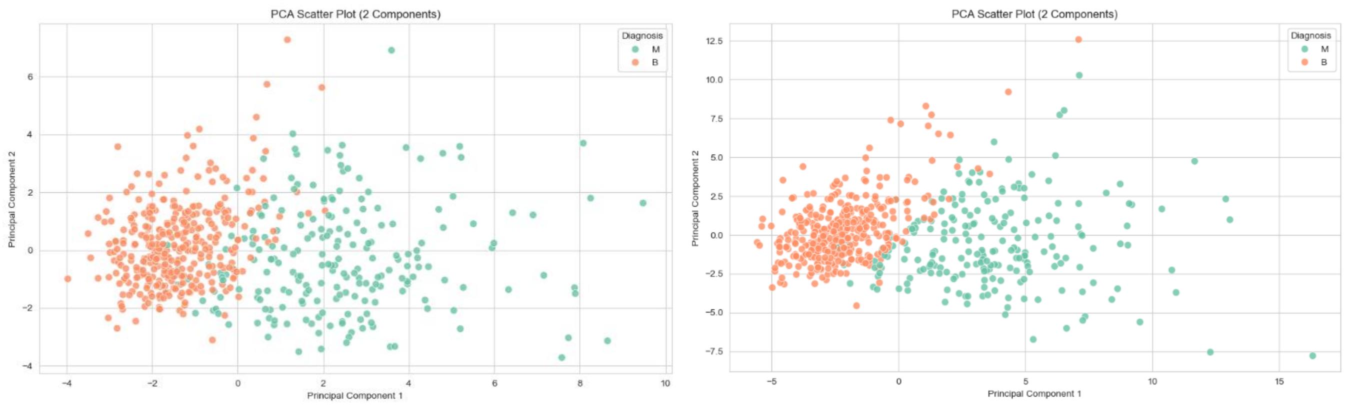
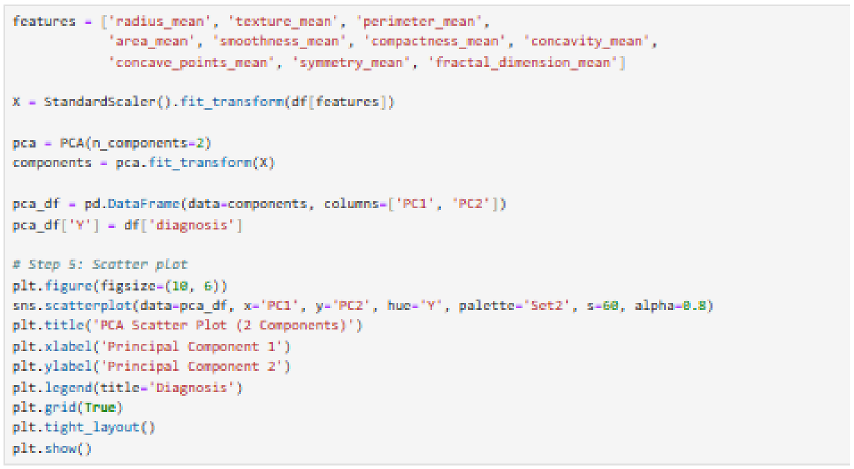


Figure 11 PCA with 10 features on left and with 30 features on right



```
features = ['radius_mean', 'texture_mean', 'perimeter_mean',
            'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
            'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean']

X = StandardScaler().fit_transform(df[features])

pca = PCA(n_components=2)
components = pca.fit_transform(X)

pca_df = pd.DataFrame(data=components, columns=['PC1', 'PC2'])
pca_df['Y'] = df['diagnosis']

# Step 5: Scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Y', palette='Set2', s=60, alpha=0.8)
plt.title('PCA Scatter Plot (2 Components)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Diagnosis')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Figure 10 Code for the PCA (10 features)

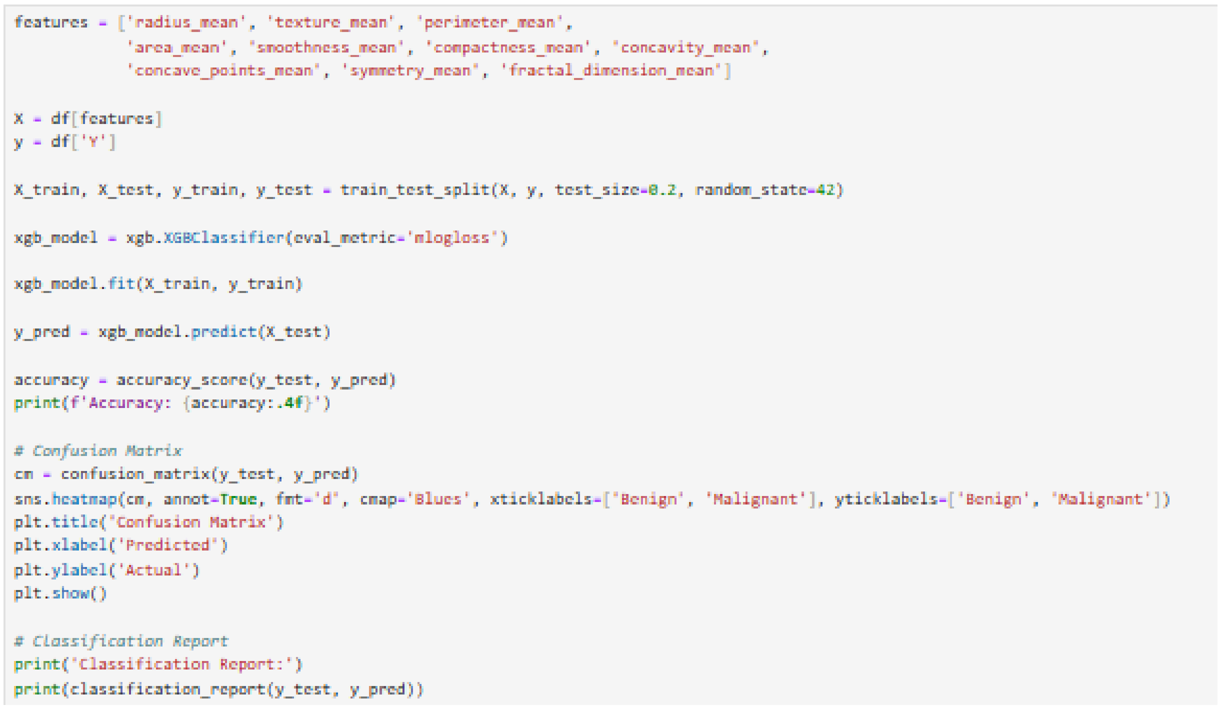
After conducting various analyses and visualisations to explore the data, I moved on to applying machine learning models. Based on prior research, we already had an idea of how different models tend to perform on this dataset. As outlined in the project objectives, the main goal was to use **XGBoost** and explore two different methods to oversee the class imbalance.

Before applying these techniques, I first trained baseline XGBoost models to serve as a reference for comparison. I created two versions:

1. **Model 1** using the ten primary features (mean values),
2. **Model 2** using all thirty features (mean, standard error, and worst-case values).

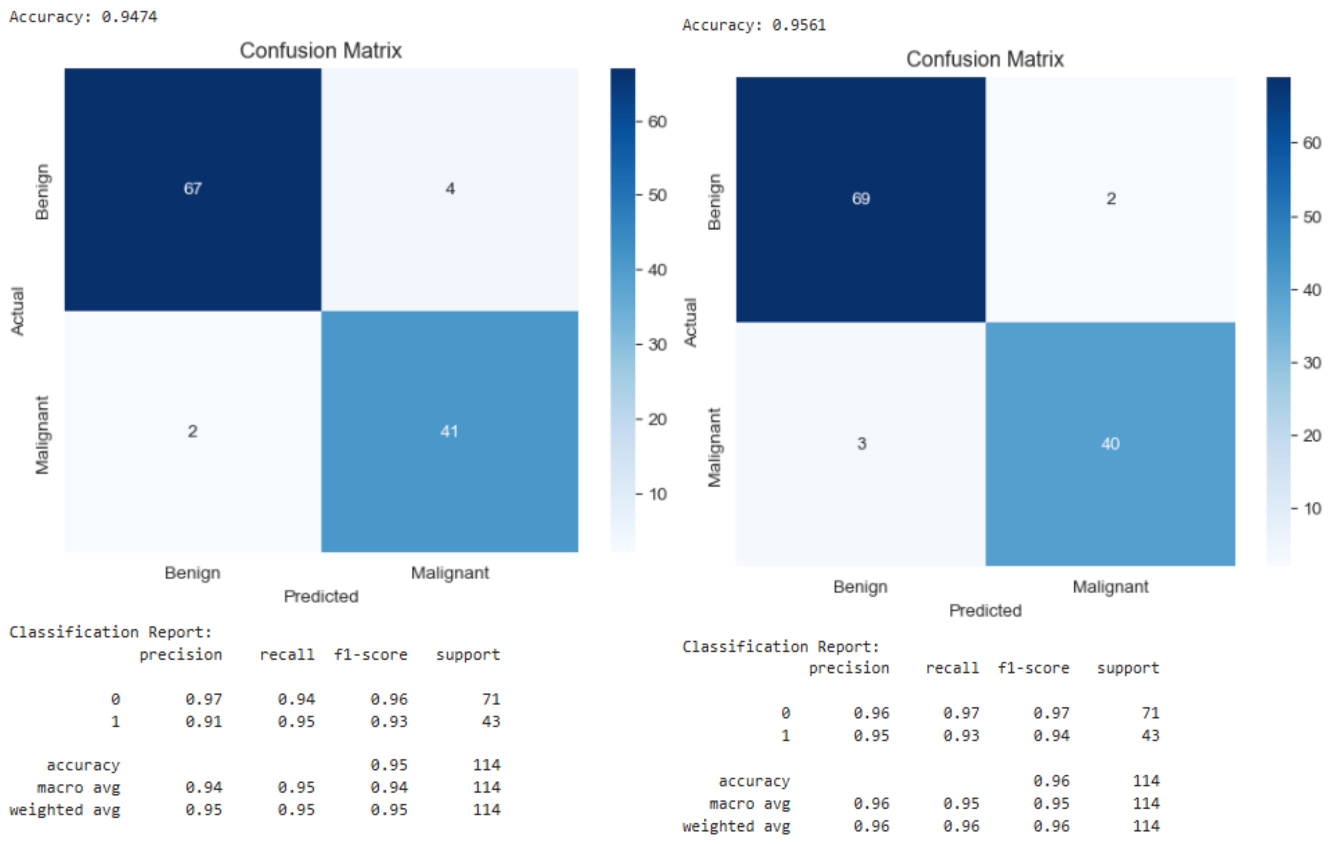
For each model, I standardly split the data into training and test sets, trained the XGBoost classifier, and evaluated its performance using accuracy, a confusion matrix, and a classification report. Additionally, for the model using all thirty features, I plotted feature importance based on gain to understand which features had the greatest impact on predictions (Figure 14).

These baseline models help us assess how well XGBoost performs without any resampling or imbalance-handling techniques and provide a benchmark to measure improvements against once these methods are applied.



```
features = ['radius_mean', 'texture_mean', 'perimeter_mean',
            'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
            'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean']

X = df[features]
y = df['Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

xgb_model = xgb.XGBClassifier(eval_metric='mlogloss')

xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Classification Report
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

Figure 12 Code for model 1 (model 2 is same approach just with 30 features)



```
Accuracy: 0.9474
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        71
           1       0.91      0.95      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.94       114
weighted avg       0.95      0.95      0.95       114
```

```
Accuracy: 0.9561
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Figure 14 Confusion matrix for baseline models (model 1 on left and model 2 on right)

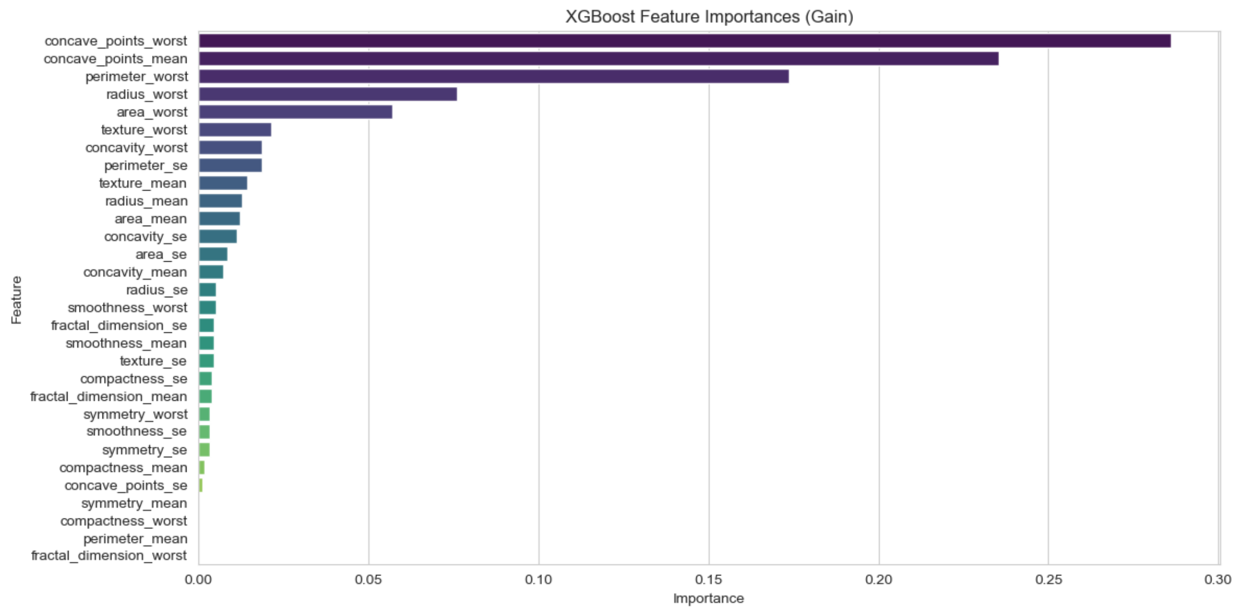
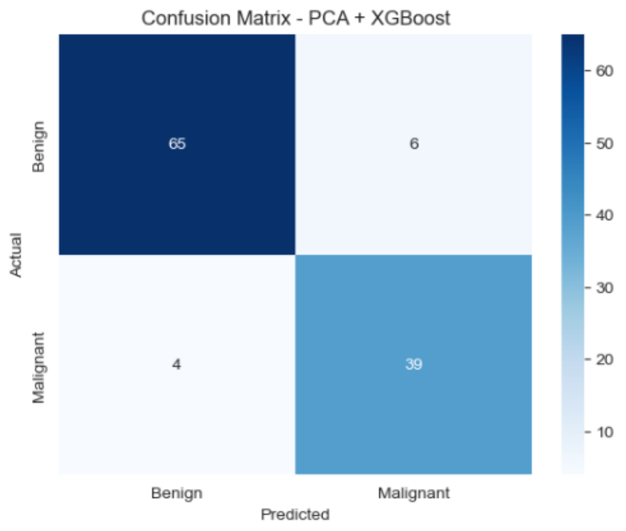


Figure 13 Feature importance

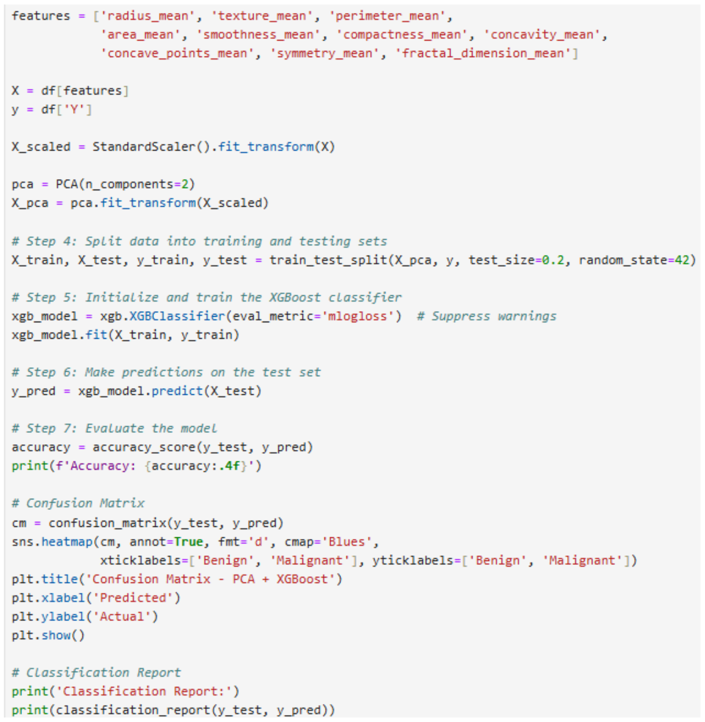
From the confusion matrices plotted for both models, we can see that the performance is generally strong. However, both models still misclassify 2 or 3 malignant cases as benign. This is concerning, as in this context, false negatives (Type II errors) are especially critical—we want to avoid labelling malignant tumours as benign. Reducing these false negatives is a key objective, as early detection of cancer is vital for effective treatment.

I also trained and tested a model using the PCA components as input features. As expected, the performance was slightly lower compared to the models trained on the original features. This is because dimensionality reduction with PCA inevitably leads to some loss of variance, which can affect the model's ability to make accurate predictions.

```
Accuracy: 0.9123
```



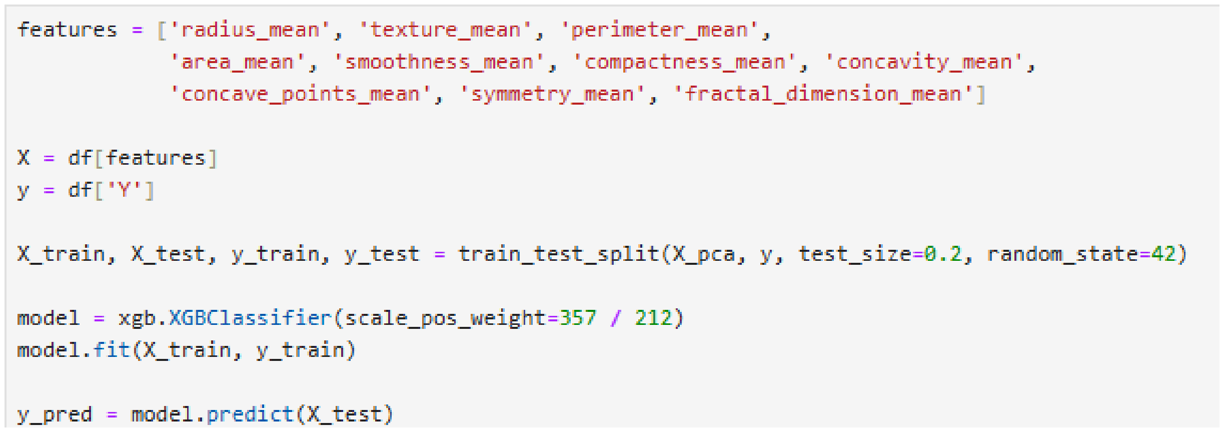
```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        71
           1       0.87      0.91      0.89        43

    accuracy                           0.91       114
   macro avg       0.90      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```



```python
features = ['radius_mean', 'texture_mean', 'perimeter_mean',
            'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
            'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean']

X = df[features]
y = df['Y']

X_scaled = StandardScaler().fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Step 5: Initialize and train the XGBoost classifier
xgb_model = xgb.XGBClassifier(eval_metric='mlogloss')  # Suppress warnings
xgb_model.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = xgb_model.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
plt.title('Confusion Matrix - PCA + XGBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Classification Report
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

Figure 15 Confusion matrix and code for PCA components model

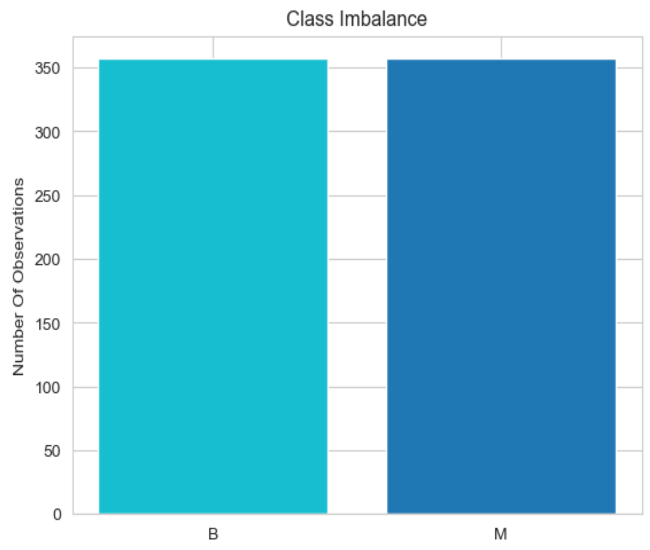
To address the class imbalance in the dataset, I first used XGBoost with the scale_pos_weight parameter (see Figure 16). This parameter adjusts the weight of the positive class, helping the model better handle imbalance. I set scale_pos_weight = 357 / 212, based on the actual distribution of the target variable. The rest of the modelling process remained the same as before.

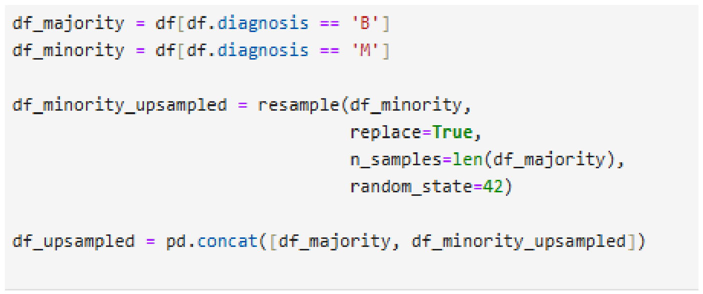


```
features = ['radius_mean', 'texture_mean', 'perimeter_mean',
            'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
            'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean']

X = df[features]
y = df['Y']

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

model = xgb.XGBClassifier(scale_pos_weight=357 / 212)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

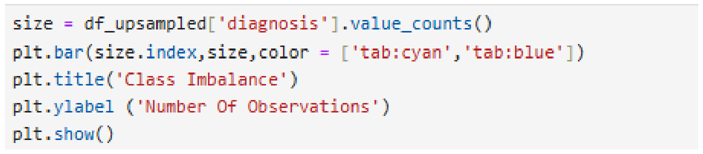
Figure 16 Code for XGBoost with 'scale_pos_weight'

Next, I applied over-sampling using resample from Scikit-learn to balance the classes manually. This involved upsampling the minority class to match the majority class, resulting in a new, balanced dataset where both classes are equally represented.





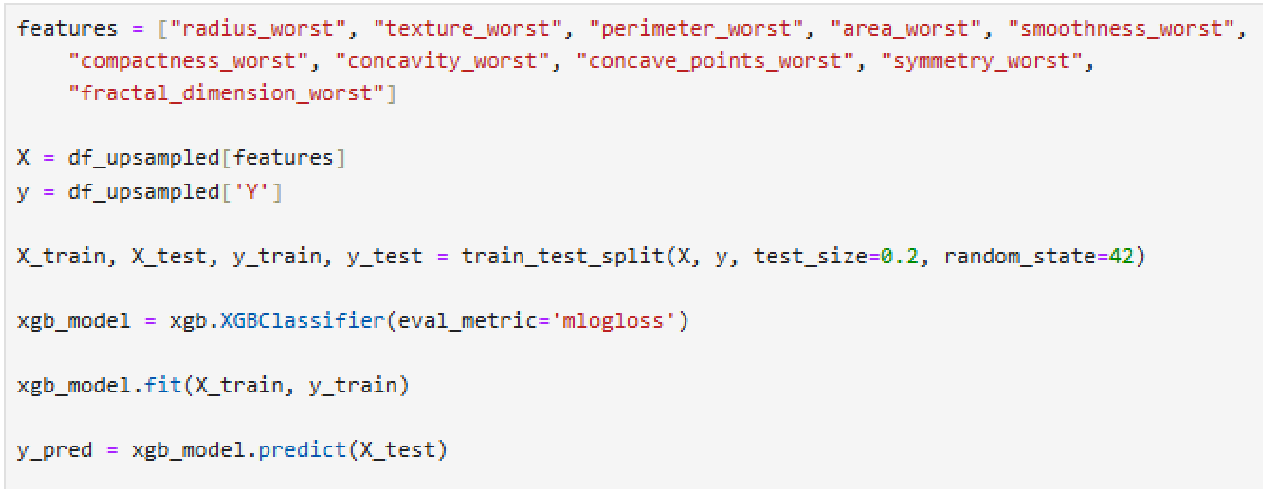
```
df_majority = df[df.diagnosis == 'B']
df_minority = df[df.diagnosis == 'M']

df_minority_upsampled = resample(df_minority,
                                 replace=True,
                                 n_samples=len(df_majority),
                                 random_state=42)

df_upsampled = pd.concat([df_majority, df_minority_upsampled])
```



```
size = df_upsampled['diagnosis'].value_counts()
plt.bar(size.index,size,color = ['tab:cyan','tab:blue'])
plt.title('Class Imbalance')
plt.ylabel ('Number Of Observations')
plt.show()
```

Figure 17 Class balanced with code

After balancing the data, I applied XGBoost again—both with and without feature selection. For the feature selection approach, I chose the most important value (the _worst) for each feature, based on earlier analysis. I then trained the model on the balanced dataset and evaluated the results using a confusion matrix and classification metrics. This allowed for a direct comparison of model performance with and without feature selection on the balanced data.

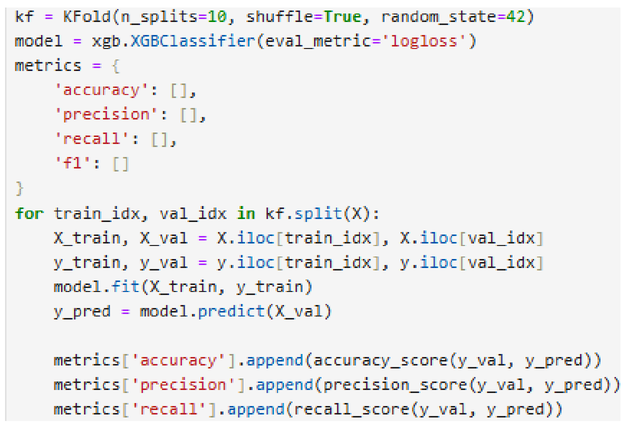


```
features = ["radius_worst", "texture_worst", "perimeter_worst", "area_worst", "smoothness_worst",
    "compactness_worst", "concavity_worst", "concave_points_worst", "symmetry_worst",
    "fractal_dimension_worst"]

X = df_upsampled[features]
y = df_upsampled['Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

xgb_model = xgb.XGBClassifier(eval_metric='mlogloss')

xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)
```

Figure 18 New data (balanced) model training code

Note: The code for both models is largely similar to the previous implementations. To avoid redundancy, I have not included all the code here. The full code can be found in the accompanying Python notebook.

As the results were promising, I performed 10-fold cross-validation to ensure their reliability. I evaluated each fold using accuracy, precision, recall, and F1-score, and then calculated the average across all ten folds to assess overall model performance.



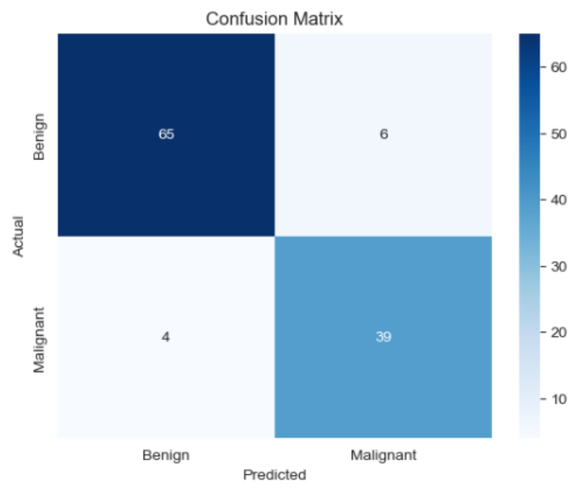
```
kf = KFold(n_splits=10, shuffle=True, random_state=42)
model = xgb.XGBClassifier(eval_metric='logloss')
metrics = {
    'accuracy': [],
    'precision': [],
    'recall': [],
    'f1': []
}
for train_idx, val_idx in kf.split(X):
    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)

    metrics['accuracy'].append(accuracy_score(y_val, y_pred))
    metrics['precision'].append(precision_score(y_val, y_pred))
    metrics['recall'].append(recall_score(y_val, y_pred))
```

Figure 19 Cross validation with 10 folds code

Evaluation

As we can see below (Figure 20) the confusion matrix and classification summary does not show any improvements with the results when we used 'scale_pos_weight' with XGBoost.

```
Accuracy: 0.9123
```

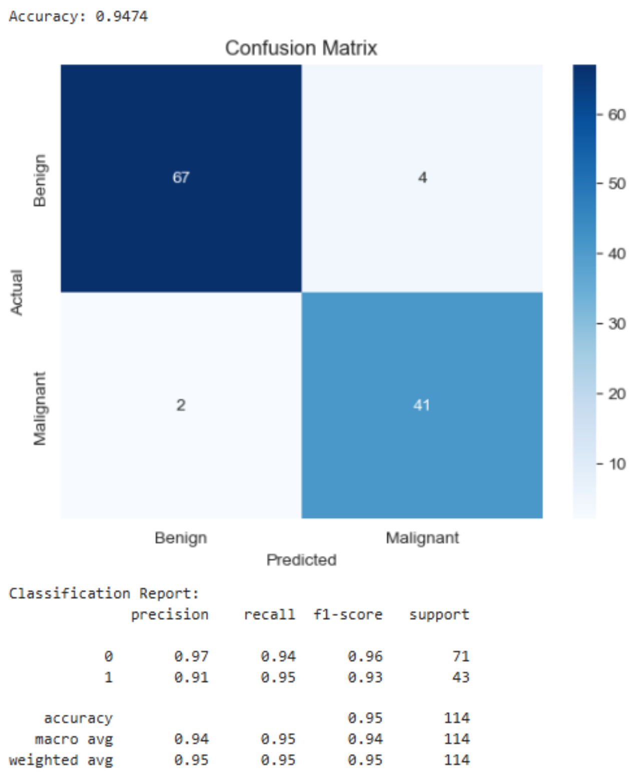


```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        71
           1       0.87      0.91      0.89        43

    accuracy                           0.91       114
   macro avg       0.90      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```

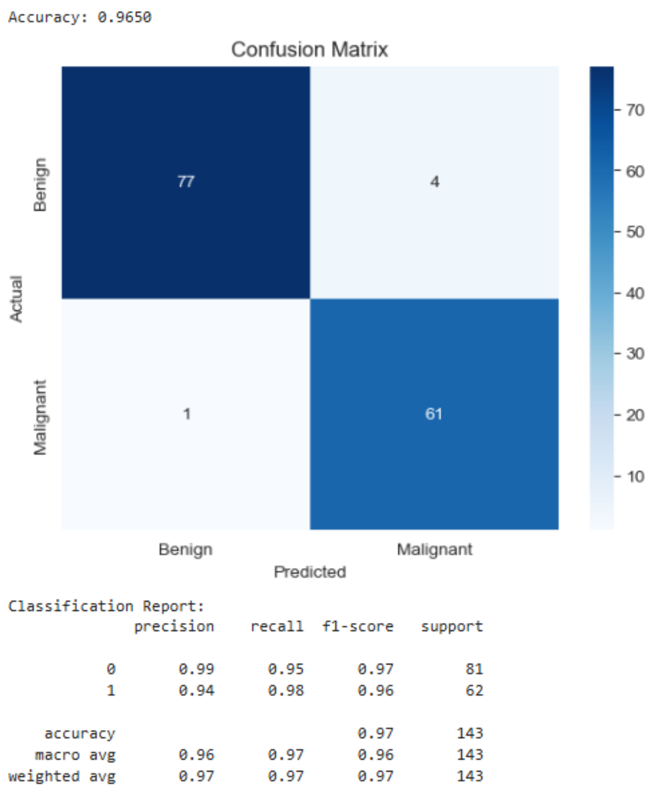
Figure 20 Confusion matrix XGBoost with 'scale_pos_weight'

Figure 21 compares the results of XGBoost models without feature selection on both the original imbalanced data and the newly balanced dataset. We can observe that, on the balanced data, the model only misclassified one malignant case, indicating a strong improvement in handling the minority class.

Figure 22 presents the results for the XGBoost model with feature selection. On the original imbalanced data, feature selection led to a slight improvement in performance. When applied to the balanced dataset, the model showed a more noticeable enhancement, further reducing misclassifications and improving overall reliability.

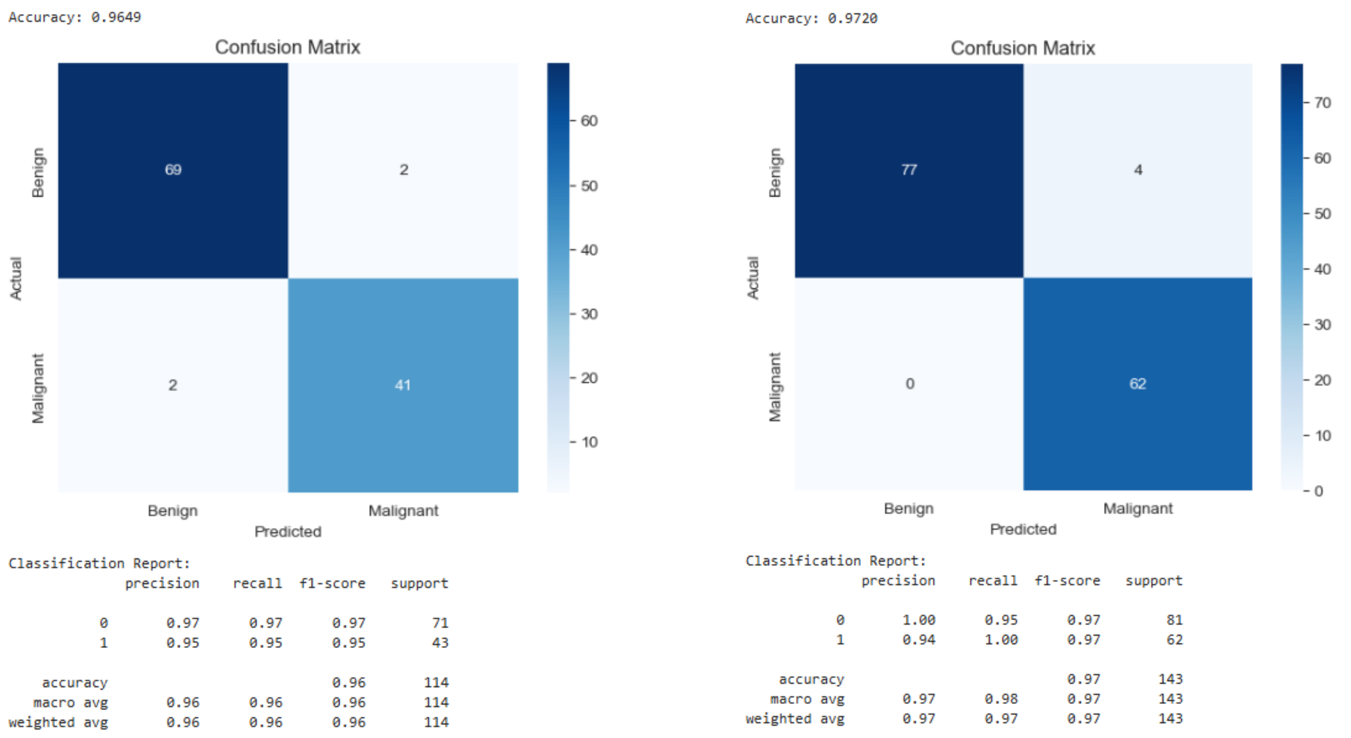

```
Accuracy: 0.9474
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        71
           1       0.91      0.95      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.94       114
weighted avg       0.95      0.95      0.95       114
```


```
Accuracy: 0.9650
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97        81
           1       0.94      0.98      0.96        62

    accuracy                           0.97       143
   macro avg       0.96      0.97      0.96       143
weighted avg       0.97      0.97      0.97       143
```

Figure 21 Comparing confusion matrix and summary on original data (left) and balanced data (right)



```
Accuracy: 0.9649

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        71
           1       0.95      0.95      0.95        43

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114
```

```
Accuracy: 0.9720

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        81
           1       0.94      1.00      0.97        62

    accuracy                           0.97       143
   macro avg       0.97      0.98      0.97       143
weighted avg       0.97      0.97      0.97       143
```

Figure 22 Comparing confusion matrix and summary on original data (left) and balanced data (right) with feature selection

Figure 23 illustrates the accuracy, precision, recall, and F1-score across the 10 folds in our cross-validation. This visual representation provides a clearer understanding of the model's consistency and performance across different subsets of the data.

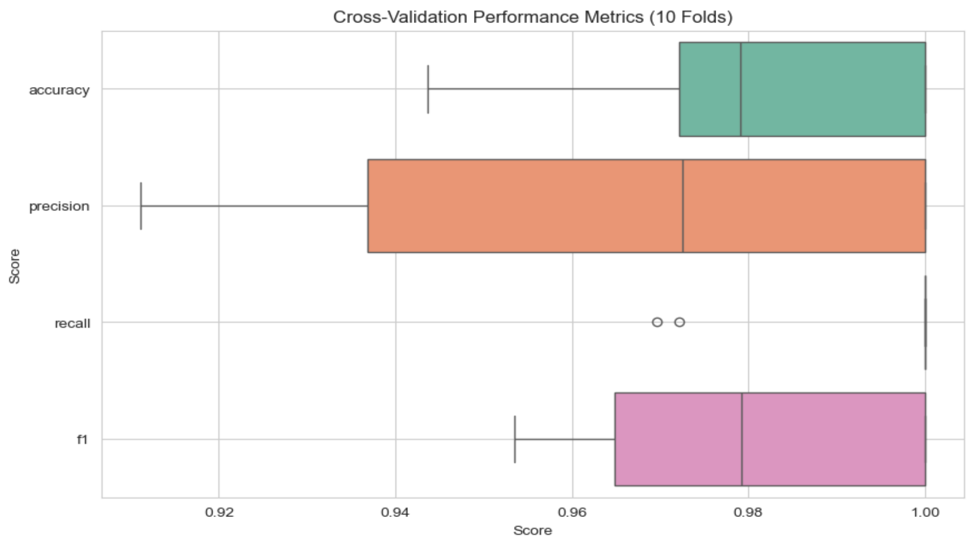


Figure 23 Cross validation results

Conclusion

In this project, we analysed the Breast Cancer Wisconsin dataset to understand feature distributions and their relationship with the diagnosis. We applied PCA for dimensionality reduction and observed its effect on model performance. The main focus was on training XGBoost models, where we addressed class imbalance using scale_pos_weight and oversampling.

We also tested the impact of feature selection on both imbalanced and balanced datasets. Results showed that balancing the data significantly reduced false negatives, which is crucial in medical diagnosis. Cross-validation confirmed the reliability of the models, showing consistent and strong performance across all metrics.

Reference List

Khan, M. A. & Islam, M. T. (2023). Artificial Intelligence for Medical Diagnostics—Existing and Future AI Technology. *International Journal of Environmental Research and Public Health*, 20(3), 5430.

Kumar, N., Verma, R., Sharma, S., Bhargava, S., Vahadane, A. & Sethi, A. (2017). A dataset and a technique for generalized nuclear segmentation for computational pathology. *IEEE Transactions on Medical Imaging*, 36(7), 1551-1560.

Radhakrishnan, A., Damodaran, K., Soylemezoglu, A. C., Uhler, C. & Shivashankar, G. V. (2017). Machine learning for nuclear mechano-morphometric biomarkers in cancer diagnosis. *Scientific Reports*, 7(1), 17946.

Street, W. N., Wolberg, W. H. & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. *Proceedings of SPIE 1905: Biomedical Image Processing and Biomedical Visualization*.

Sharma, A., Kulshrestha, S. & Daniel, S. B. (2018). Machine learning approaches for cancer detection. *International Journal of Engineering and Manufacturing*, 8, 45-55.

Ćwiklińska-Jurkowska, M. M. (2013). Performance of resampling methods based on decision trees, parametric and nonparametric Bayesian classifiers for three medical datasets. *Studies in Logic, Grammar and Rhetoric*, 35(1), 71-86.

Wolberg, W. H., Street, W. N. & Mangasarian, O. L. (1995). Image analysis and machine learning applied to breast cancer diagnosis and prognosis. *Analytical and Quantitative Cytology and Histology*, 17(2), 77-87.

Sidey-Gibbons, J. & Sidey-Gibbons, C. (2019). Machine learning in medicine: a practical introduction. *BMC Medical Research Methodology*, 19, 64.

He, H. & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263-1284.

Chen, T. & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, New York, NY, USA, 785–794.

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, 321-357.

Kohavi, R. (1995). A study of cross-validation and bootstraps for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2* (IJCAI'95), San Francisco, CA, USA, 1137–1143. Morgan Kaufmann Publishers Inc.