Task 3: Models and Metrics

Yevgen Petroyuk Pidhaynyy

1. Data Preprocessing

Each of the programmes handles the data before training it and testing it with different machine learning algorithms.

First, the programmes load and inspect the data. Here we have an example of one of the programmes:

```
def load_and_inspect_data(file_path):
    data = pd.read_csv(file_path)
    print("\nValores faltantes por columna:\n", data.isnull().sum())
    print("\nPrimeras filas del dataset:\n", data.head())
    return data
```

As we can see, we look for missing values in each column, and print the first rows of the dataset to have a visual idea of our dataset structure.

Second, each of the programmes handle missing values by using SimpleImputer, filling out missing values with the mean value of the entire column. Also, it encodes categorical variables using LabelEncoder.
After this process, the data is being scaled using StandartScaler() from the scikit learn library. It standarizes the features so that each feature has mean 0, and variance 1.
Here we can see the process:

```
def preprocess_data(data):
    imputer = SimpleImputer(strategy='mean')
    data_numeric = data.select_dtypes(include=['float64', 'int64'])
    data[data_numeric.columns] = imputer.fit_transform(data_numeric)

    label_encoders = {}
    for column in data.select_dtypes(include=['object']).columns:
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le

    return data, label_encoders

def scale_data(data, feature_columns):
    scaler = StandardScaler()
    data[feature_columns] = scaler.fit_transform(data[feature_columns])
    return data
```

The last step of our data preprocessing process is spliting the data. We split it in order to have a portion of the data destinied to train the model, while the other part of the data is used to test the model. The programmes use a 80%-20% data split.

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

Here we can see how the test_size parameter is set to 0,2. This represents the portion of out data that is going to be used for testing (20%).

2. Classification Task

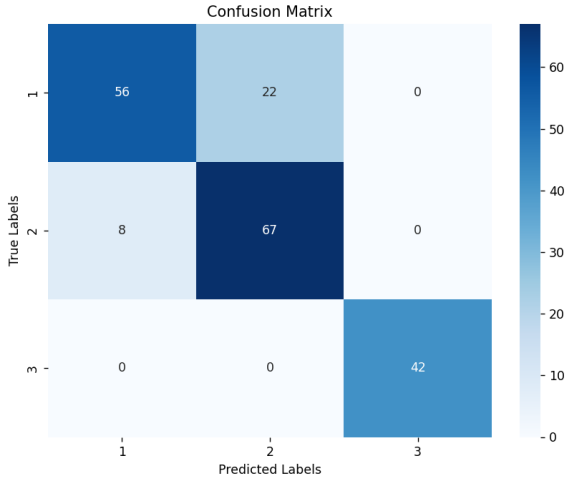
- K-Nearest Neighbors (KNN)

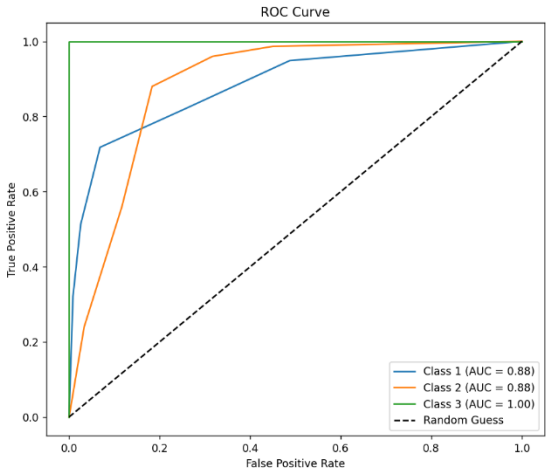
KNN is a simple machine learning algorithm that uses the concept of distance to predict which class does an input belong to. This algorithm assumes that data belonging to the same class are close one to each other. By assuming this, it calculates the distance of the input to each of the “data points”.
KNN algorithm requires of a parameter, which is the number of neighbours it is going to have into account. As an example, we give it 7 as the neighbout number parameter. KNN is going to search for the 7 nearest neighbours. After that, it is going to check what classes do those neighbours belong to. The predicted class will be the majority class.

Here we can see how our programme calculated different metrics for our KNN algorithm predicting *Experience_Level* in our *gym_members_exercise_tracking.csv* file:

```
Correct Predictions: 165 out of 195
Accuracy: 84.62%
Precision: 85.49%
Recall: 84.62%
F1 Score: 84.51%
```

Also, we can see how our programme calculated Confusion Matrix and ROC curves (as well as AUC scores) to evaluate the KNN agorithm:





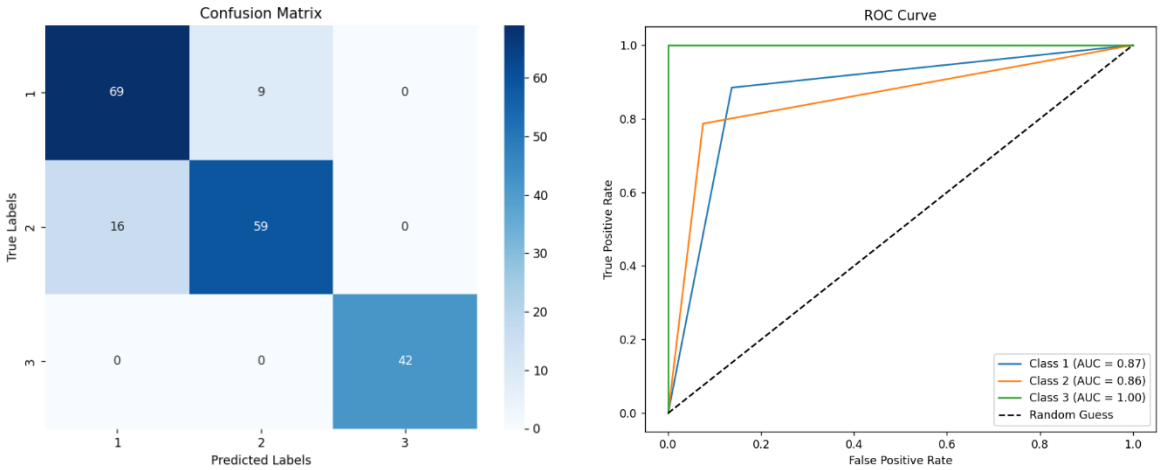
- Decision Tree Classifier

A Decision Tree Classifier is an algorithm that uses a tree structure to split data into smaller groups. It doesn't split the data randomly; instead, it looks for the best way to divide it. Given that data often has many attributes, the algorithm searches for the best "if-else" condition to split the data into two parts using one of the attributes. It evaluates all possible conditions and chooses the one that provides the greatest information gain, which is a measure of how well the split helps to classify the data.
The algorithm continues splitting the data until each group (or "leaf") contains data points from a single class, or until it reaches a predefined limit on the number of splits.
To make a prediction, the algorithm sends a new data sample through the tree. At each node, it answers an "if-else" question based on the sample's attributes, moving down the tree until it reaches a leaf. The class of the sample is determined by the most common class in that leaf.

Here we can see how our programme evaluated the Decision Tree Classifier when predicting *Experience_Level* variable, using different evaluation metrics:

```
Correct Predictions: 170 out of 195
Accuracy: 87.18%
Precision: 87.38%
Recall: 87.18%
F1 Score: 87.14%
```

As well as Confusion Matrix, ROC curves and AUC scores:



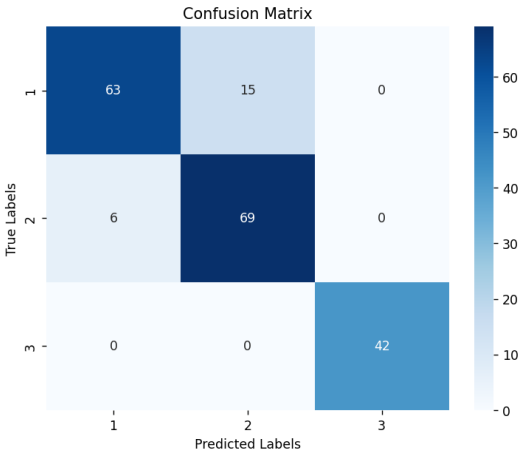
- Random Forest Classifier

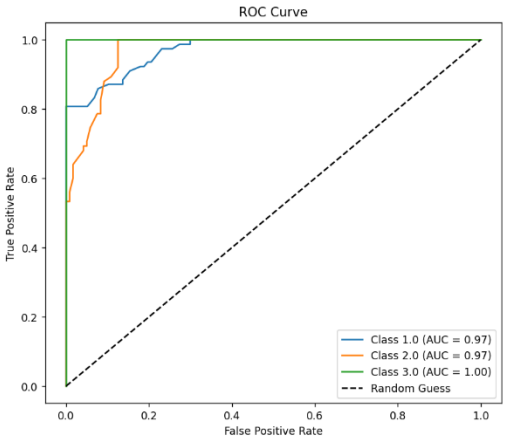
A Random Forest Classifier is a learning algorithm that builds multiple decision trees to make predictions. Instead of relying on a single decision tree, the random forest creates a "forest" of trees, each trained on a different subset of the data. This helps to improve the model's accuracy and robustness.
To train a random forest, the algorithm generates multiple decision trees using a technique called bootstrapping, where each tree is trained on a random sample of the data, and each split in a tree is based on a random subset of features. This randomness ensures that the trees are diverse and less likely to overfit the data.
Once the trees are trained, the random forest makes a prediction by aggregating the results from all the individual trees. For classification tasks, the prediction is typically made by majority voting — the class that is predicted by most of the trees becomes the final prediction.

Here we can see how our programme evaluated the Random Forest Classifier when predicting *Experience_Level* variable, using different evaluation metrics:

```
Correct Predictions: 174 out of 195
Accuracy: 89.23%
Precision: 89.65%
Recall: 89.23%
F1 Score: 89.21%
```

Also, the Confusion Matrix, ROC curves and AUC scores:

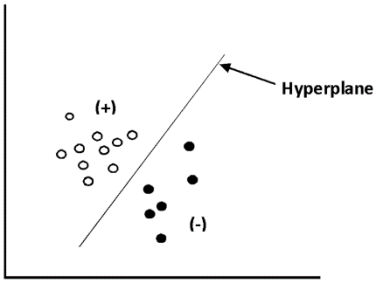




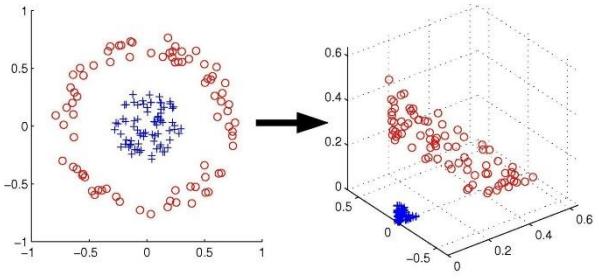
- Support Vector Machine (SVM)

A Support Vector Machine (SVM) Classifier is a supervised learning algorithm used for classification tasks. It works by finding the optimal hyperplane that separates data points from different classes in the feature space.
In order to achieve this, each of the features of our data samples have an associated weight, as well as a common bias. When the model is training, it adjusts those weights in order to get the best hyperplane (just like neural networks do).
The key idea is to maximize the margin between the nearest data points of each class, known as support vectors. A larger margin generally leads to better generalization, meaning the model is less likely to overfit and performs better on unseen data.
When the data is not linearly separable, SVM uses a technique called the kernel trick. This transforms the data into a higher-dimensional space where a linear hyperplane can be found to separate the classes. Common kernel functions include linear, polynomial, and radial basis function (RBF), depending on the complexity of the data.
Once the optimal hyperplane is identified, new data points are classified based on which side of the hyperplane they fall on.

Linearly separable:



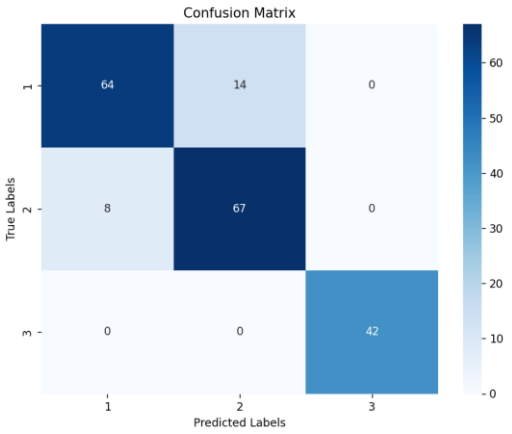
Non linearly separable (uses Kernel):

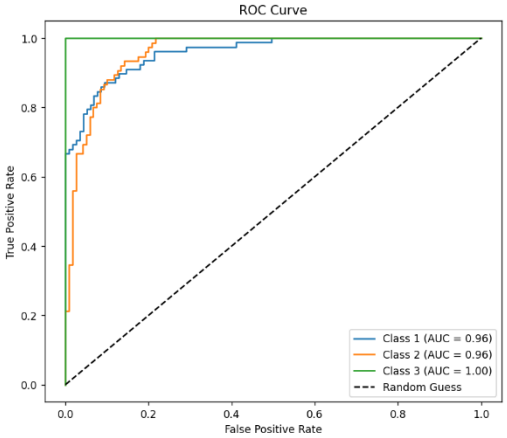


Here we can see how our programme evaluated the Support Vector Machine when predicting *Experience_Level* variable, using different evaluation metrics:

```
Correct Predictions: 173 out of 195
Accuracy: 88.72%
Precision: 88.91%
Recall: 88.72%
F1 Score: 88.71%
```

Also, the Confusion Matrix, the ROC curves, and the AUC scores:





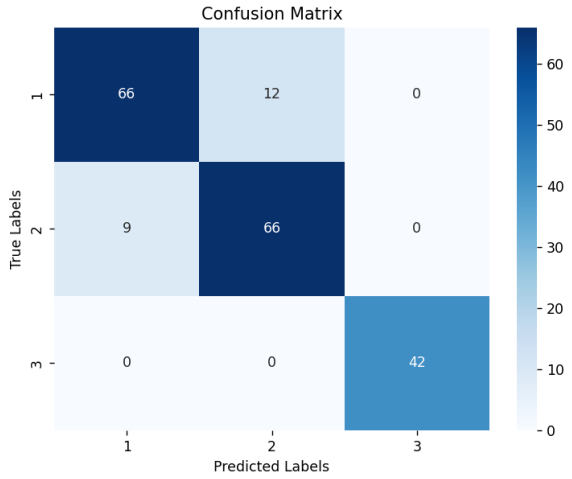
- Naive Bayes Classifier

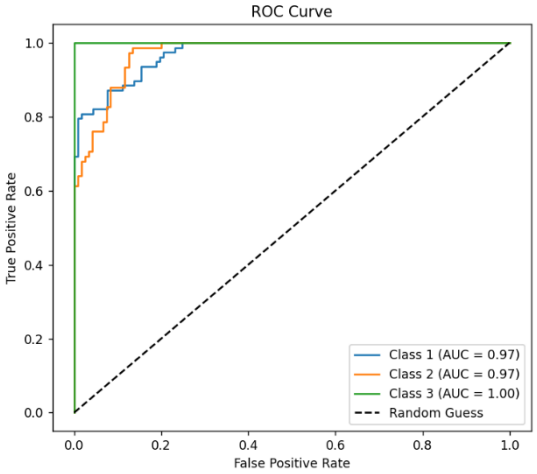
A Naive Bayes Classifier is a probabilistic learning algorithm based on Bayes' Theorem, which is used for classification tasks. It assumes that the features in the dataset are independent of each other, hence the term "naïve".
To train a Naive Bayes classifier, the algorithm calculates the probability of each class given the input features using Bayes' Theorem. It first estimates the prior probability of each class (how likely each class is before seeing the data), and then calculates the likelihood of the features given each class (how likely the features are in each class). The classifier combines these two to compute the posterior probability for each class and selects the class with the highest posterior probability as the predicted class.

Here we can see how our programme evaluated the Support Vector Machine when predicting *Experience_Level* variable, using different evaluation metrics:

```
Correct Predictions: 174 out of 195
Accuracy: 89.23%
Precision: 89.28%
Recall: 89.23%
F1 Score: 89.23%
```

Also, the Confusion Matrix, the ROC curves, and the AUC scores:





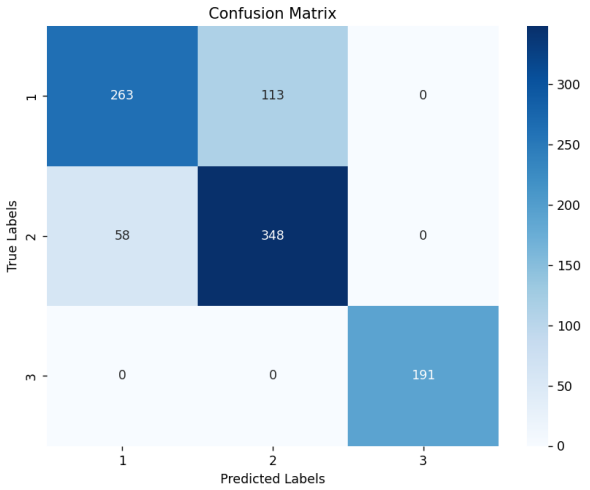
- K-Nearest Neighbors (Cross Validation)

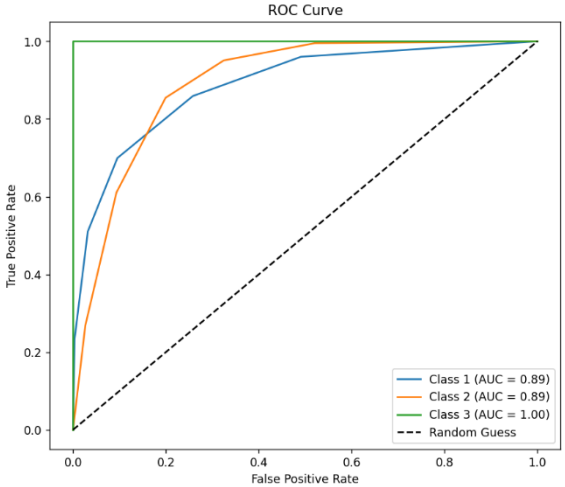
K-fold cross-validation for KNN splits the data into K subsets. The model is trained on K-1 folds and tested on the remaining fold, repeating this for each subset. The average performance across all iterations gives a more reliable estimate of the model's accuracy.

Here are the evaluation metrics for this model:

```
Correct Predictions: 802 out of 973
Accuracy: 82.43%
Precision: 82.79%
Recall: 82.43%
F1 Score: 82.29%
```

Also, the Confusion Matrix and ROC curves, as well as AUC scores:





We can see that the averaging effect reduces the variance caused by the choice of a single train-test split and results in a smoother and more stable ROC curve.

- Grid Search for Hyperparameter Tuning in SVM Classifier

Grid search is a method for tuning hyperparameters in a machine learning model, like an SVM, by testing all possible combinations of specified hyperparameter values. It uses cross-validation to evaluate each combination and selects the best one based on model performance, helping improve accuracy and generalization.

Here we can see the hyperparameter tuning results:

```
Best Hyperparameters found by GridSearchCV:
{'C': 0.1, 'class_weight': 'balanced', 'degree': 3, 'gamma': 'scale', 'kernel': 'linear'}
```

Nonetheless, the results were very similar to our original SVM classifier model (174 correct predictions out of 195).

Overall Results:

K-Nearest Neighbors (KNN): 165 /195 correct predictions
Decision Tree Classifier: 170/195 correct predictions
Random Forest Classifier: 174/195 correct predictions
Support Vector Machine (SVM): 173/195 correct predictions
Naive Bayes: 174/195 correct predictions
K-Nearest Neighbors with Cross Validation: 802/973 correct predictions
SVM with hyperparameter tuning: 174/195 correct predictions

We can see that all models had similar results and made very good predictions. The best performing models were Random Forest Classifier and Naive Bayes Classifier. This can be seen in the number of correct predictions, but also seeing the ROC curve and the AUC scores for each class. Both of them had 0,97 for class 1 and 2; and 1.00 for class 3.

3. Regression Task

- Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship, where the model predicts the dependent variable as a weighted sum of the independent variables. The goal is to find the best-fit line that minimizes the difference between predicted and actual values.

Here we have some metrics that have evaluated our linear regression model predicting Calories_Burned:

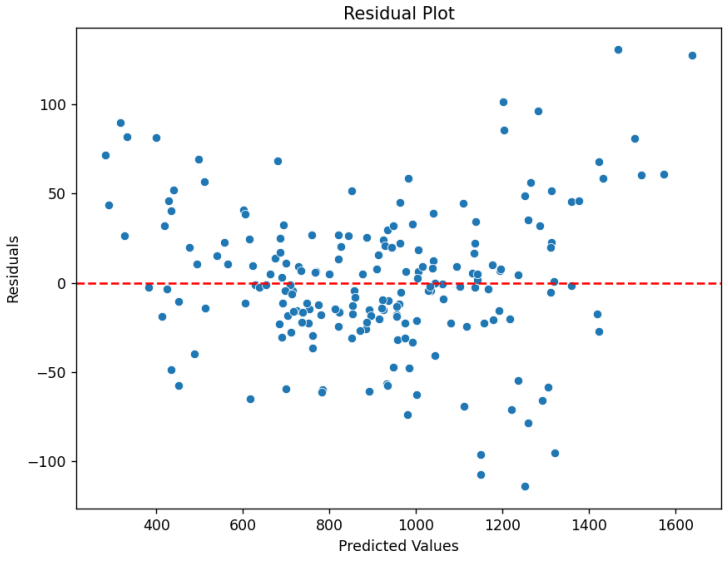
```
Mean Absolute Error (MAE): 30.22
Mean Squared Error (MSE): 1639.91
Root Mean Squared Error (RMSE): 40.50
R² (Coefficient of Determination): 0.98
```

The model's performance is strong:

- MAE (30.22) and RMSE (40.50) are relatively small errors compared to the target range (600-1300 calories), indicating good prediction accuracy.
- R^2 (0.98) shows that 98% of the variance in Calories_Burned is explained by the model, which is excellent.

Overall, these metrics suggest the model performs very well with minimal error.

Here we have the residual plot:



We can see that residuals are randomly scattered around the plot. This means our model fit is quite good.

- Polynomial Regression

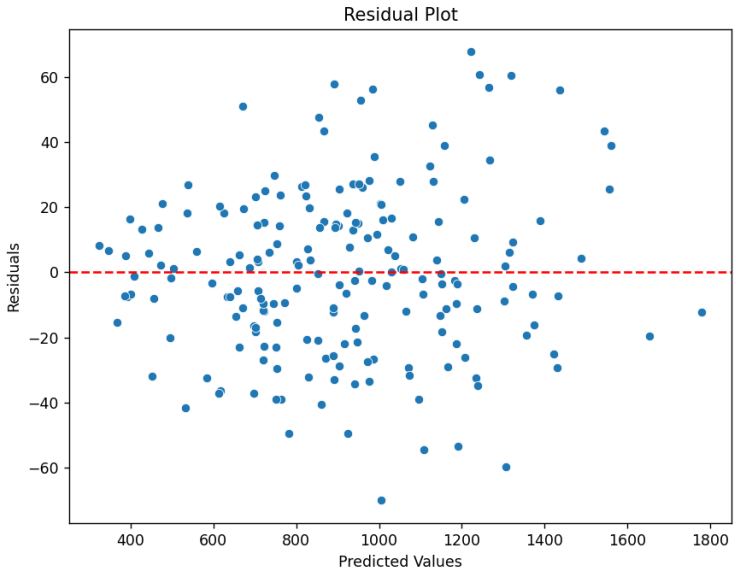
Polynomial regression is a type of regression analysis where the relationship between the independent variable(s) and the dependent variable is modeled as an nth-degree polynomial. It is used when data shows a curvilinear relationship that linear regression cannot capture. By adding higher-degree terms of the predictor variables, polynomial regression allows for more flexibility in fitting the data. However, it is important to avoid overfitting by selecting an appropriate degree for the polynomial.

Here we have some metrics that have evaluated our polynomial regression model predicting Calories_Burned:

```
Mean Absolute Error (MAE): 20.03
Mean Squared Error (MSE): 643.49
Root Mean Squared Error (RMSE): 25.37
R² (Coefficient of Determination): 0.99
```

The model shows excellent performance with a low Mean Absolute Error (MAE) of 20.03, and a low Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) of 643.49 and 25.37, respectively. The high R^2 value of 0.99 indicates that the model explains 99% of the variance in the data, suggesting a very strong fit.

Here we have the residual plot:



As before, we can see that residuals are randomly scattered around the plot, allowing us to make the conclusion that our model fit is good.

- Support Vector Regression (SVR)

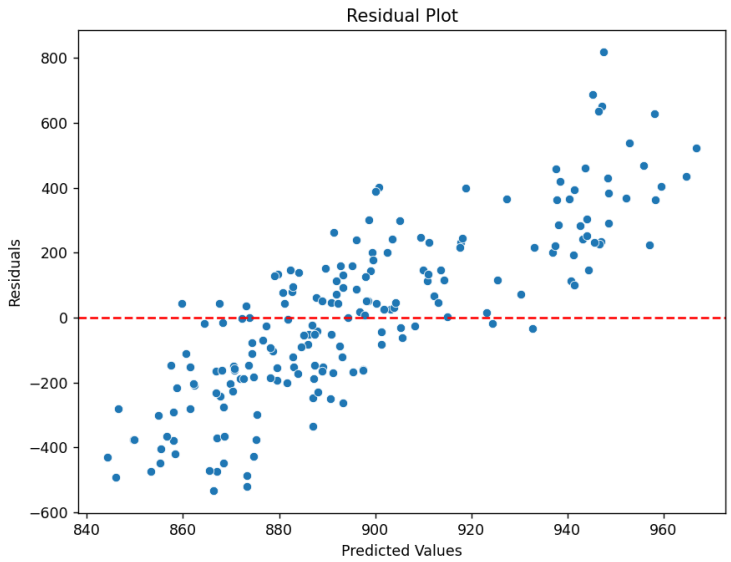
SVM (Support Vector Machine) regression is a powerful machine learning technique used for predicting continuous values. It works by finding the hyperplane that best fits the data while maximizing the margin between data points. SVM regression is effective for both linear and non-linear relationships, and it uses kernel functions (like linear, polynomial, or radial basis function) to handle complex patterns. The key advantage of SVM regression is its ability to perform well with high-dimensional data and its robustness to outliers.

Here we have some metrics that have evaluated our Support Vector regression model predicting Calories_Burned:

```
Mean Absolute Error (MAE): 211.07
Mean Squared Error (MSE): 69813.34
Root Mean Squared Error (RMSE): 264.22
R² (Coefficient of Determination): 0.16
```

The model's performance is quite poor, with a high Mean Absolute Error (MAE) of 211.07, indicating large average errors. The Mean Squared Error (MSE) of 69813.34 and Root Mean Squared Error (RMSE) of 264.22 further emphasize substantial prediction errors. The R^2 value of 0.16 shows that only 16% of the variance in the data is explained by the model, suggesting that it does not capture the underlying patterns effectively.

Also, our residual plot indicates that our model fit is bad, as the points are not randomly spread across the plot, but more in a diagonal way:



This means that the errors are systematically increasing or with the predicted values.

- Decision Tree Regressor

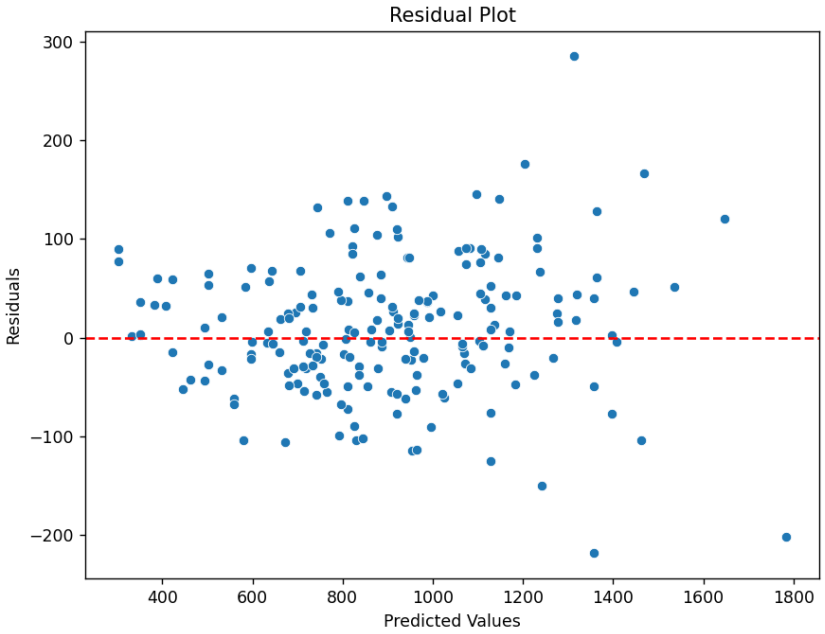
A Decision Tree Regressor is a model that predicts continuous values by splitting data into subsets based on feature values, forming a tree structure. Each node represents a decision rule, and leaf nodes represent predictions. It's easy to interpret and doesn't require feature scaling, but can overfit if the tree is too deep. Proper tuning helps improve its performance and prevent overfitting.

Here we have some metrics that have evaluated our Decision Tree regressor predicting Calories_Burned:

```
Mean Absolute Error (MAE): 52.52
Mean Squared Error (MSE): 4759.84
Root Mean Squared Error (RMSE): 68.99
R² (Coefficient of Determination): 0.94
```

The model demonstrates strong performance with an R^2 of 0.94, meaning it explains 94% of the variance in the data. The MAE of 52.52 and RMSE of 68.99 indicate moderate prediction errors, while the MSE of 4759.84 suggests some larger deviations. Overall, the model provides accurate predictions with relatively low error.

Here we have the residual plot for this model:



We can see a very small tendency in our residuals to be larger in the right side of the plot. But as this tendency is really small, we can consider this model fit valid.

- Random Forest Regressor

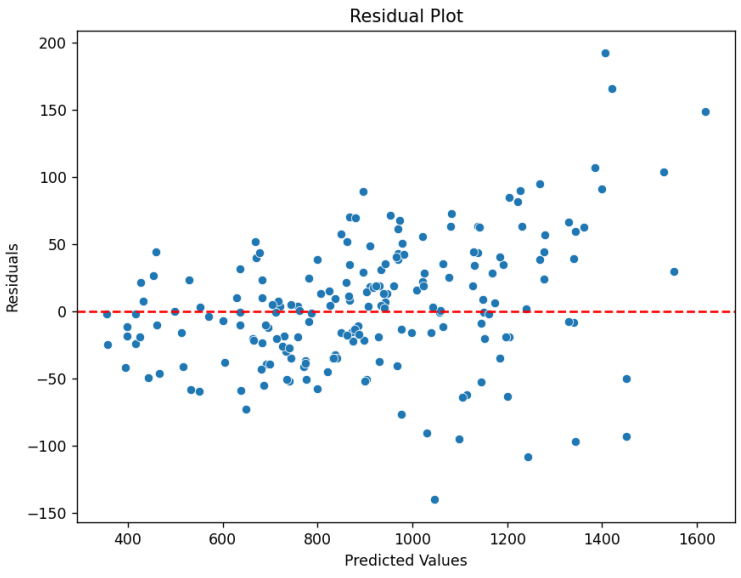
A Random Forest Regressor is an ensemble learning method that combines multiple decision trees to make predictions. It works by averaging the outputs of several decision trees, which helps reduce overfitting and improves generalization. Each tree is trained on a random subset of the data, and the final prediction is made by averaging the predictions of all individual trees. This technique is highly effective for capturing complex patterns in data, especially when there are non-linear relationships. It is robust and performs well even with large datasets and features with varying importance.

Here we have some metrics that have evaluated our Random Forest regressor predicting Calories_Burned:

```
Mean Absolute Error (MAE): 36.30
Mean Squared Error (MSE): 2292.73
Root Mean Squared Error (RMSE): 47.88
R² (Coefficient of Determination): 0.97
```

The Random Forest Regressor performs better than the Decision Tree Regressor based on these metrics. The MAE, MSE, and RMSE values are lower, indicating more accurate predictions. Additionally, the R^2 value of 0.97 is higher, suggesting that the Random Forest model explains more of the variance in the data compared to the Decision Tree, which had an R^2 of 0.94. Overall, the Random Forest Regressor provides more reliable and precise predictions.

Here we have the residual plot:



We can see that it is similar to the one before. Nonetheless, it can be considered as valid.

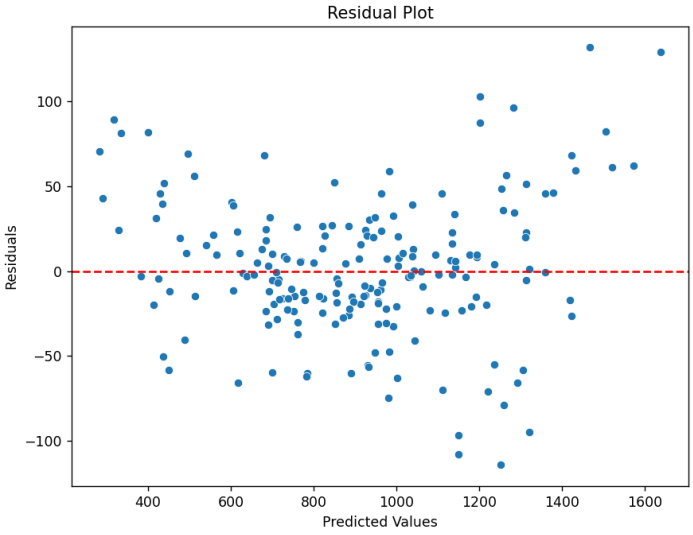
- Ridge Regression

Ridge regression is a type of linear regression that addresses multicollinearity by adding a penalty term to the loss function. This penalty term, controlled by the regularization parameter (alpha), discourages large coefficients, helping to prevent overfitting. Ridge regression is particularly useful when there are many correlated predictors in the data, as it helps to stabilize the model and improve its generalization to new data.

Here we have some metrics that have evaluated our Ridge regression model predicting Calories_Burned:

```
Mean Absolute Error (MAE): 30.28
Mean Squared Error (MSE): 1645.54
Root Mean Squared Error (RMSE): 40.57
R² (Coefficient of Determination): 0.98
```

The metrics suggest that the model performs well. The MAE of 30.28 indicates small average errors, while the MSE and RMSE values of 1645.54 and 40.57 respectively show that the model's predictions are close to the actual values. The R^2 score of 0.98 indicates that the model explains 98% of the variance in the target variable, suggesting a strong fit. Overall, these metrics reflect a good model performance.



We can see that residuals are more or less randomly distributed among all the plot. This means that our model fit is valid.

- Lasso Regression

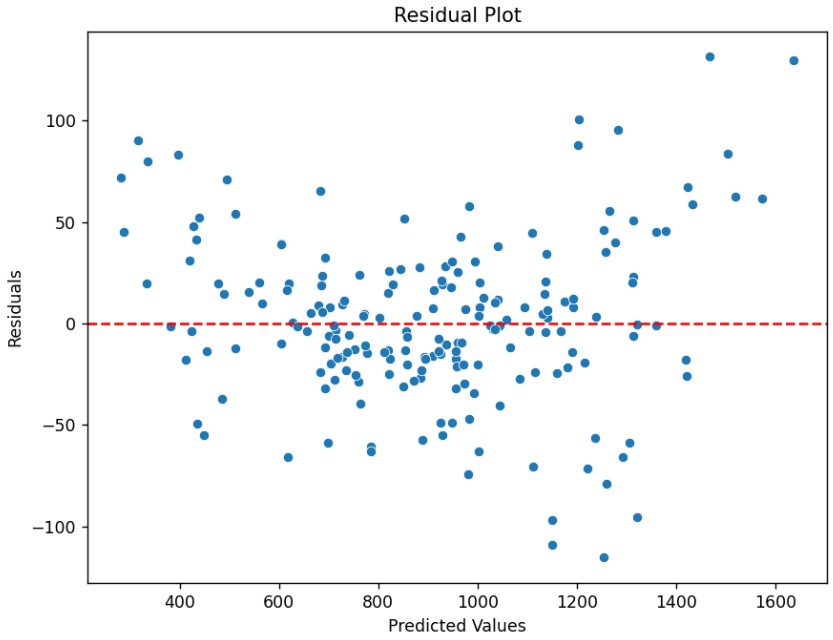
Lasso regression is a type of linear regression that uses L1 regularization, which adds a penalty based on the absolute values of the coefficients. This can shrink some coefficients to zero, effectively performing feature selection. In contrast, Ridge regression uses L2 regularization, which penalizes the squared values of the coefficients but does not eliminate any features. While both methods aim to reduce model complexity and prevent overfitting, Lasso can result in a sparse model by removing irrelevant features, whereas Ridge retains all features with smaller coefficients.

Here we have some metrics that have evaluated our Lasso regression model predicting Calories_Burned:

```
Mean Absolute Error (MAE): 30.20
Mean Squared Error (MSE): 1646.00
Root Mean Squared Error (RMSE): 40.57
R² (Coefficient of Determination): 0.98
```

In this case, metrics from our Lasso model are really similar to the ones from our ridge model. We can make the same conclussion.

Also, the residual plot is very similar:



Comparison

Polynomial regression performs the best, with the highest R^2 and lowest error metrics, effectively capturing non-linear patterns in the data. Linear regression also performs well but slightly lags behind polynomial regression. Support Vector Regression (SVR) performs poorly, showing a very low R^2 and high errors, making it unsuitable for this dataset. The Decision Tree Regressor has a lower R^2 and higher errors, indicating overfitting. Random Forest Regressor balances bias and variance well, offering good performance. Ridge and Lasso regressions are similar to linear regression, with regularization helping but not significantly improving results. Overall, Polynomial Regression and Random Forest are the top performers, while SVR and Decision Tree require further refinement.

4. Conclusion

In both classification and regression models, the performance varies depending on the model type and the data. For classification, K-Nearest Neighbors (KNN), Decision Tree, and Support Vector Machine (SVM) all achieved similar results, with Naive Bayes and Random Forest being the top performers. Both models demonstrated strong prediction accuracy, with Naive Bayes and Random Forest scoring 0.97 for class 1 and 2, and a perfect 1.00 for class 3 in the ROC curve and AUC scores.

In the case of regression models, Polynomial Regression emerged as the best performer, showing the highest R^2 and the lowest error metrics, followed closely by Linear Regression. Random Forest Regressor also showed strong performance, striking a balance between bias and variance. However, Support Vector Regression (SVR) and Decision Tree Regressor lagged behind, with SVR performing poorly and Decision Trees showing signs of overfitting. Ridge and Lasso regressions were comparable to Linear Regression, providing regularization but not significant improvements.

Overall, Random Forest and Naive Bayes stood out in classification, while Polynomial Regression and Random Forest excelled in regression tasks, making them the most reliable models for both types of problems in this dataset.

Tuning hyperparameters through techniques like Grid Search is essential for improving model performance. Hyperparameters, such as the learning rate, regularization strength, and the number of neighbors in KNN, can significantly impact a model's accuracy and generalization ability. Grid Search helps find the optimal combination of these parameters by exhaustively searching over a specified parameter grid.

Cross-validation is crucial because it allows for a more reliable assessment of model performance by splitting the data into multiple subsets, training the model on some subsets, and testing on others. This reduces the risk of overfitting and ensures that the model generalizes well to unseen data. Together, hyperparameter tuning and cross-validation help to enhance model accuracy, stability, and reliability.

Through the evaluation metrics, we observed that models like Random Forest and Naive Bayes showed consistent performance with high accuracy, which was reflected in their AUC scores and confusion matrix results. However, the Support Vector Machine (SVM) model, despite its good performance in classification, had slightly slower prediction times compared to others.

In regression, Polynomial Regression outperformed other models with the lowest MAE and MSE, indicating it captured the data trends more effectively. On the other hand, the Support Vector Regression (SVR) model underperformed with a low R^2 score, suggesting it struggled to fit the data well. A challenge during the comparison was balancing model complexity and performance, as more complex models sometimes led to overfitting, especially in the case of decision trees.