COP528 - Applied Machine Learning Coursework

Task 1:

I. Introduction

Maternal health refers to the health of women from the pregnancy until the postnatal period. Many organizations are concerned about this so the goal is to reduce the maternal and child mortality.

The purpose of this task is that with the given dataset we will create a machine learning algorithm which it will predict the stage of risk of health of a woman in the future. This will be succeeded with the use of multiple classification methods.

.

II. DATA AND PRELIMINARY ANALYSIS

The dataset contains information of different health exams as well as the Age. The csv file contains 1014 data and 6 attributes. First of all, we check if there were any missing values but we didn't have any. Then, we check the type and name of the columns. The target "RiskLevel" was nominal, so we encoded it from (low risk, mid risk, high risk) to (1,2,3). Furthermore, due to the reason that our data were in different scales we Normalize them it will skewed between [0,1].

After we check the dataset, we started doing the preliminary analysis. With this we can understand better how to build the model and of course what data to drop (if we are going to drop any attributes.

For the feature selection we create a pair plot as well as a heatmap to see the correlation of all the attributes between them against the target. Sometimes the more attributes we have the more complex the model will be and that is but because if two attributes are correlated, it will take more time and the results they won't be that good. The results of the heatmap shown in Fig.1 are pretty good and the correlation of all attributes show that they are not correlated between them. The only two attributes that they have quite "high" correlation are SystolicBP and DiastolicBP, but it is not that high to drop one of them, so we keep them all.

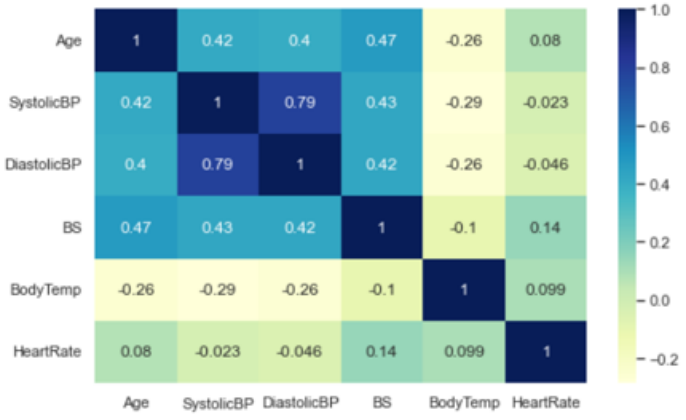


Figure 1: Relation between attributes

III. METHODS

The methods is the machine learning pipeline showing below in Fig.2. These steps are the way from where we start process the data until we evaluate the models.

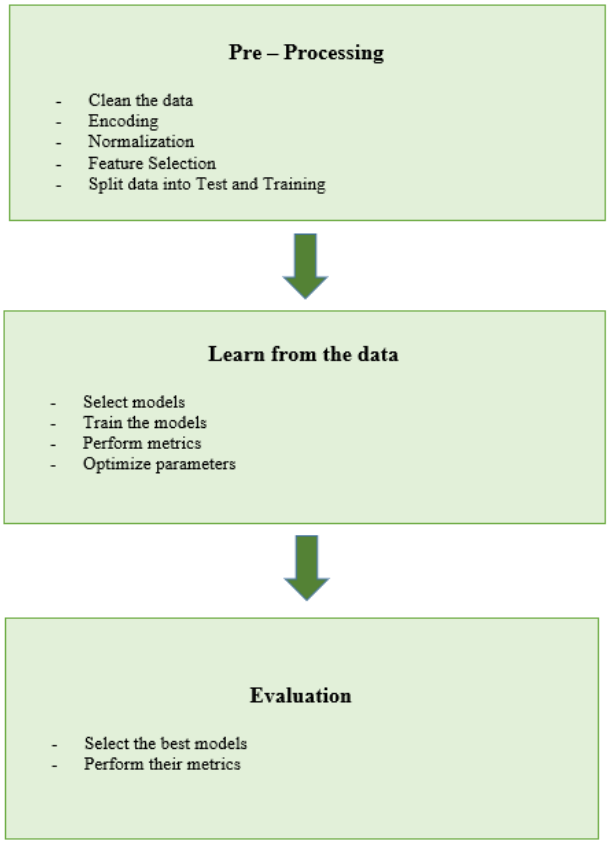


Figure 2: Method - Machine Learning Pipeline

1. *Pre - Processing*

This is the first step of the Machine Learning methods. In the pre-process as we mention in the preliminary analysis as well is when we clean our data, encoding the nominal values to numeric, select some features and split our data into test and training.

2. *Learn from the Data*

With the split of the data, we have the training set which our model is learning from it, it trains the model, so it will be able later to do the validation. Afterwards, we perform the metrics, changing different parameters until we finally find the optimal parameters in the models such as "Decision Tree".

3. *Evaluation*

Now we have done all the essential steps to calculate and find the optimal parameters, so we have the results to

see and select the best model. This will be succeeded with the use of multiple methods.

IV. EXPERIMENTS

In this stage we use different methods of classification to find the higher one after we improved each method with the optimal parameters. We used multiple methods but as you can see from Fig3 only few of them improved to the optimal giving us higher accuracy.

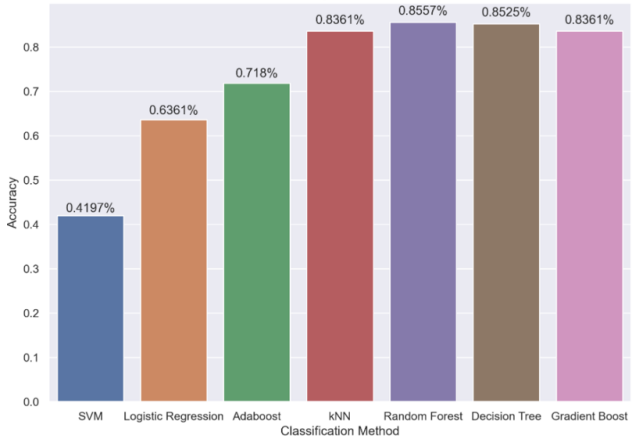


Figure 3: Accuracy of the Models after the optimization of the parameters

A. Classification Methods

1) Support Vector Machine (SVM)

SVM is a supervised learning method which can be used either for classification of regression. It draws a decision boundary which is hyperplane between the classes in order to classify them. It's purpose is to find the best line or decision boundary that can divide n-dimensional space into classes so that fresh data points can be readily placed in the correct category in the future [9].

At first, we created a loop with different kernels and of course the results were low as we expected due to the reason that SVM is for binomial classes and our data is multinomial.

2) AdaBoost

As SVM, here logistic regression is used commonly for binary classification tasks that is why the results are greater than SVM but not so high. We used a for loop to find the optimal n-estimators and that improved the model.

3) kNN

This method classifies the data point on how its neighbor is classified. It learns the class that assigned from the training dataset and is considering a predicted class for the new data points. More specifically, it finds the distance between a dot and the other data and it selects the k number of the most frequent label.

For this method we loop to find the optimal n-neighbors. We repeated the parameter loop for "weights = uniform and distance". The results for both were that the optimal k = 1.

4) Random Forest

Random Forest is a supervised method. It has the ability to utilized for both classification and regression. It is a classifier that contains several decision trees on a various subset on the data and then it takes the average to improve the predictive accuracy. It uses randomness when it's building the tree to try and create uncorrelated forest of trees whose prediction is more accurate than any other tree. Also, the more trees it has, the more robust is the forest [10].

The accuracy score of this method was high enough before any change of the parameters. We created a loop to find the best number of random states which was 7 and increases the model's accuracy.

5) Decision Tree

The way Decision Tree works is in the shape of a tree structure. It breaks down a dataset into smaller and smaller sections and then a decision tree is developed at the same time from them.

For optimization we loop through a various number of random states finding the optimal equals to 422 when the criterion = "gini". Afterwards, we did the same loop with the criterion = "entropy" finding the optimal now 50. The results in the second loop were slightly better.

6) Gradient Boost

This method is used for continuous and categorical target variables. It produces an additive predictive model by combining various weak predictors (decision trees).

At first, we try to loop just to find the optimal learning rate which it was 1. Then we attempt to find with the use of for loop again the optimal n-estimators. With this both tunings we manage to reach higher accuracy compared to when we did the method first.

B. Comparison of the Models

From all 7 methods we used, only 4 of them were improved reaching an optimal stage in the end and we show the results on the Table I below

TABLE I - COMPARISON OF THE MODELS

Performance Of Metrics	kNN	Random Forest	Decision Tree	Gradient Boost
Accuracy	82.62 %	85.57 %	85.24 %	83.60 %
Precision	82.58 %	82.58 %	82.58 %	82.58 %
F1 - score	82.51 %	82.51 %	82.51 %	82.51 %
Rank	4	1	2	3

From the Table I the best model was Random Forest. As for the Precision and F1 score the weighted average was very close to each method which cannot be seemed here. Based on the results the tree methods worked better than kNN. This is probably because we didn't have a lot of attributes.

C. Evaluation

To evaluate the performance of the models and to compare them we used the average of metrics. Although the precision and F1 scores were the same as we can see in Table I, the accuracy was a little bit better in Random Forest thing that makes it better.

Precision scores are the numbers of positive class predictions that belong in the positive class. More specific it states how many correct instances are of all the true predictions that have been made. In here both precision and F1 are high here so that is good because it means that not so many women who their risk is “low risk” are seemed like “high risk”.

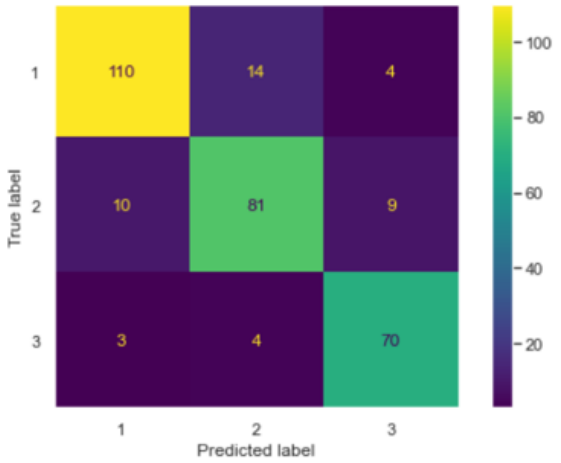


Figure 4: Confusion Matrix of Random Forest

Moreover, as we can see on Fig.4 on the Confusion Matrix of Random Forest, we can notice that only 14 women misclassified as mid-risk and 4 as high-risk rather than low-risk.

V. REFLECTION

Every step has been done and from the raw data we clean them, train our models and in the end reach the optimize for all of them. We succeed a satisfied amount of accuracy at 85.57 % in Random Forest which is pretty good. This model can be used now to predict the Risk of the maternal health.

An important problem that we faced was that every different methods we experiment the Accuracy score reached its maximum figure of around 85 %. This may be cause of the small number of attributes. That wasn’t bad and after some checks of the relations none of them were correlated with another one. Although, if we have had more attributes that they were more important we could have drop some of those ones that weren’t so much important. Also, the number of instances weren’t so large, and it is possible that with a grater number of instances a more accurate the results will be.

Another observation we have made through the experiments was that on the confusion matrices the misclassified results weren’t that bad but with a better model we could have reduce them.

It is very impressive the fact that both precision and F1 scores were so high in almost every experiment. We expected the results of the accuracy to be a little bit higher. Maybe one more thing that maybe could help the model to be better will be in the feature selection to remove an attribute although we don’t have so many attributes so on the other hand maybe this will not help. Overall, the results were good and the model can be used.

Task 2:

I. INTRODUCTION

We are given an image dataset with 10 different classes. Aim of this task is to try and create a Machine Learning model which is going to identify the image and classify it correctly.

In this task we are going to use a Deep Learning algorithm called Convolutional Neural Network (CNN) for image classification [11]. It is processing data that they have a pattern like images in our case and then it is designed to learn some hierarchies of the features by itself and adapting them from low to high level hierarchies [3].

From the given CNN model after some developments and reducing the overfitting we ended up having validation accuracy 77.62%.

II. DATA AND PRELIMINARY ANALYSIS

From the dataset we have 13,394 images which are formed in training and validation datasets. From each of the datasets we have 10 folders which they are the 10 different classes as follow: 'tench', 'English_springer', 'cassette_player', 'chain_saw', 'church', 'French_horn', 'garbage_truck', 'gas_pump', 'golf_ball' and 'parachute'. For the train data we have 9,469 images and for the validation data 3,925.

In the Preliminary Analysis first the model defined some parameters for the batch size as well as the images height and width. Then it renames the 10 classes properly. Afterwards as shown in it plotted a sample of images and "tench" shown 3 times. Then using a for loop provided we can see the image and labels shape.

III. METHODS

As we can see from the Fig.5 the structure of the Task 2 with the CNN is similar, like the one in Task 1. Below we will explain the steps in more detail.

A. Pre – Processing

After we uploaded the dataset the first step was to set the shuffle of the data equals to true. This is because we want to randomly shuffles the elements of our dataset. The data were spitted into 70% of Training and 30% of Validation.

The next step of the Pre-Processing is to configure the dataset. This is to keep in the memory the images after they loaded for the first epoch and to overlap the data processing while it is being trained.

Then a standardization is being made to the data. A normalization rescales the images to [0,1].

B. Model Design

In this stage the creation of the model is being made. This will be succeeded with the used of multiple layers and compiling the model.

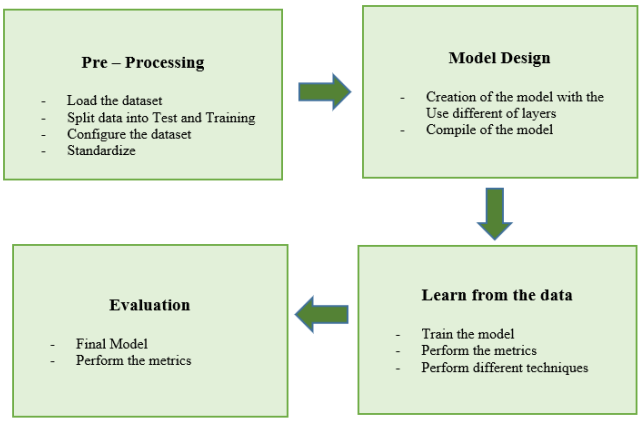


Figure 5: Method - Machine Learning Pipeline

To start with, in the creation of the model they have been used couple of layers that will be mentioned below.

Rescaling Layer is a preprocessing layer which it rescales the input values to a new range. In here the input was 1. /255 to rescale it to [0,1].

The simple application of a filter to an input that results in activation is known as a convolutional layer. The location and strength of a detected feature in an input, like as an image, are shown by a map of activations termed a feature map, which is created by repeatedly applying the same filter to an input. Under the restrictions of a given predictive modelling problem, such as image classification, it can automatically learn many filters in parallel specific to a training dataset. As a result, extremely precise traits appear on input photographs that can be identified everywhere [4]. In this task is used a common combination of 16 kernel numbers with kernel size = 3.

Pooling layers are used to progressively reduce the dimensions of the feature maps. What it does is that it reduces the number of parameters to learn. So, max pooling is an operation that select the maximum element from the region of the feature map used by that filter. After the use of this filter the output will contain only the most prominent features [5].

As for the compile of the model, it defines the loss function, the optimizer, and the metrics. For the optimizer it used "Adam" which is very good because it has faster computation time, and it requires fewer parameters for tuning. For the metrics it uses the accuracy so it will calculate how often the prediction will be equal to the labels.

C. Learn from the data

After the design of the model, we are trying a few things which will help us understand and learn from the model. The first thing is to train and run the model as the CNN model is build to. It is set to calculate the accuracy of the epochs 10 times. Every epoch from 1-10 we are going to have the accuracy and the loss so we are going to have 10 results of metrics.

D. EVALUATION

After the calculation of different metric results, we will be able to see the optimal. Then finally the CNN model will be able to run and provide the optimal measurements.

IV.EXPERIMENT

A. Pre – Processing

Starting from the beginning we imported the data, shuffle them with the dimensions of the photos being 180 x 180 as it existed before in the CNN model. As for the batch size it left as it was also before 32. After that we normalize the data and now, they are ready for the experiments.

B. Methods

The CNN model had from the beginning some layers which some of them were the Rescaling, Pooling, Convolutional and Flatten.

The number of epochs were 10 for the model. We plotted both Accuracy and Loss for training and validation. Frist of all, as we can see from the Fig. 6, the accuracy of the training set was increased with each epoch as well as the validation accuracy. The problem was the big gap between training and validation set which it means that there is overfitting to the model.

Overfitting occurs when the accuracy of our training dataset, the dataset used to train the model, is greater than our validation accuracy. In terms of 'loss', overfitting reveals itself when your model has a low error in the training set and a higher error in the testing set [7].

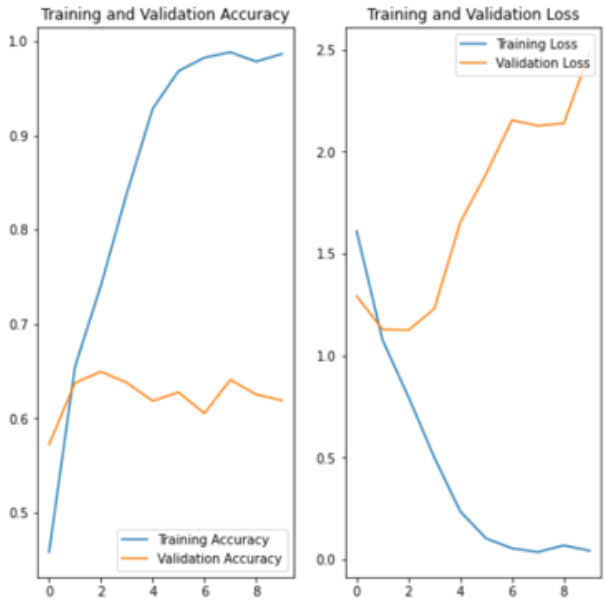


Figure 6: Base CNN Model Acuraccy and Loss

One solution for overfitting is Dropout. It is a regularization technique for reducing overfitting in artificial neural networks by preventing complex co-adaptations on training data. It is an efficient way of performing model CNN as we do now.

Data Augmentation is another way to improve the performance of the underlying CNN model. It is a strategy for increasing the quantity of data available by adding slightly changed copies of current data or creating new synthetic data from existing data. When training a machine learning model, it functions as a regularizer (similar to Dropout) and helps minimize overfitting. More precisely, in our model, it rotates and zooms the images, allowing the model to see the data more clearly.

C. Evaluation

In the evaluation stage we are going to discuss the results. As we can see from both the Table II and the Fig.7 it provides the Accuracy and Loss for both Training and Validation sets.

TABLE II - COMPARISON OF THE MODELS

Performance of Metrics		Base model	Optimal model
Maximum Accuracy	Training	0.9865	0.7762
	Validation	0.6191	0.6712
Minimum Loss	Training	0.0437	0.6887
	Validation	2.4884	1.0300

To start with, for the Base CNN model, the accuracy score of the Validation set is 0.61 whereas on the Optimal is increased to 0.67 on the Optimal CNN model. Thus, we can now say comparing the results the decrease of the overfitting found on the base model gave us better results. On the other hand we have the loss which improved. For the Base CNN was 2.48 where it decreased almost 1.45 on the Optimal model. Both the reduce of overfitting and loss c can be seen on the Fig.6 and Fig 7 as well.

V. REFLECTION

The steps that have been used for the image processing CNN model were very good. Furthermore the

While it's clear that this isn't the best accuracy and that there's still need for progress, although the total accuracy is still quite good. This CNN model didn't have many layers which it made it more clear and easiest to run. On the other hand, it is obvious that the more layers the model have the better the accuracy will be. This is because the calculations for the model to understand the image will be more so it will be more successful to find the right results.

It is obvious that the more layers a model has and the more complex the layers are, the more distinguishing features it will be able to extract, and hence the model will achieve a better accuracy than this.

The next time, to improve the CNN model I would add more layers and of course I will choose the complex ones because this will provide me with more accurate results. Nevertheless, overall, the results were good enough.

Figure 7: Optimal CNN model Accuracy and Loss

REFERENCES

[1] Dr. H. Fang. Applied Machine Learning [Notes]. Available: https://learn.lboro.ac.uk/course/view.php?id=21420

[2] ROC for multiclass classification. Code Copied from [https://stackoverflow.com/questions/45332410/roc-for-multiclass-classification] [Accessed: 17/03/2022]

[3] Convolutional neural networks: an overview and application in radiology. Available:https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9#:~:text=1),%2D%20to%20high%2Dlevel%20patterns [Accessed: 19/03/2022]

[4] How Do Convolutional Layers work in Deep Leaning NN. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ [Accessed: 19/03/2022]

[5] CNN Introduction to Pooling Layer. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/#:~:text=Pooling%20layers%20are%20used%20to,generated%20by%20a%20convolution%20layer [Accessed: 19/03/2022]

[6] How to treat overfitting in Convolutional Neural Networks Available: https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/ [Acessed: 19/03/2022]

[7] Wikipedia. Delution (neural networks). Available: https://en.wikipedia.org/wiki/Dilution_(neural_networks) [Acessed: 19/03/2022]

[8] Wikipedia. Data Augmentation. Available: https://en.wikipedia.org/wiki/Data_augmentation [Acessed: 19/03/2022]

[9] Support Vector Machine Algorith. Available: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm [Acessed: 19/03/2022]

[10] Understanding Random Forest Classifiers. Available: https://www.datacamp.com/community/tutorials/random-forests-classifier-python [Acessed: 19/03/2022]

[11] Image Classification. Code Copied for Task 2. Available: https://www.tensorflow.org/tutorials/images/classification

Appendix:

Google Collab Links for both Task 1 and Task 2:

Task 1: https://colab.research.google.com/drive/1nA_lWwrx4sTN9B0x8v7rIp4u2JSexzsx

Task 2: https://colab.research.google.com/drive/1tekjJQrKicMabnI6bAKvKvXryFsEyd6j

If any of the links above doesn't work, the code for both tasks is attached below and separately as individual ipynb files.

In [ ]:
```
COURSEWORK AML - TASK 1
```

In [196…
```
# Import Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn import preprocessing
import numpy as np
import seaborn as sns

from sklearn.metrics import confusion_matrix, classification_report, plot_confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.svm import SVC
from sklearn import svm,datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_curve, auc
```

A) Preprocessing

In [196…
```
#Load the dataset
maternal = pd.read_csv('Maternal Health Risk Data Set.csv')

# Print the first 5 entries
maternal.head(5)
```

Out[196…

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk

In [196…
```
# See the number of observations and attributes
maternal.shape[:]
```

Out[196… (1014, 7)

In [197…
```
# Check if there is any missing value
maternal.isnull().sum()
maternal.isna().sum()
```

Out[197… Age 0

```
SystolicBP     0
DiastolicBP    0
BS             0
BodyTemp       0
HeartRate      0
RiskLevel      0
dtype: int64
```

In [197…

```
# Summarize the data
maternal.describe()
```

Out[197…

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate
count	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000
mean	29.871795	113.198225	76.460552	8.725986	98.665089	74.301775
std	13.474386	18.403913	13.885796	3.293532	1.371384	8.088702
min	10.000000	70.000000	49.000000	6.000000	98.000000	7.000000
25%	19.000000	100.000000	65.000000	6.900000	98.000000	70.000000
50%	26.000000	120.000000	80.000000	7.500000	98.000000	76.000000
75%	39.000000	120.000000	90.000000	8.000000	98.000000	80.000000
max	70.000000	160.000000	100.000000	19.000000	103.000000	90.000000

In [197…

```
# Check the data type of each attribute
maternal.dtypes
```

Out[197…

```
Age              int64
SystolicBP       int64
DiastolicBP      int64
BS             float64
BodyTemp       float64
HeartRate        int64
RiskLevel       object
dtype: object
```

In [197…

```
# See the name of columns
maternal.columns
```

Out[197…

```
Index(['Age', 'SystolicBP', 'DiastolicBP', 'BS', 'BodyTemp', 'HeartRate',
       'RiskLevel'],
      dtype='object')
```

In [197…

```
# Replace RiskLevel column values from nominal to categorical
maternal['RiskLevel'] = maternal['RiskLevel'].replace({'low risk': '1', 'mid risk': '2', '
maternal.head()
```

Out[197…

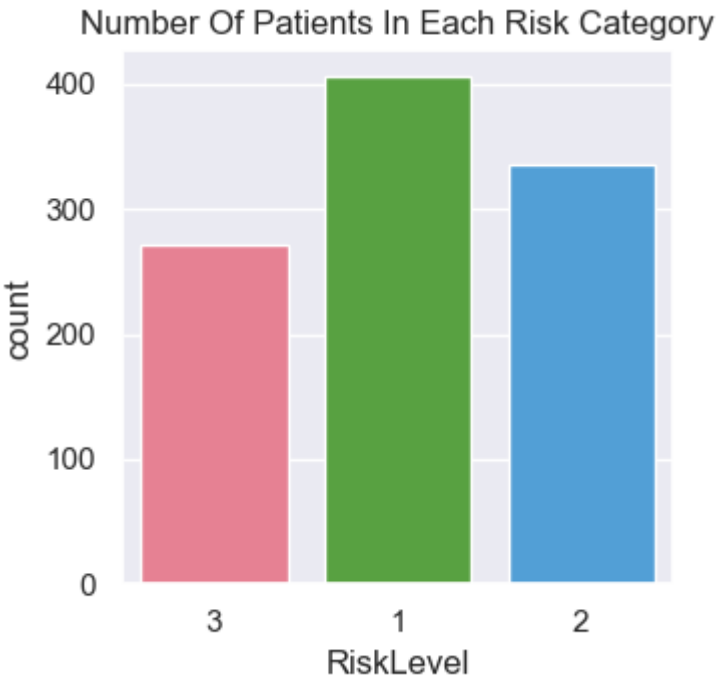
	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	3
1	35	140	90	13.0	98.0	70	3
2	29	90	70	8.0	100.0	80	3
3	30	140	85	7.0	98.0	70	3
4	35	120	60	6.1	98.0	76	1

B) Preliminary Analysis

In [197…

```
# Plot the number of patient in each category of Risk Level
plt.figure(figsize=(4,4), dpi=100)
sns.countplot(data=maternal, x="RiskLevel",palette = "husl").set(title="Number Of Patients
```

Out[197… [Text(0.5, 1.0, 'Number Of Patients In Each Risk Category')]



In [197…

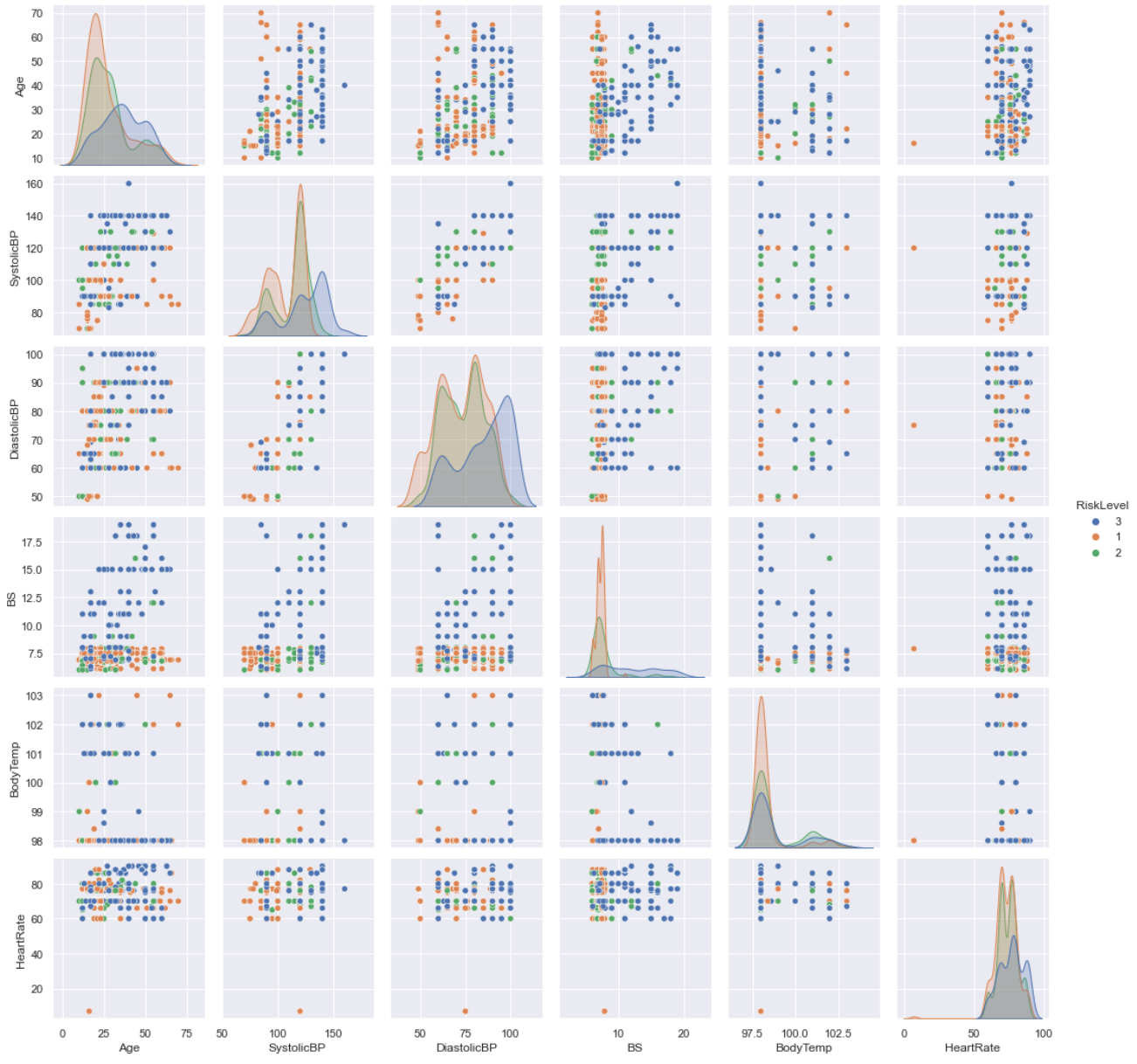
```
# Plot all the attributes with the target
fig, axes= plt.subplots(2, 3,figsize=(18, 10))

fig.suptitle('Relationship between attributes and the target value')

fig1 = sns.stripplot(ax=axes[0, 0],x="RiskLevel", y="Age", data=maternal,jitter = 0.30)
fig2 = sns.stripplot(ax=axes[0, 1],x="RiskLevel", y="SystolicBP", data=maternal,jitter = 0
fig3 = sns.stripplot(ax=axes[0, 2],x="RiskLevel", y="DiastolicBP", data=maternal,jitter =
fig4 = sns.stripplot(ax=axes[1, 0],x="RiskLevel", y="BS", data=maternal,jitter = 0.30)
fig5 = sns.stripplot(ax=axes[1, 1],x="RiskLevel", y="BodyTemp", data=maternal,jitter = 0.3
fig6 = sns.stripplot(ax=axes[1, 2],x="RiskLevel", y="HeartRate", data=maternal,jitter = 0.
```

In [197…

```
# Multiple comparison plots of all attributes with the target to see the correlation
sns.pairplot(maternal,vars=['Age', 'SystolicBP', 'DiastolicBP', 'BS', 'BodyTemp', 'HeartRa
```

Out[197…

```
<seaborn.axisgrid.PairGrid at 0x18f6f693af0>
```



In [197…

```
# This matrix to see if some features are correlated between them
sns.heatmap(maternal.corr(method ='pearson'),cmap="YlGnBu", annot=True)
```

Out[197…

```
<AxesSubplot:>
```

In [197…

```
# Create Train and Test data (Split them)
X = maternal.drop(['RiskLevel'], axis=1)
y = maternal.RiskLevel

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train.shape, X_test.shape

y_test
```

Out[197…

```
985    3
842    2
581    1
453    1
710    2
      ..
68     2
376    1
323    1
103    3
895    1
Name: RiskLevel, Length: 305, dtype: object
```

In [198…

```
# Scaling between [0,1]
```

```
scaler = MinMaxScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)

# # Standardization
# scaler = StandardScaler()
# scaled_X_train = scaler.fit_transform(X_train)
# scaled_X_test = scaler.transform(X_test)
```

In [198…

```
# Function for collecting the data of the clasification method to plot them

def plot_multiclass_roc(clf, scaled_X_test, y_test, n_classes, figsize=(3,3)):
    y_score = clf.decision_function(scaled_X_test)

    # Information
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # Calculate the dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Make roc curve for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Roc Curve')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for label %i' % (roc_auc[i
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()
```

C) Experiment

1) Support Vector Machine (SVM) Classification

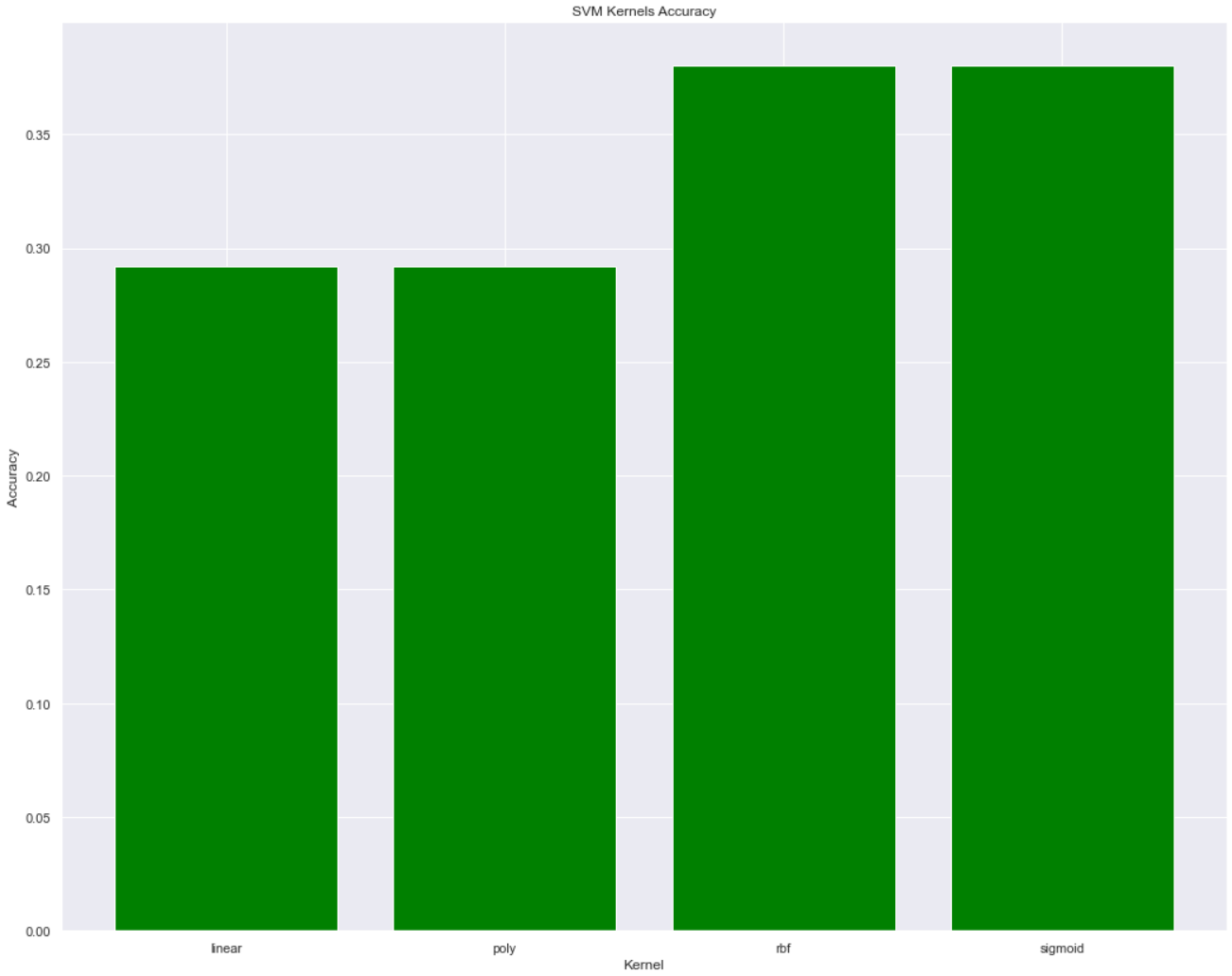
In [198…

```
# SVM classification

# create a list to save the accuracy scores
acc = []

# create a loop to check the accuracy scores of all the four kernels
for kernel in ['linear','poly','rbf', 'sigmoid']:
 model = svm.SVC(kernel=kernel, degree=3)
 clf_SVM = model.fit(scaled_X_train, y_train)
 y_pred_SVM = clf_SVM.predict(X_test)
 acc.append(accuracy_score(y_test, y_pred_SVM))

 print(f'The accuracy of SVM with {kernel} is:{accuracy_score(y_test, y_pred_SVM)}')
```

```
The accuracy of SVM with linear is:0.29180327868852457
The accuracy of SVM with poly is:0.29180327868852457
The accuracy of SVM with rbf is:0.380327868852459
The accuracy of SVM with sigmoid is:0.380327868852459
```

In [198…

```
# Plot the kernel's accuracy scores

ker = ['linear','poly','rbf', 'sigmoid']

plt.bar(ker, acc, color='green')
plt.xlabel("Kernel")
plt.ylabel(" Accuracy ")
plt.title("SVM Kernels Accuracy")
plt.show()
```



In [198…

```
# SVM try to find the optimal classification

clf_SVM = svm.SVC(kernel='sigmoid', degree=3, decision_function_shape = 'ovo', gamma = 'sc
clf_SVM.fit(scaled_X_train, y_train)
y_pred_SVM = clf_SVM.predict(X_test)

svm_acc = accuracy_score(y_test, y_pred_SVM)
print(f'The accuracy of SVM with poly is:{svm_acc}')

print(classification_report(y_test, y_pred_SVM, zero_division = 0))
```

```
The accuracy of SVM with poly is:0.380327868852459
              precision    recall  f1-score   support

           1       0.38      1.00      0.55       116
           2       0.00      0.00      0.00       100
           3       0.00      0.00      0.00        89

    accuracy                           0.38       305
   macro avg       0.13      0.33      0.18       305
weighted avg       0.14      0.38      0.21       305
```

2) AdaBoost

In [198…

```
# AdaBoost

ab = AdaBoostClassifier()
ab.fit(scaled_X_train, y_train)

y_pred_ab = ab.predict(scaled_X_test)

ab_acc = accuracy_score(y_test, y_pred_ab)
print('The accuracy of the test set is:',ab_acc)
```

```
The accuracy of the test set is: 0.6032786885245902
```

In [198…

```
# AdaBoost
# for loop from 1-500 find the optimal n-estimators


n_estimators = range(1,100)
max_acc = 0
list = []


for n in n_estimators:
    ab = AdaBoostClassifier(n_estimators = n)
    ab.fit(scaled_X_train, y_train)

    y_pred_ab = ab.predict(scaled_X_test)

    ab_acc = accuracy_score(y_test, y_pred_ab)

    list.append(ab_acc)

print('For n =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

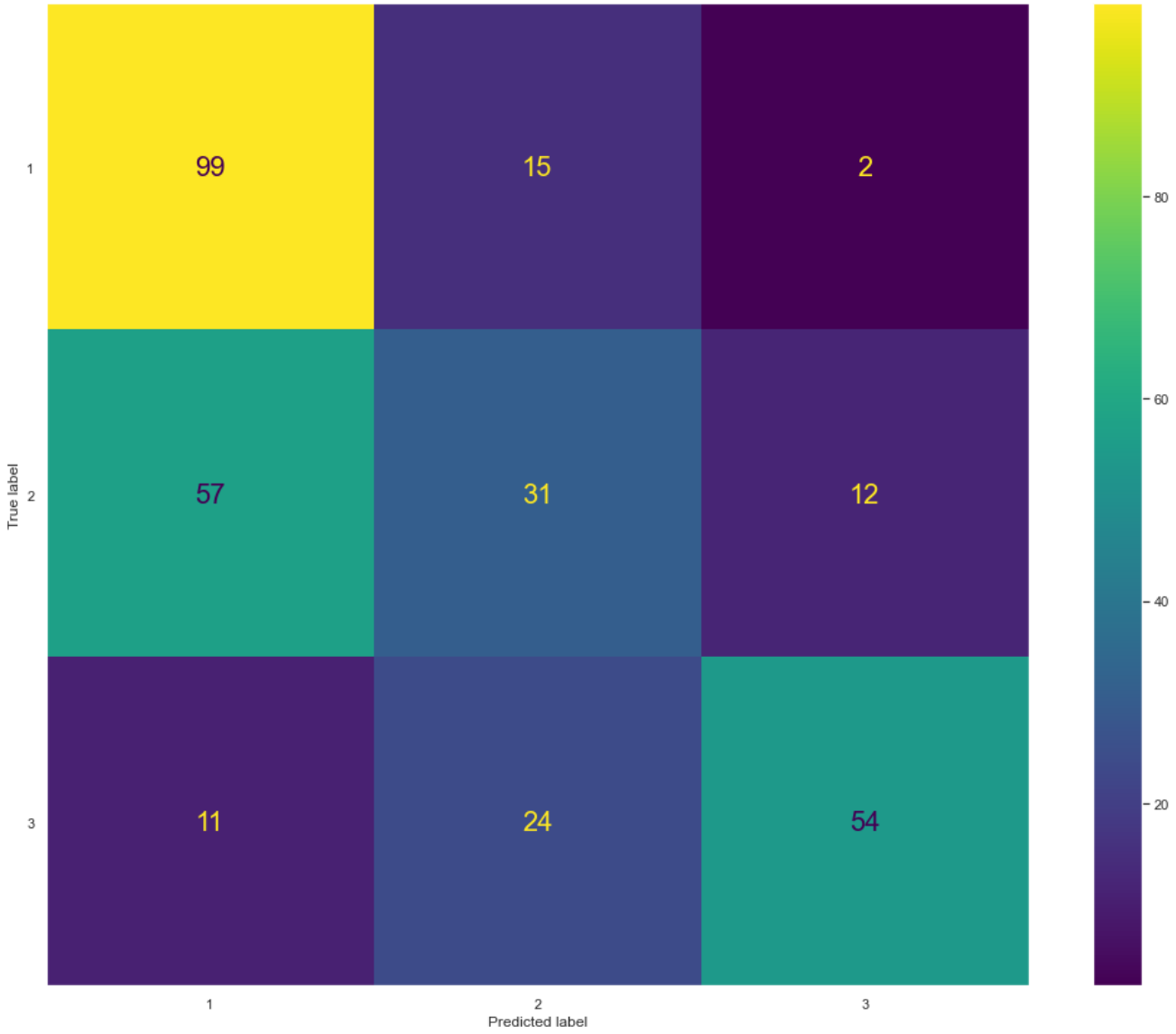
```
For n = 4 the accuracy of the test set is: 0.6327868852459017
```

In [198…

```
# AdaBoost - Optimal

ab = AdaBoostClassifier(n_estimators = 28)
ab.fit(scaled_X_train, y_train)

y_pred_ab = ab.predict(scaled_X_test)

ab_acc = accuracy_score(y_test, y_pred_ab)
print('The accuracy of the test set is:',ab_acc)

print(classification_report(y_test, y_pred_ab))

# Create a confusion matrix
print('The confusion matrix is:\n')
plot_confusion_matrix(ab, scaled_X_test, y_test)
plt.grid(False)
```

```
The accuracy of the test set is: 0.6032786885245902
              precision    recall  f1-score   support

           1       0.59      0.85      0.70       116
           2       0.44      0.31      0.36       100
           3       0.79      0.61      0.69        89

    accuracy                           0.60       305
   macro avg       0.61      0.59      0.58       305
weighted avg       0.60      0.60      0.59       305
```

The confusion matrix is:



In [ ]:

3) kNN

In [198…

```
# KNN classification

clf_knn = KNeighborsClassifier()
clf_knn.fit(scaled_X_train, y_train)
KNeighborsClassifier(n_neighbors=1)

# calculate the prediction of knn
y_pred_knn = clf_knn.predict(scaled_X_test)

# calculate the accuracy
knn_acc = accuracy_score(y_test, y_pred_knn)
print('The accuracy of the test set is:',knn_acc)
```

The accuracy of the test set is: 0.6655737704918033

In [198…

```
# kNN for loop to find the best n_neighbor (weights = "uniform")

# checks of 100 n_neighbors
n_neighbors = range(1,500)
# the maximum accuracy starts from 0
```

```
max_acc = 0
# create an empty list to save the accuracy scores
list = []

# Creating a for loop to find the optimal n_neighbors number with weights = "uniform"

for n in random_states:

  #Build a kNN model and train a test set
  classifier = KNeighborsClassifier(n_neighbors=n,weights = "uniform").fit(scaled_X_train,
  # predict
  y_pred_knn = classifier.predict(scaled_X_test)

  knn_acc = metrics.accuracy_score(y_test, y_pred_knn)

  # save to a list all the accuracy scores
  list.append(knn_acc)

# print the number of neighbors equals to the maximum accuracy score of the list
print('For n =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

```
For n = 1 the accuracy of the test set is: 0.8032786885245902
```

In [ ]:

In [199…

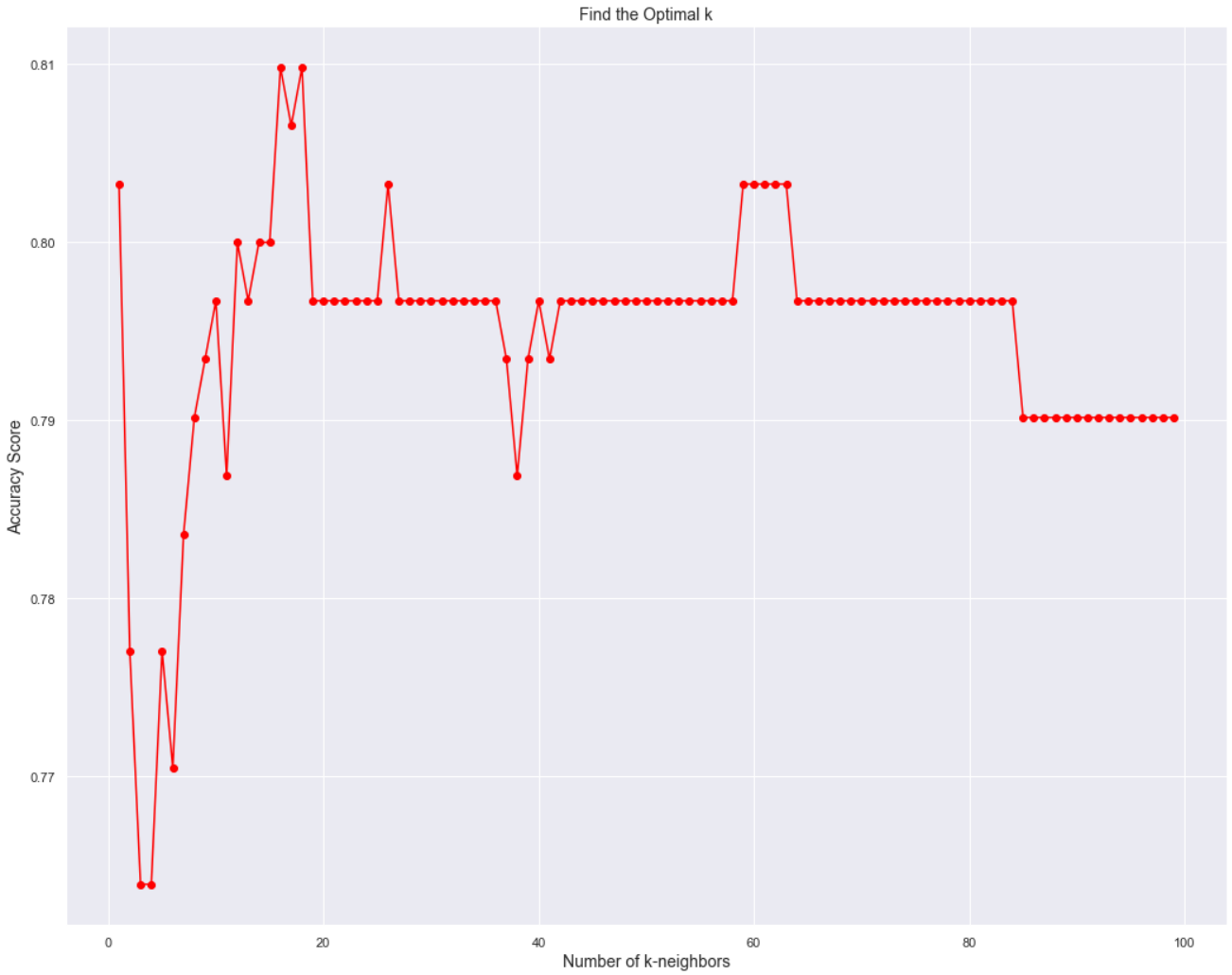
```
# kNN for loop to find the best n_neighbor (weights = "distance")

# checks of 100 n_neighbors
n_neighbors = range(1,500)
# the maximum accuracy starts from 0
max_acc = 0
# create an empty list to save the accuracy scores
list = []


# Creating a for loop to find the optimal n_neighbors number with weights = "distance"

for n in random_states:
  classifier = KNeighborsClassifier(n_neighbors=n,weights = "distance").fit(scaled_X_train
  y_pred_knn = classifier.predict(scaled_X_test)

  knn_acc = metrics.accuracy_score(y_test, y_pred_knn)
  list.append(knn_acc)

print('For n =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

```
For n = 16 the accuracy of the test set is: 0.8098360655737705
```

In [199…

```
# Search and Plot the optimal k

k_range = range(1, 100)
# create an empty list to save the accuracy scores
k_scores = []

# Create a loop to check multible accuracy scores based on k

for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors=k,weights = "distance").fit(scaled_X_tra
    y_pred_knn = classifier.predict(scaled_X_test)
    scores = metrics.accuracy_score(y_test, y_pred_knn)
    k_scores.append(scores)

# Plot accuracy scores of each n_neighbor
```

```
plt.plot(k_range, k_scores, color='red', marker='o')
plt.title('Find the Optimal k', fontsize=14)
plt.xlabel('Number of k-neighbors', fontsize=14)
plt.ylabel('Accuracy Score', fontsize=14)
plt.grid(True)
plt.show()
```

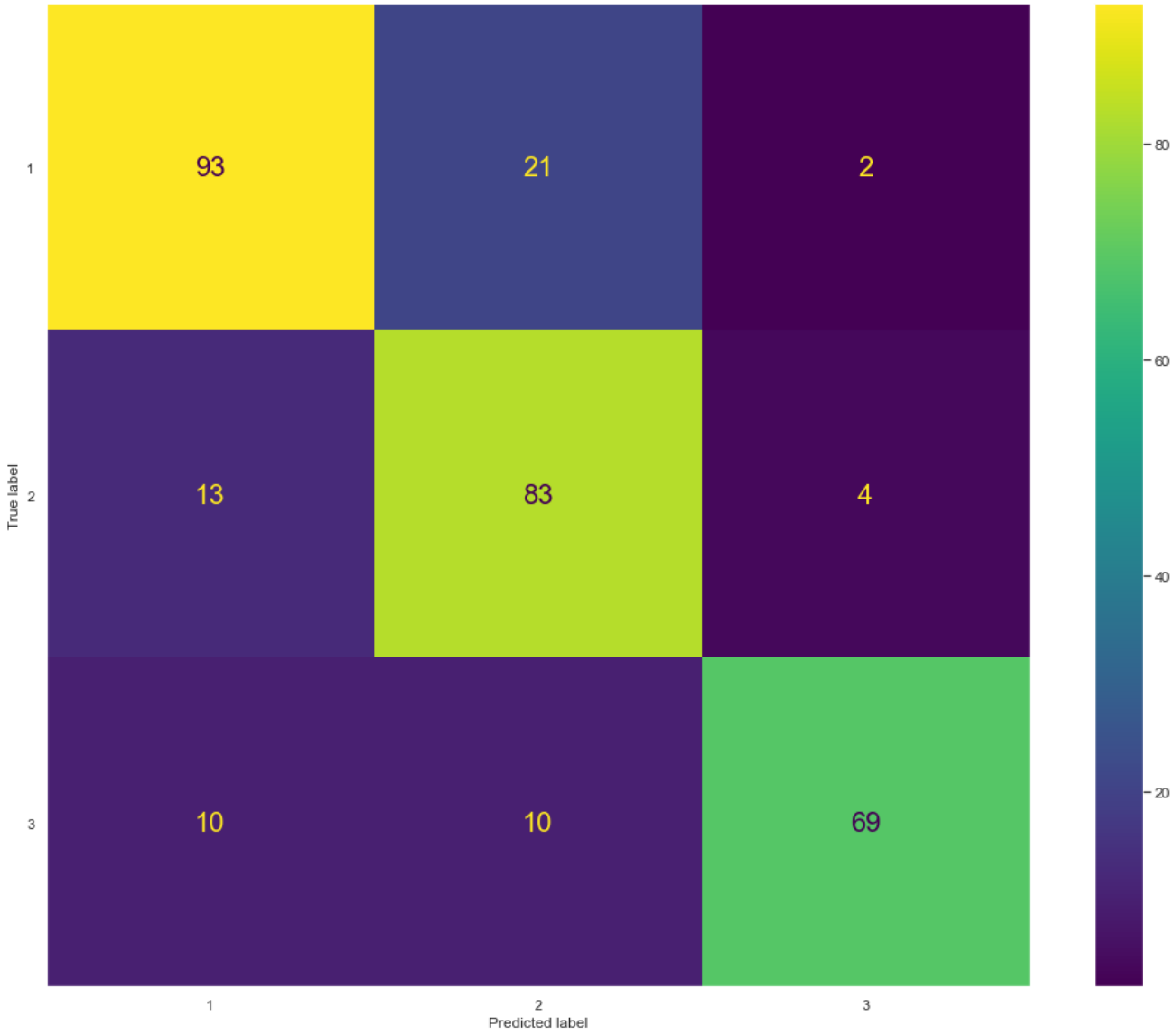


In [199…

```
# KNN classification - OPTIMAL

clf_knn = KNeighborsClassifier(n_neighbors=1,weights='distance')
clf_knn.fit(scaled_X_train, y_train)

y_pred = clf_knn.predict(scaled_X_test)


# Create a classification report
report_test = classification_report(y_test, y_pred)
print(f'The classification report for the training set is: \n {report_test}')

# Calculate the accuracy score
knn_acc = accuracy_score(y_test, y_pred)
print('The accuracy of the test set is:',knn_acc)

# Calculate the precision score
knn_prec = precision_score(y_test, y_pred, average = "weighted")
print('The precision of the test set is:',knn_prec)

# Calculate the f1 score
knn_f1 = f1_score(y_test, y_pred, average = "weighted")
print('The f1 of the test set is:',knn_f1)
```

```
# Create a confusion matrix
print('The confusion matrix is:\n')
plot_confusion_matrix(clf_knn, scaled_X_test, y_test)
plt.grid(False)
```

```
The classification report for the training set is:
              precision    recall  f1-score   support

           1       0.80      0.80      0.80       116
           2       0.73      0.83      0.78       100
           3       0.92      0.78      0.84        89

    accuracy                           0.80       305
   macro avg       0.82      0.80      0.81       305
weighted avg       0.81      0.80      0.80       305

The accuracy of the test set is: 0.8032786885245902
The precision of the test set is: 0.8120885821110153
The f1 of the test set is: 0.8047879913455178
The confusion matrix is:
```



4) Random Forest Classification

In [199…

```
# Random Forest

# Build a Random Forest classification model
rf = RandomForestClassifier()
rf.fit(scaled_X_train, y_train)
```

```
y_pred_rf = rf.predict(scaled_X_test)

# Calculate the accuracy score
rf_acc = accuracy_score(y_test, y_pred_rf)
print('The accuracy of the test set is:',rf_acc)

# Create a classification report
print(classification_report(y_test, y_pred_rf, zero_division = 0))
```

```
The accuracy of the test set is: 0.8032786885245902
              precision    recall  f1-score   support

           1       0.78      0.84      0.80       116
           2       0.75      0.79      0.77       100
           3       0.93      0.78      0.85        89

    accuracy                           0.80       305
   macro avg       0.82      0.80      0.81       305
weighted avg       0.81      0.80      0.80       305
```

In [199…

```
#  Random Forest
# for loop for random_state 1-50

random_states = range(1,51)
max_acc = 0
list = []

# Loop through the model to find the optimal random state

for r in random_states:
    rf = RandomForestClassifier(random_state = r)
    rf.fit(scaled_X_train, y_train)

    y_pred_rf = rf.predict(scaled_X_test)

    rf_acc = accuracy_score(y_test, y_pred_rf)

    list.append(rf_acc)

print('For r =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

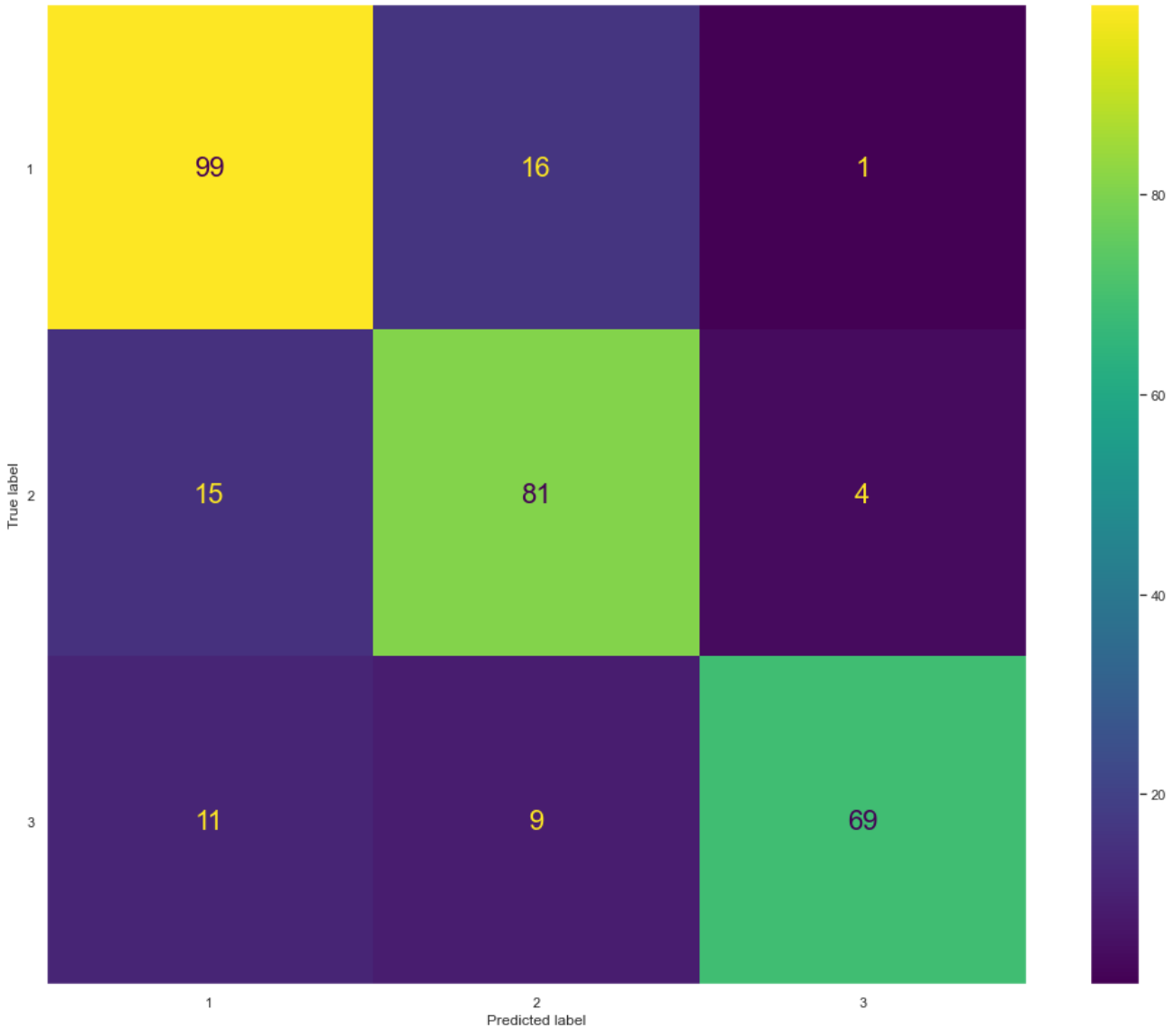
```
For r = 36 the accuracy of the test set is: 0.8229508196721311
```

In [199…

```
# Random Forest - OPTIMAL

rf = RandomForestClassifier(random_state = 7)
rf.fit(scaled_X_train, y_train)

y_pred_rf = rf.predict(scaled_X_test)

rf_acc = accuracy_score(y_test, y_pred_rf)
print('The accuracy of the test set is:',rf_acc)

# Calculate the precision score
rf_prec = precision_score(y_test, y_pred, average = "weighted")
print('The precision of the test set is:',rf_prec)

# Calculate the f1 score
rf_f1 = f1_score(y_test, y_pred, average = "weighted")
print('The f1 of the test set is:',rf_f1)

print(classification_report(y_test, y_pred_rf, zero_division = 0))

# Create a confusion matrix
```

```
print('The confusion matrix is:\n')
plot_confusion_matrix(rf, scaled_X_test, y_test)
plt.grid(False)
```

```
The accuracy of the test set is: 0.8163934426229508
The precision of the test set is: 0.8120885821110153
The f1 of the test set is: 0.8047879913455178
              precision    recall  f1-score   support

           1       0.79      0.85      0.82       116
           2       0.76      0.81      0.79       100
           3       0.93      0.78      0.85        89

    accuracy                           0.82       305
   macro avg       0.83      0.81      0.82       305
weighted avg       0.82      0.82      0.82       305

The confusion matrix is:
```



5) Decision Tree

In [199…

```
# Decision Tree

# Build a Decision Tree model
dt = tree.DecisionTreeClassifier()
dt.fit(scaled_X_train, y_train)

y_pred_dt = dt.predict(scaled_X_test)
```

```
dt_acc = accuracy_score(y_test, y_pred_dt)
print('The accuracy of the test set is:',dt_acc)
```

```
The accuracy of the test set is: 0.7737704918032787
```

In [199…]

```
# Decision Tree
# for loop for random_state 1-500 and criterion='gini'

random_states = range(1,501)
list = []

# Build a for loop to find the optimal random state number with criterion='gini'

for j in random_states:
  dt = tree.DecisionTreeClassifier(criterion='gini', max_features = 'auto',
                                   random_state = j,splitter = 'best')
  dt.fit(scaled_X_train, y_train)

  y_pred_dt = dt.predict(scaled_X_test)

  dt_acc = accuracy_score(y_test, y_pred_dt)
  list.append(dt_acc)

print('For j =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

```
For j = 390 the accuracy of the test set is: 0.8229508196721311
```

In [199…]

```
# Decision Tree
# for loop for random_state 1-500 and criterion='entropy'

random_states = range(1,501)
list = []

# Build a for loop to find the optimal random state number with criterion='entropy'

for j in random_states:
  dt = tree.DecisionTreeClassifier(criterion='entropy', max_features = 'auto',
                                   random_state = j,splitter = 'best')
  dt.fit(scaled_X_train, y_train)

  y_pred_dt = dt.predict(scaled_X_test)

  dt_acc = accuracy_score(y_test, y_pred_dt)
  list.append(dt_acc)

print('For j =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

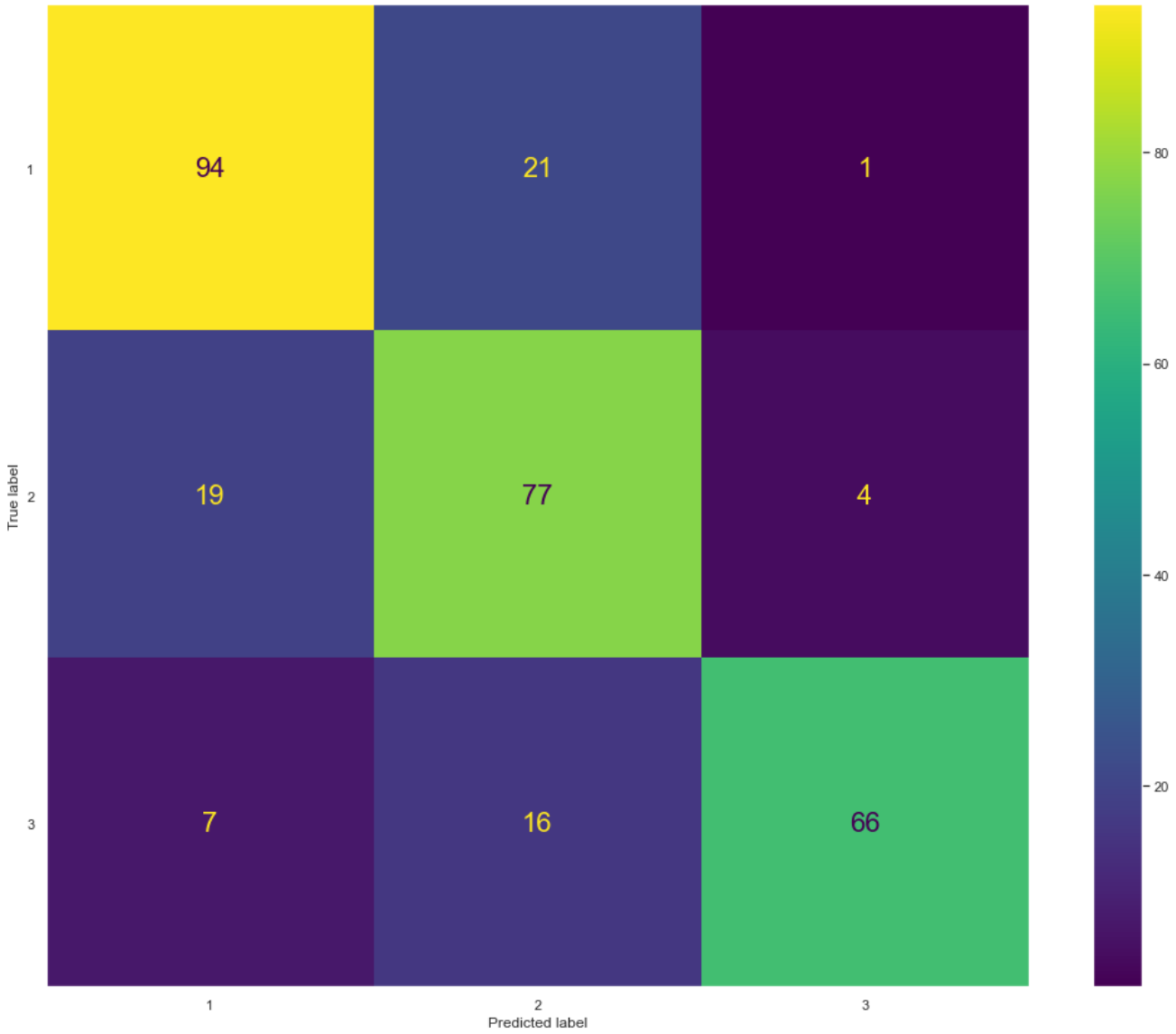
```
For j = 354 the accuracy of the test set is: 0.8229508196721311
```

In [199…]

```
# Decision Tree - OPTIMAL

dt = tree.DecisionTreeClassifier(criterion='entropy', max_features = 'auto',
                                   random_state = 50,splitter = 'best')
dt.fit(scaled_X_train, y_train)

y_pred_dt = dt.predict(scaled_X_test)

dt_acc = accuracy_score(y_test, y_pred_dt)


report_test = classification_report(y_test, y_pred_dt)
print(f'The classification report for the training set is: \n {report_test}')
```

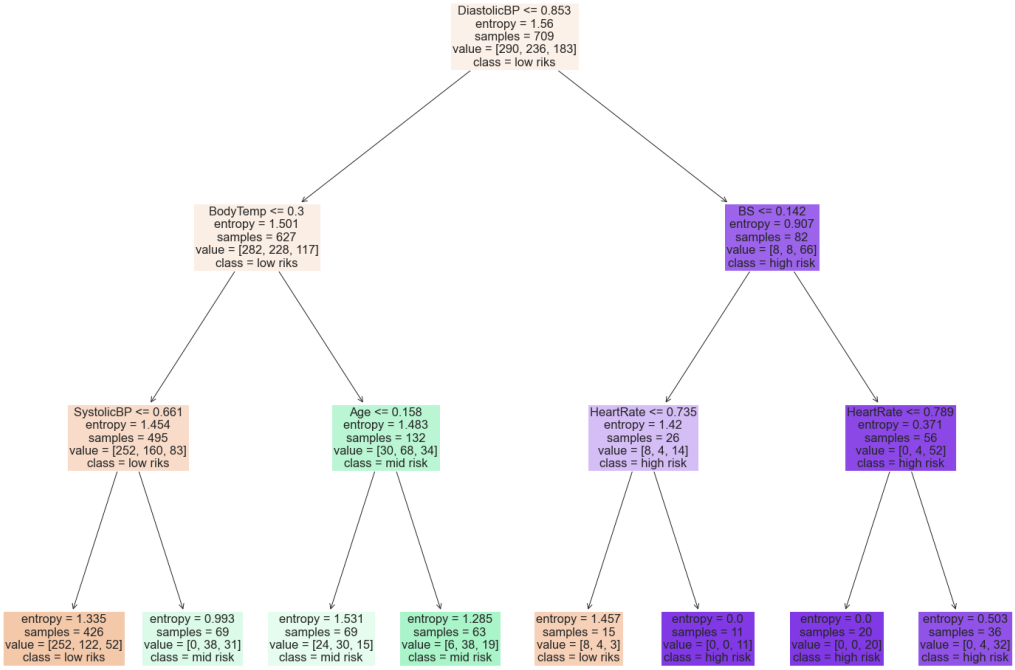
```
dt_acc = accuracy_score(y_test, y_pred_dt)
print('The accuracy of the test set is:',dt_acc)

dt_prec = precision_score(y_test, y_pred, average = "weighted")
print('The precision of the test set is:',dt_prec)

dt_f1 = f1_score(y_test, y_pred, average = "weighted")
print('The f1 of the test set is:',dt_f1)


# Create a confusion matrix
print('The confusion matrix is:\n')
plot_confusion_matrix(dt, scaled_X_test, y_test)
plt.grid(False)
```

```
The classification report for the training set is:
              precision    recall  f1-score   support

           1       0.78      0.81      0.80       116
           2       0.68      0.77      0.72       100
           3       0.93      0.74      0.82        89

    accuracy                           0.78       305
   macro avg       0.80      0.77      0.78       305
weighted avg       0.79      0.78      0.78       305

The accuracy of the test set is: 0.7770491803278688
The precision of the test set is: 0.8120885821110153
The f1 of the test set is: 0.8047879913455178
The confusion matrix is:
```



In [200…

```
# Decision Tree - Plot

# Build and plot the decision tree with maximum depth = 3
dt = tree.DecisionTreeClassifier(criterion='entropy', max_features = 'auto',
                                 random_state = 50,splitter = 'best',max_depth=3)
dt.fit(scaled_X_train, y_train)
y_pred_dt = dt.predict(scaled_X_test)
dt_acc = accuracy_score(y_test, y_pred_dt)


fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(dt,
 feature_names=['Age', 'SystolicBP', 'DiastolicBP', 'BS', 'BodyTemp', 'HeartRate'],
 class_names=['low riks', 'mid risk', 'high risk'],
 filled=True)
```



6) Gradient Boost

In [200…

```
# Gradient Boost

gb = GradientBoostingClassifier()
gb.fit(scaled_X_train, y_train)

y_pred_gb = gb.predict(scaled_X_test)

gb_acc = accuracy_score(y_test, y_pred_gb)
print('The accuracy of the test set is:',gb_acc)
```

```
The accuracy of the test set is: 0.7573770491803279
```

In [200…

```
# for loop from 1-500 find the optimal n-estimators


learning_rate = range(1,100)
max_acc = 0
list = []

# Create a loop to find the optimal learning rate

for n in n_estimators:
    gb = GradientBoostingClassifier(learning_rate = n )
    gb.fit(scaled_X_train, y_train)

    y_pred_gb = gb.predict(scaled_X_test)

    gb_acc = accuracy_score(y_test, y_pred_gb)
    list.append(gb_acc)
```

```
print('For n =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

```
For n = 1 the accuracy of the test set is: 0.8065573770491803
```

In [200…

```
# for loop from 1-500 find the optimal n-estimators


n_estimators = range(1,100)
max_acc = 0
list = []

# Create a loop to find the optimal n_estimators

for n in n_estimators:
    gb = GradientBoostingClassifier(learning_rate = 1 ,n_estimators = n)
    gb.fit(scaled_X_train, y_train)

    y_pred_gb = gb.predict(scaled_X_test)

    gb_acc = accuracy_score(y_test, y_pred_gb)
    list.append(gb_acc)

print('For n =',(list.index(max(list))+1),'the accuracy of the test set is:', max(list))
```

```
For n = 18 the accuracy of the test set is: 0.8131147540983606
```

In [ ]:

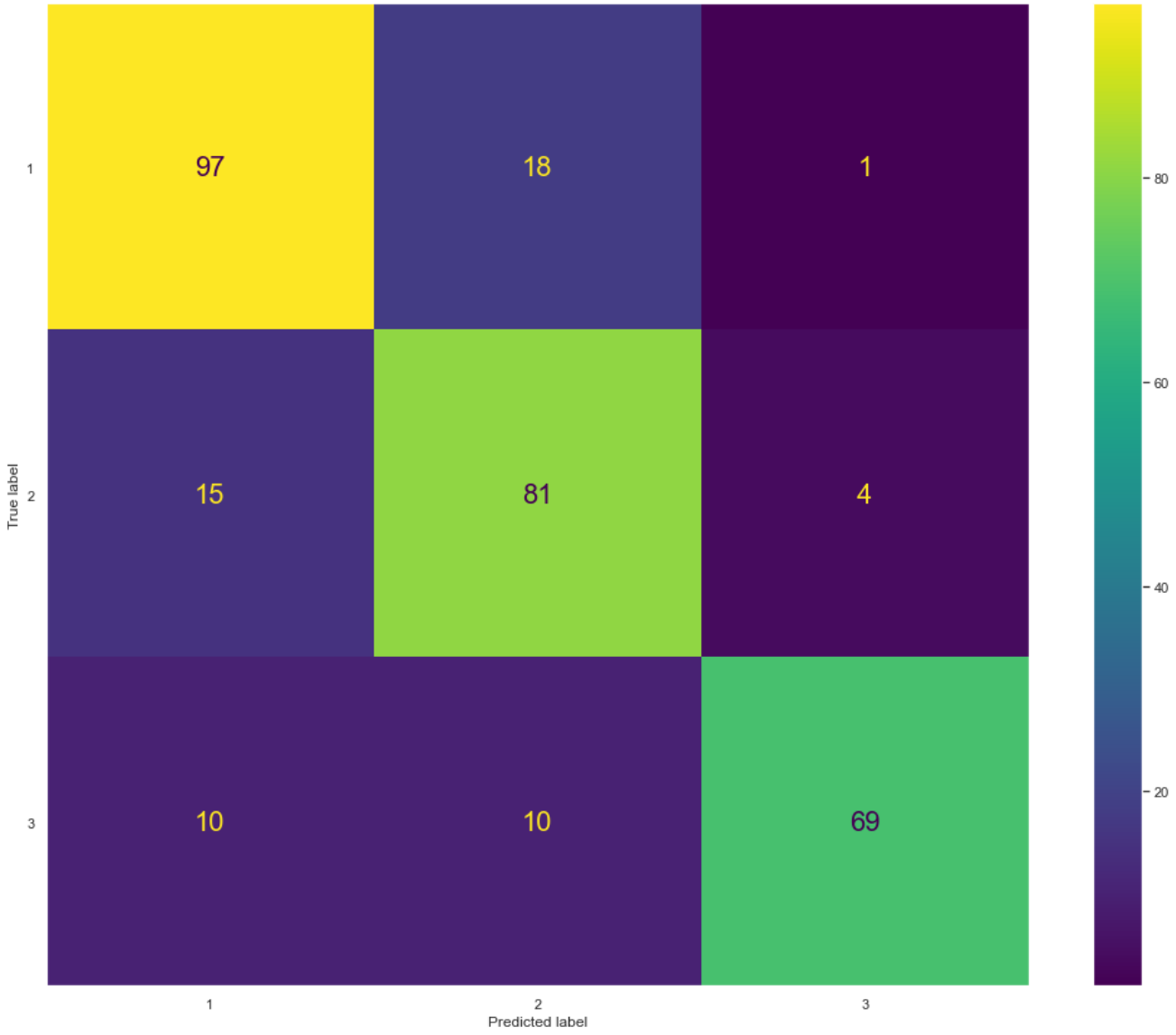
```
```

In [200…

```
# Gradient Boost

gb = GradientBoostingClassifier(n_estimators = 47 ,learning_rate = 1, loss = 'deviance')
gb.fit(scaled_X_train, y_train)

y_pred_gb = gb.predict(scaled_X_test)

gb_acc = accuracy_score(y_test, y_pred_gb)



report_test = classification_report(y_test, y_pred_gb)
print(f'The classification report for the training set is: \n {report_test}')


gb_acc = accuracy_score(y_test, y_pred_gb)
print('The accuracy of the test set is:',gb_acc)

gb_prec = precision_score(y_test, y_pred, average = "weighted")
print('The precision of the test set is:',gb_prec)

gb_f1 = f1_score(y_test, y_pred, average = "weighted")
print('The f1 of the test set is:',gb_f1)

print('The confusion matrix for the training set is:\n')
plot_confusion_matrix(gb, scaled_X_test, y_test)
plt.grid(False)
```

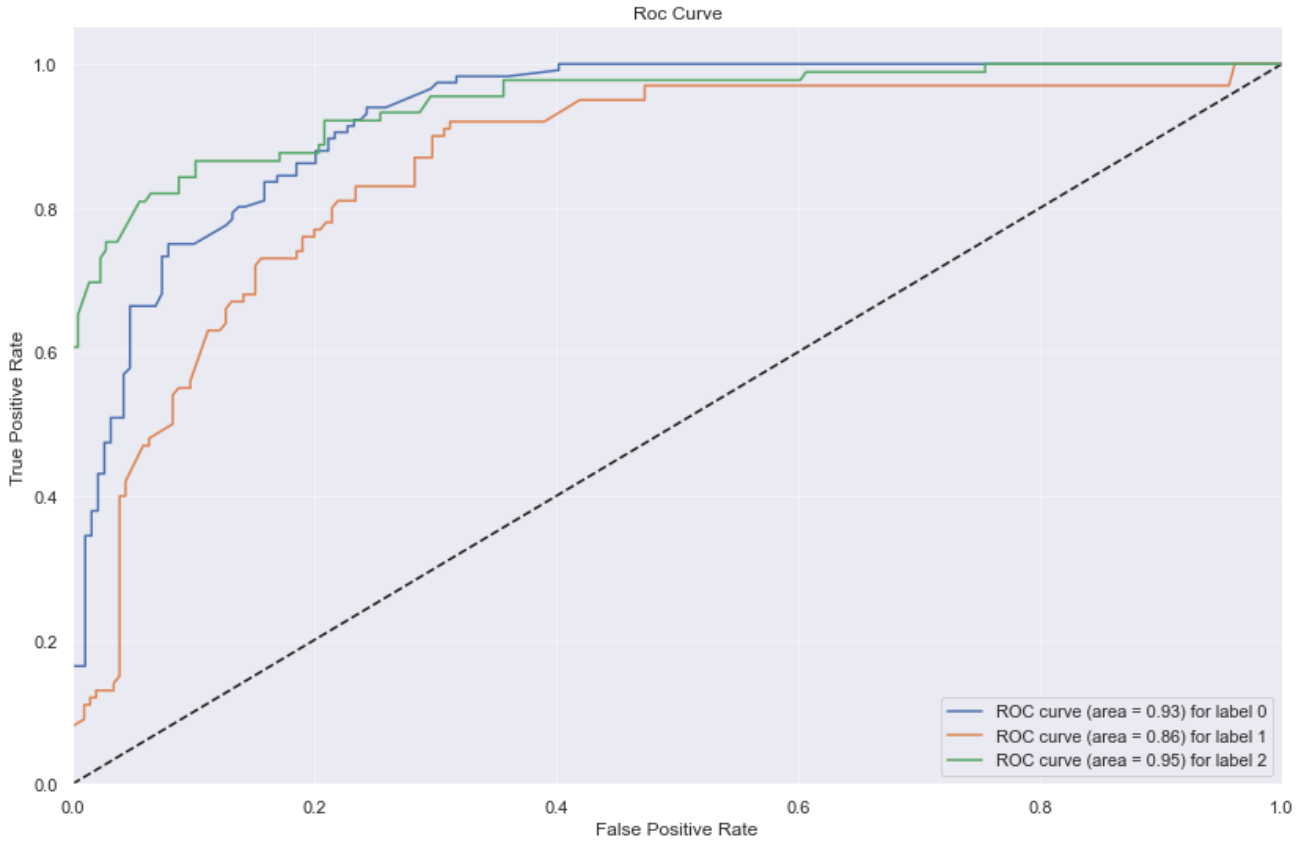
```
The classification report for the training set is:
              precision    recall  f1-score   support

           1       0.80      0.84      0.82       116
           2       0.74      0.81      0.78       100
           3       0.93      0.78      0.85        89
```

```
    accuracy                           0.81       305
   macro avg       0.82      0.81      0.81       305
weighted avg       0.82      0.81      0.81       305

The accuracy of the test set is: 0.8098360655737705
The precision of the test set is: 0.8120885821110153
The f1 of the test set is: 0.8047879913455178
The confusion matrix for the training set is:
```



In [200…

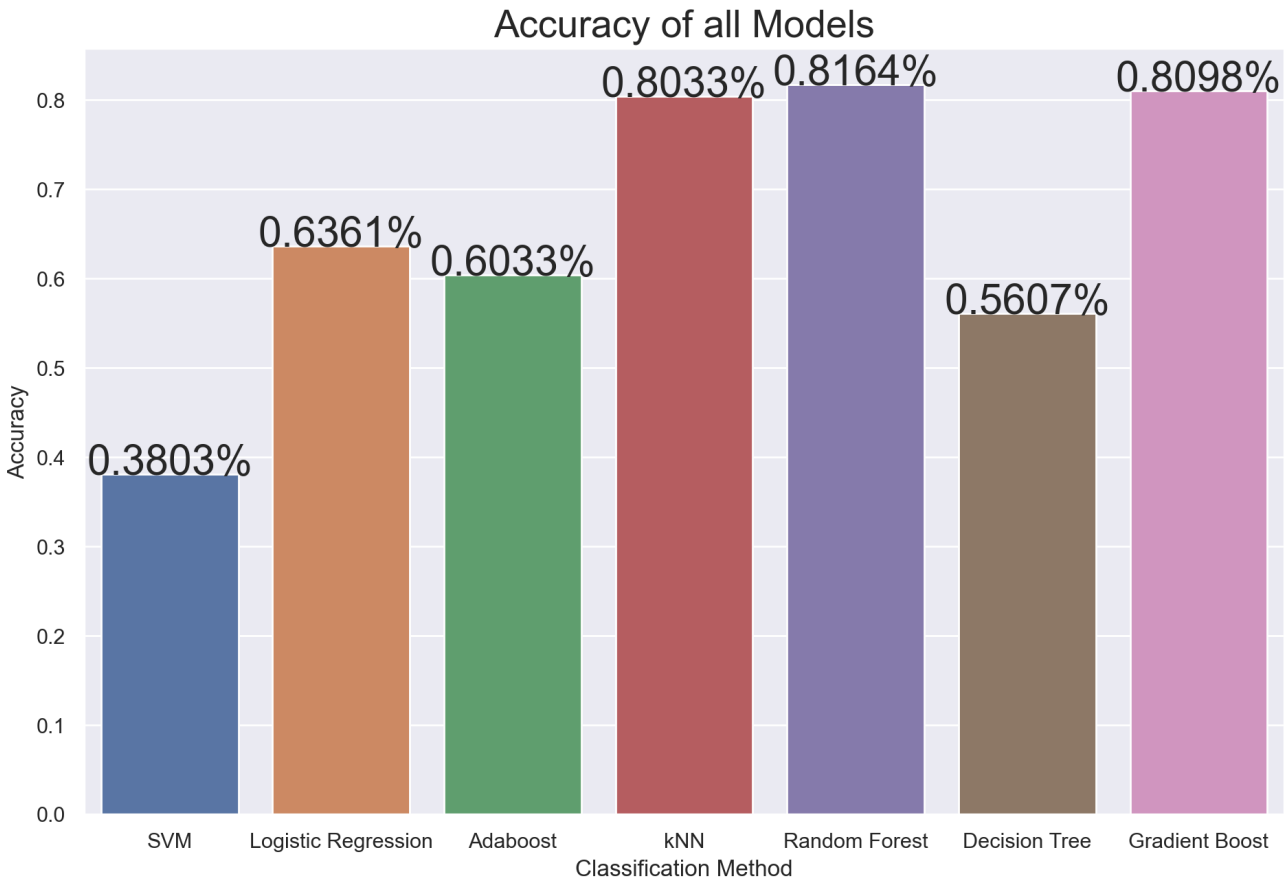
```
# Plot the roc curve for all three classes
plot_multiclass_roc(gb, scaled_X_test, y_test, n_classes = 3, figsize=(12,8))
```



D) Reflection

In [200…

```
# Plot the optimal accuracy scoores of all methods

clf_models = ["SVM","Logistic Regression", "Adaboost" ,"kNN", "Random Forest", "Decision T
accuracies = [svm_acc,lg_acc,ab_acc,knn_acc,rf_acc, dt_acc,gb_acc ]

plt.figure(figsize=(10,7), dpi=200)
ax = sns.barplot(x=clf_models, y=accuracies)
plt.title("Accuracy of all Models",fontsize =20)
plt.xlabel('Classification Method')
plt.ylabel("Accuracy")
for i in ax.patches:
    width, height = i.get_width(), i.get_height()
    x, y = i.get_xy()
    ax.annotate(f'{round(height,4)}%', (x + width/2, y + height*1.00), ha='center')
plt.show()
```


Accuracy of all Models
0.3803%
0.6361%
0.6033%
0.8033%
0.8164%
0.5607%
0.8098%
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
SVM
Logistic Regression
Adaboost
kNN
Random Forest
Decision Tree
Gradient Boost
Classification Method

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
COURSEWORK AML - TASK 2
```

Import Libraries

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D
import matplotlib.pyplot as plt
from google.colab import drive
from tensorflow.keras import Model

import numpy as np
import os

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

In [ ]:
```
# Make the storage devide accecible
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

Load the data and create the dataset

In [ ]:
```
# Upload the data from Google Drive
train_dir = 'drive/MyDrive/imageset/train'

# Define some parameters for the photos
BATCH_SIZE = 32
IMG_SIZE = (180, 180)

# Split the data images into 80% for training and 20% for validations
train_ds = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                       shuffle=True,
                                                       batch_size=BATCH_SIZE,
                                                       image_size=IMG_SIZE)
# train_ds = keras.utils.to_categorical(train_ds, 10) #10 is number of classes
validation_dir = 'drive/MyDrive/imageset/val'

val_ds = tf.keras.utils.image_dataset_from_directory(validation_dir,
                                                     shuffle=True,
                                                     batch_size=BATCH_SIZE,
                                                     image_size=IMG_SIZE)
```

```
Found 9469 files belonging to 10 classes.
Found 3925 files belonging to 10 classes.
```

In [ ]:
```
# Define some parameters for the images
batch_size = 32
img_height = 180
img_width = 180
```

In [ ]:
```
class_names = train_ds.class_names
```

Visualization of the data

In [ ]:
```
# Create a dictionary and rename the classes names
```

```
class_names = train_ds.class_names

names_dict = {
    'n01440764' : 'tench',
    'n02102040' : 'English_springer',
    'n02979186' : 'cassette_player',
    'n03000684' : 'chain_saw',
    'n03028079' : 'church',
    'n03394916' : 'French_horn',
    'n03417042' : 'garbage_truck',
    'n03425413' : 'gas_pump',
    'n03445777' : 'golf_ball',
    'n03888257' : 'parachute'
}

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    http://localhost:8889/notebooks/OneDrive/Desktop/Applied%20Machine%20Learning%20-%20CV
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(names_dict[class_names[labels[i]]])
    plt.axis("off")
```

In [ ]:

```
# Now we want to see the tenson of the shape

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
```

```
  print(labels_batch.shape)
  break
```

```
(32, 180, 180, 3)
(32,)
```

Configure the dataset

In [ ]:

```
AUTOTUNE = tf.data.AUTOTUNE

# Keep in the memory the images after they loaded for the first epoch
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)

# Overlap the data processing while it trains
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Standardize of the data

In [ ]:

```
# Scale the data from [0,1]
normalization_layer = layers.Rescaling(1./255)
```

In [ ]:

```
# Normalize the data
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]

print(np.min(first_image), np.max(first_image))
```

```
0.0 0.99850523
```

Creation of the Model

In [ ]:

```
num_classes = len(class_names)

# Layers
model = Sequential([
  # Rescaling layer input values to a new range
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  # Convolutional Layer of 16 kernel numbers and kernel size 3
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  # Max Pooling Layer
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  # Flatten Layer to convert from 2D image to 1D vector
  layers.Flatten(),
  # Dense layer with output size 128
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Compile the model

In [ ]:

```
# Compiling the model with optimizer 'Adam' and use of 'Entropy'
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [ ]:

```
# See the layers of the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 3,989,930
Trainable params: 3,989,930
Non-trainable params: 0
_________________________________________________________________
```

Train the model

In [ ]:

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  batch_size=32,
  epochs=10
  )
```

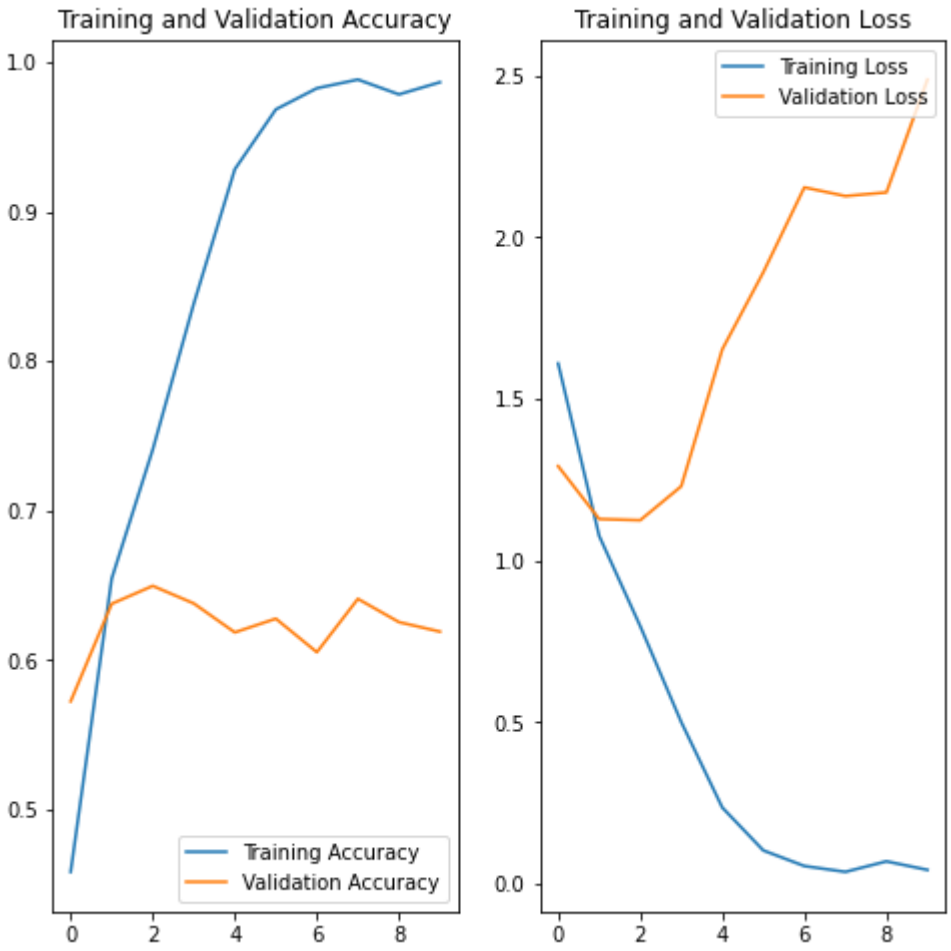
```
Epoch 1/10
296/296 [==============================] - 1088s 4s/step - loss: 1.6101 - accuracy: 0.4584
- val_loss: 1.2925 - val_accuracy: 0.5725
Epoch 2/10
296/296 [==============================] - 324s 1s/step - loss: 1.0768 - accuracy: 0.6546
- val_loss: 1.1290 - val_accuracy: 0.6377
Epoch 3/10
296/296 [==============================] - 325s 1s/step - loss: 0.7978 - accuracy: 0.7408
- val_loss: 1.1253 - val_accuracy: 0.6497
Epoch 4/10
296/296 [==============================] - 323s 1s/step - loss: 0.5010 - accuracy: 0.8384
- val_loss: 1.2303 - val_accuracy: 0.6380
Epoch 5/10
296/296 [==============================] - 323s 1s/step - loss: 0.2366 - accuracy: 0.9283
- val_loss: 1.6538 - val_accuracy: 0.6186
Epoch 6/10
296/296 [==============================] - 320s 1s/step - loss: 0.1038 - accuracy: 0.9683
- val_loss: 1.8924 - val_accuracy: 0.6278
Epoch 7/10
296/296 [==============================] - 321s 1s/step - loss: 0.0556 - accuracy: 0.9825
```

```
- val_loss: 2.1549 - val_accuracy: 0.6054
Epoch 8/10
296/296 [==============================] - 322s 1s/step - loss: 0.0375 - accuracy: 0.9883
- val_loss: 2.1280 - val_accuracy: 0.6410
Epoch 9/10
296/296 [==============================] - 323s 1s/step - loss: 0.0701 - accuracy: 0.9784
- val_loss: 2.1393 - val_accuracy: 0.6255
Epoch 10/10
296/296 [==============================] - 322s 1s/step - loss: 0.0437 - accuracy: 0.9865
- val_loss: 2.4884 - val_accuracy: 0.6191
```

Visualise training results

In [ ]:

```python
# Visualise the Accuracy and Loss of both Training and Validation sets

epochs = 10
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



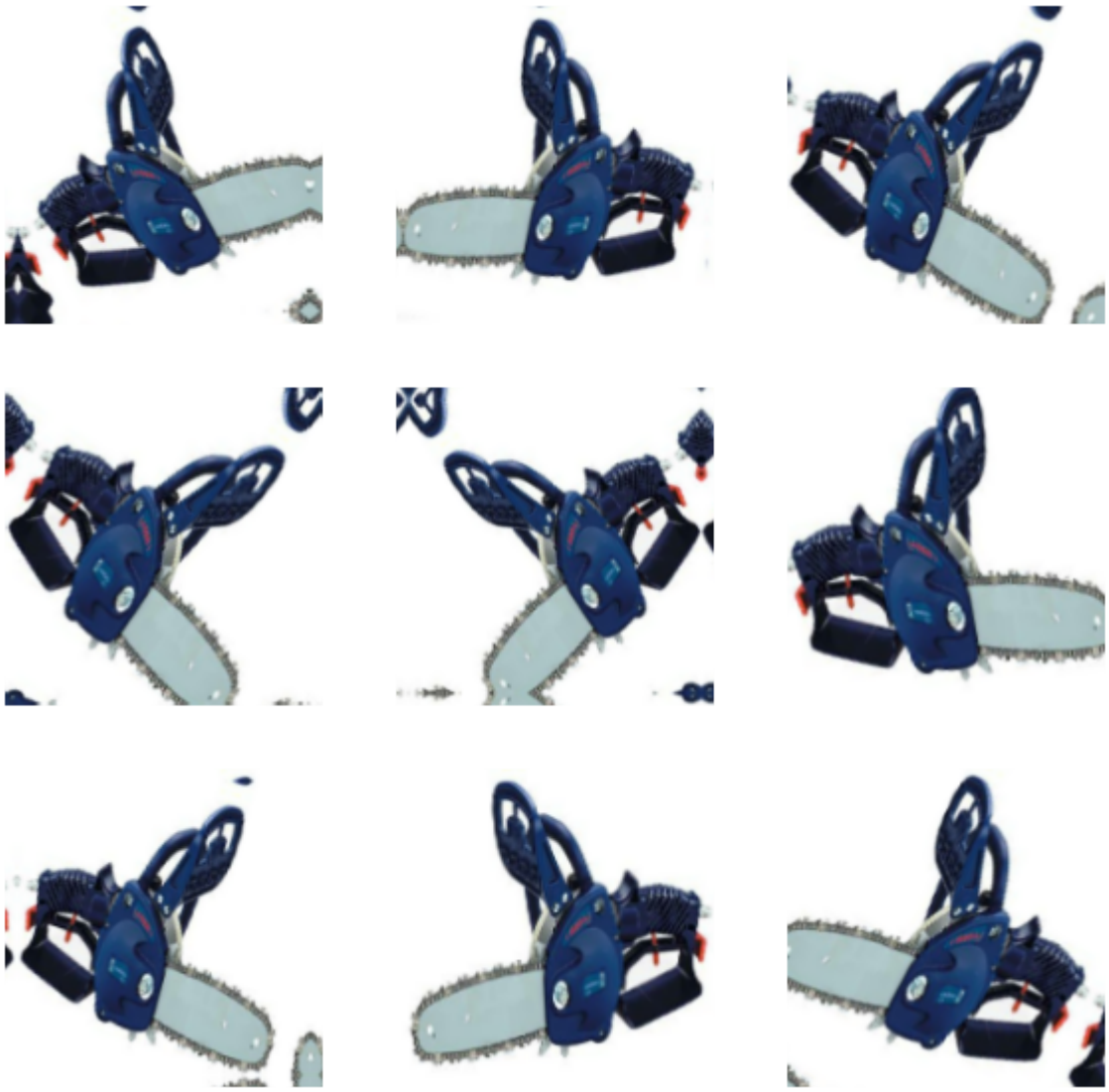
Data Augmentation

In [ ]:

```
# Due to overfitting we now try to impore using Data Augmentation
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

In [ ]:

```
# Plot augmented image example
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



Dropout

In [ ]:

```
# Dropout a number of units from the layer while it training
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Compile and train the model

In [ ]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [ ]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_1 (Sequential)   (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 dropout (Dropout)           (None, 22, 22, 64)        0

 flatten_1 (Flatten)         (None, 30976)             0

 dense_2 (Dense)             (None, 128)               3965056

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 3,989,930
Trainable params: 3,989,930
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

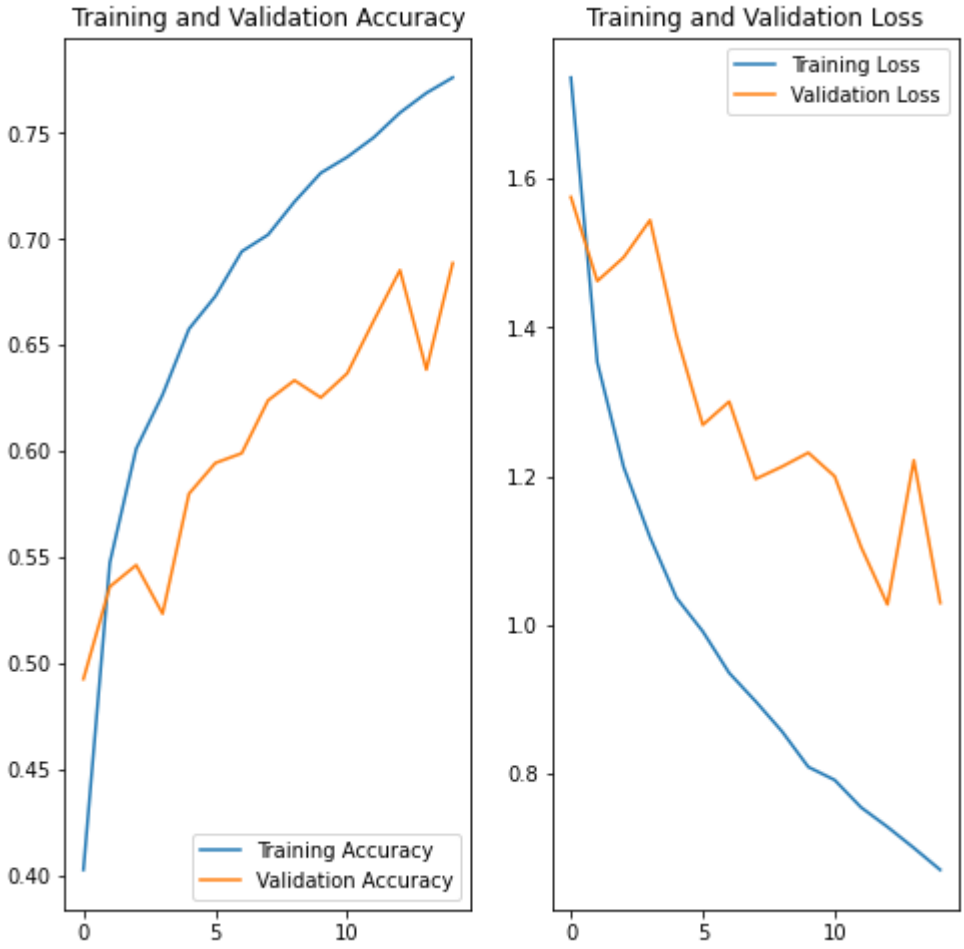
```
Epoch 1/15
296/296 [==============================] - 386s 1s/step - loss: 1.7359 - accuracy: 0.4025
- val_loss: 1.5753 - val_accuracy: 0.4925
Epoch 2/15
296/296 [==============================] - 390s 1s/step - loss: 1.3532 - accuracy: 0.5474
- val_loss: 1.4624 - val_accuracy: 0.5361
Epoch 3/15
296/296 [==============================] - 393s 1s/step - loss: 1.2124 - accuracy: 0.6011
- val_loss: 1.4942 - val_accuracy: 0.5462
Epoch 4/15
296/296 [==============================] - 390s 1s/step - loss: 1.1181 - accuracy: 0.6266
- val_loss: 1.5443 - val_accuracy: 0.5233
Epoch 5/15
296/296 [==============================] - 388s 1s/step - loss: 1.0369 - accuracy: 0.6576
- val_loss: 1.3897 - val_accuracy: 0.5799
Epoch 6/15
296/296 [==============================] - 388s 1s/step - loss: 0.9916 - accuracy: 0.6730
- val_loss: 1.2692 - val_accuracy: 0.5944
Epoch 7/15
296/296 [==============================] - 394s 1s/step - loss: 0.9358 - accuracy: 0.6942
- val_loss: 1.3005 - val_accuracy: 0.5990
Epoch 8/15
296/296 [==============================] - 391s 1s/step - loss: 0.8978 - accuracy: 0.7021
```

```
- val_loss: 1.1964 - val_accuracy: 0.6239
Epoch 9/15
296/296 [==============================] - 392s 1s/step - loss: 0.8574 - accuracy: 0.7176
- val_loss: 1.2130 - val_accuracy: 0.6334
Epoch 10/15
296/296 [==============================] - 396s 1s/step - loss: 0.8095 - accuracy: 0.7312
- val_loss: 1.2317 - val_accuracy: 0.6252
Epoch 11/15
296/296 [==============================] - 391s 1s/step - loss: 0.7919 - accuracy: 0.7387
- val_loss: 1.2000 - val_accuracy: 0.6367
Epoch 12/15
296/296 [==============================] - 391s 1s/step - loss: 0.7551 - accuracy: 0.7479
- val_loss: 1.1051 - val_accuracy: 0.6614
Epoch 13/15
296/296 [==============================] - 393s 1s/step - loss: 0.7290 - accuracy: 0.7596
- val_loss: 1.0278 - val_accuracy: 0.6854
Epoch 14/15
296/296 [==============================] - 389s 1s/step - loss: 0.7007 - accuracy: 0.7689
- val_loss: 1.2215 - val_accuracy: 0.6385
Epoch 15/15
296/296 [==============================] - 387s 1s/step - loss: 0.6712 - accuracy: 0.7762
- val_loss: 1.0300 - val_accuracy: 0.6887
```

Visualise training results

In [ ]:

```python
# Visualise the Accuracy and Loss of both Training and Validation sets

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```


Training and Validation Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
5
10
Training Accuracy
Validation Accuracy
Training and Validation Loss
1.6
1.4
1.2
1.0
0.8
0
5
10
Training Loss
Validation Loss

In [ ]:

In [ ]:

In [ ]: