Configuration Manual

MSc Research Project
Msc Data Analytics

Srivathsav Venugopal
Student ID: 20130660

School of Computing
National College of Ireland

Supervisor: Bharathi Chakravarthi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Srivathsav venugopal
Student ID:	X20130660
Programme:	Msc Data Analytics
Year:	2021
Module:	Msc Research Project
Lecturer:	Bharathi Chakravarthi
Submission Due Date:	31/01/2021
Project Title:	A study over Supervised and Unsupervised Learning in predicting the impact of bird strike in aviation industry
Word Count:	2418
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Srivathsav Venugopal
Date:	31/01/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Srivathsav Venugopal
Student ID: x20130660

1 Introduction:

Throughout the study, the software, hardware, and system setup have all been described in the configuration manual shown here. Implementation of the code used in the research project
" A Study over Supervised amd Unsupervised learning algorithms in predicting the impact of bird strike in aviation industry."

2 System configuration

2.1 Hardware

Intel(R) core i5 10th generation; 16GB RAM, 512GB SSD; Windows 11 64-bit operating system; and 16GB of RAM.

2.2 Software

python was used to carry out the study, and in order to go on with the analysis, the google colaboratory was used. The data pre-processing, feature selection, data validation, and implementation of machine learning algorithms were all carried out with the help of Python libraries and libraries for other languages. The raw data was gathered and analysed using Microsoft Excel, which is a spreadsheet programme.

3 Project Development

The research was primarily concerned with evaluating various machine learning technologies, which included a step-by-step procedure of obtaining data cleaning, preprocessing, features selection, and sampling approaches in order to balance the unbalanced data. When more than 60 percent of the data had no null values, they were discarded and replaced with fresh data. Data from previously verified sources were used to build the model, which was then iterated upon when machine learning methods were put into place. The data was sampled more thoroughly and a full dataset was obtained in order to conduct a thorough analysis. Afterwards, the data was entered into the data once again for confirmation.

3.1 Data collection and Data preprocessing

A kaggle repository was used to gather the data, which was then analysed in Microsoft Excel since the raw data needed to be saved as a CSV file. A Google Colaboratory examination of this data was then conducted out using python. Here is a step-by-step graphic of how to begin analysing data in Google Colab.

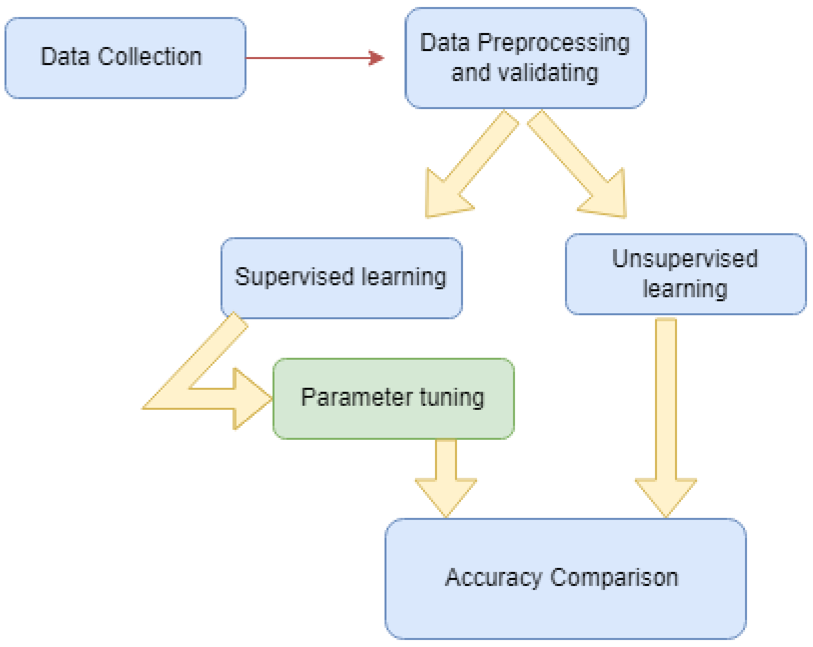
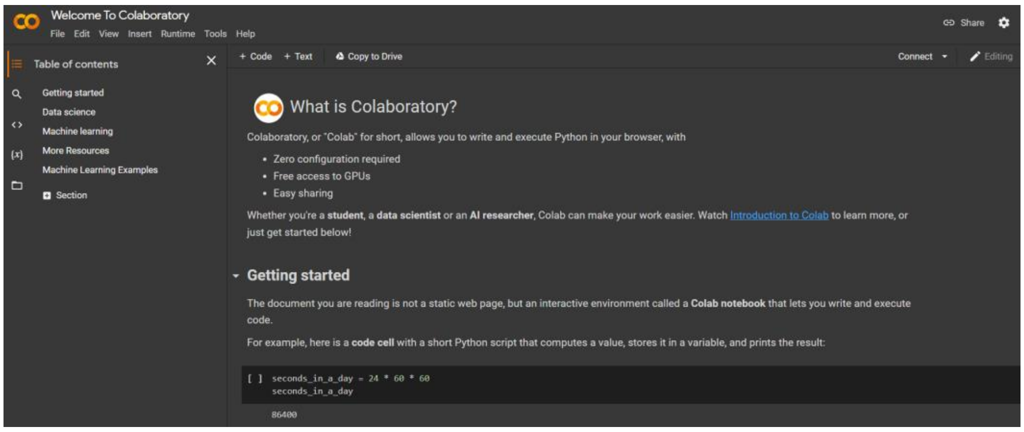


Figure 1. Methodology



What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

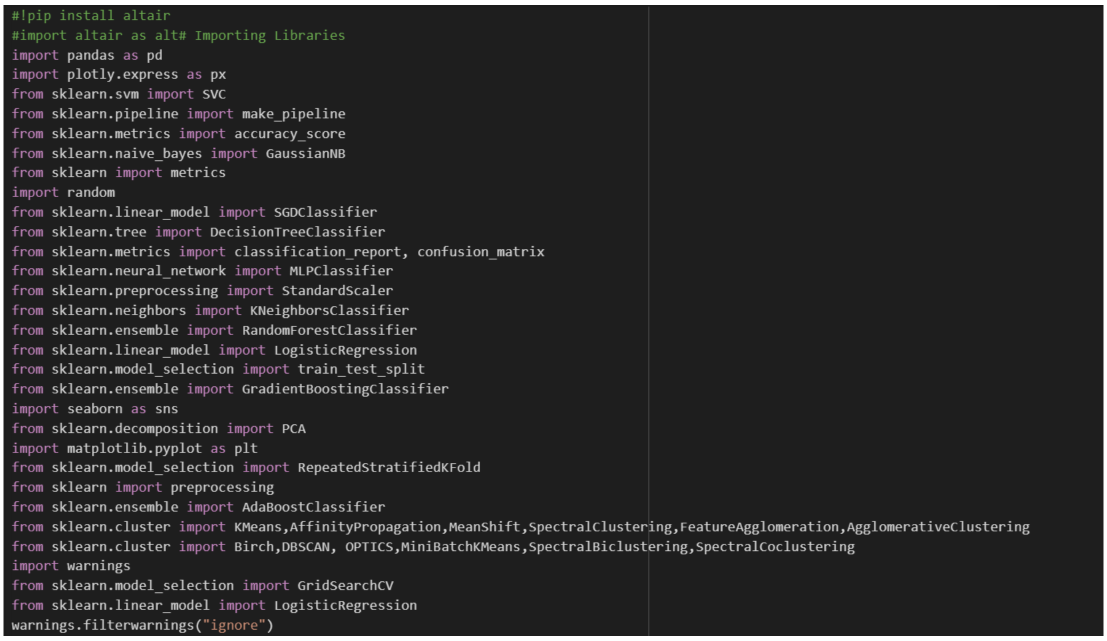
The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day

    86400
```

Figure 2: Implementing in Google colab

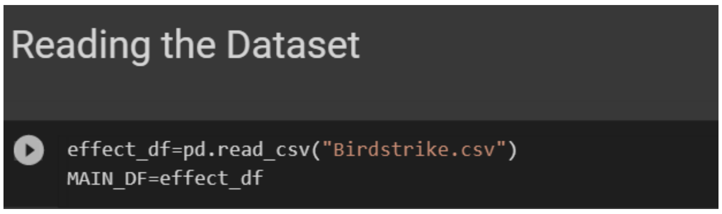
Once the data has been read programmatically, it is then placed in a second dataframe, which is then read again. Before that, the necessary libraries for the analysis were loaded into the system. The libraries for each portion of the analysis were imported into the environment by using the import command. The imported libraries into the notebook are shown in the image below. The data has been put into the environment after it has been compared to the data. It has been compared to the data. It can be seen from the technique diagram presented in (Figure 1) that data has been fed into the machine learning algorithms, which were around 12 in number, and that data has been pushed again after the validation has been completed. They were from both the controlled and unsupervised learning environments. After the data was entered, it was put into two different types of machine learning, supervised and unsupervised, in order to determine their usefulness in forecasting the damage caused by bird attacks. The more thoroughly described approach will be presented in the following paragraphs. The (figure 2), shows the colab directory where the data has been read by uploading the fie over there and then passing them in to the parameters in order to read the data programatically for the analysis was achieved,



```
#!pip install altair
#import altair as alt# Importing Libraries
import pandas as pd
import plotly.express as px
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
import random
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
import seaborn as sns
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn import preprocessing
from sklearn.ensemble import AdaBoostClassifier
from sklearn.cluster import KMeans,AffinityPropagation,MeanShift,SpectralClustering,FeatureAgglomeration,AgglomerativeClustering
from sklearn.cluster import Birch,DBSCAN, OPTICS,MiniBatchKMeans,SpectralBiclustering,SpectralCoclustering
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
warnings.filterwarnings("ignore")
```

Figure 3. Importing Libraries

To determine how successful supervised and unsupervised machine learning algorithms are in forecasting the effects of bird attacks in the aviation sector, the primary goal of this research is to compare the results of two different types of algorithms. The numerous libraries required for the models and other approaches that are being used have been created.

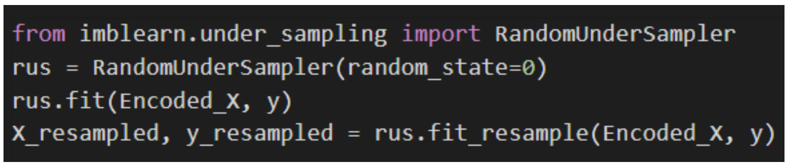


Reading the Dataset

```
effect_df=pd.read_csv("Birdstrike.csv")
MAIN_DF=effect_df
```

Figure 4. Reading the data

With the help of the Pandas package, data was read in the manner indicated (Figure 4). The data being read and saved in the dataframe is called the input data. Following the first reading of the data, the subsequent additional analysis was carried out step by step. After the data was first examined, it was discovered that there were more null columns than expected. After that, the data was filtered by substituting the null values in order to avoid losing a large number of features from the dataset. Features with more than 60 percent null values were removed from consideration for the analysis. The data was kept by substituting the null values with the features that were more supporting or required for the study. This was done in order to make the data full for the analysis. Once the data had been pre-processed, it was carried on to the next level of analysis, where the Exploratory Data Analysis (EDA) method was used to learn more about the data and discover new information. While developing the graphs from the EDA section, it was discovered that the data was too uneven, thus sampling procedures were used in order to make the data more balanced.



```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
rus.fit(Encoded_X, y)
X_resampled, y_resampled = rus.fit_resample(Encoded_X, y)
```

Figure 5. Handling Unbalanced Data

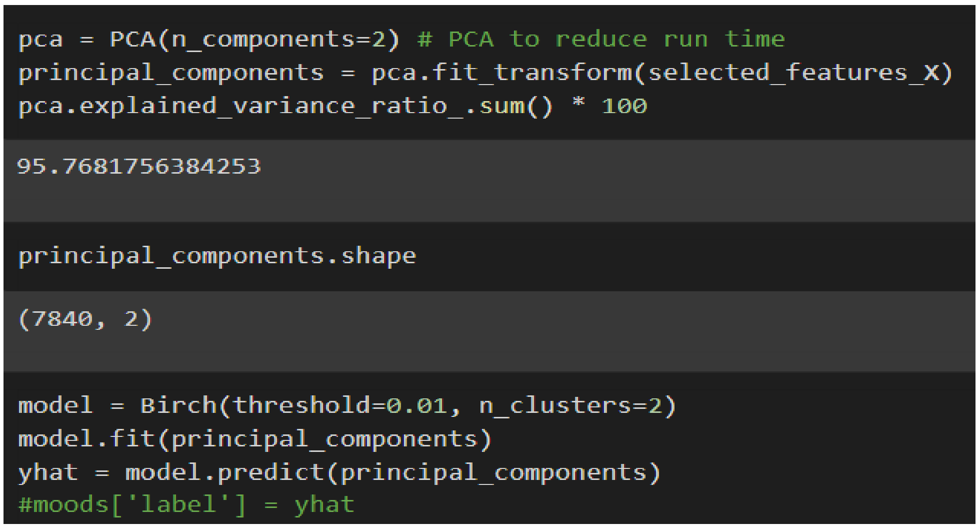
The data was balanced by implementing the above code as shown in (Figure 5).once after the balanced dataset has been achieved the data was moved for further analysis. The data has been further being explored using different various libraries and based on the plots there were more information carried out as a meaningful insights.

3.2 Machine learning implementation

Once after the data has been preprocessed and cleaned, the data has been fed into the machine learning algorithm. The data was further carried out to the important part of the analysis. Unsupervised machine learning approach were first implemented and once after that the data was then fed into the algorithms.

4 Unsupervised machine learning implementation

4.1 Birch Algorithm

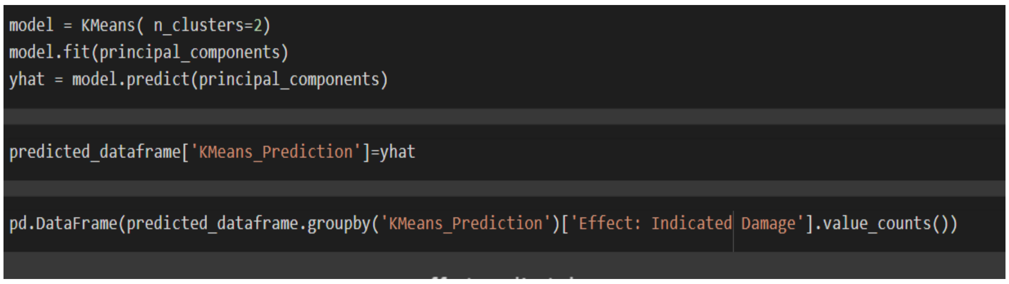


```
pca = PCA(n_components=2) # PCA to reduce run time
principal_components = pca.fit_transform(selected_features_X)
pca.explained_variance_ratio_.sum() * 100

95.7681756384253

principal_components.shape

(7840, 2)

model = Birch(threshold=0.01, n_clusters=2)
model.fit(principal_components)
yhat = model.predict(principal_components)
#moods['label'] = yhat
```

Figure 6. implementing Birch algorithm and PCA

The data was initially used in unsupervised machine learning techniques (figure 6), which were then applied to the data. The use of principal component analysis (PCA) has been incorporated in order to decrease the amount of time that the methods consume. Once the PCA analysis has been conducted, the data that has been judged to be the bare minimum in terms of data must be supplied with the greatest amount of information. After the birch algorithm has been applied, the data has been compared based on the accuracy of the birch algorithms. The accuracy of the model has been evaluated using several metrics, including f1, precision, and recall, and the correctness of the model has been studied and compared to other models that have been deployed.

4.2 K-means algorithm

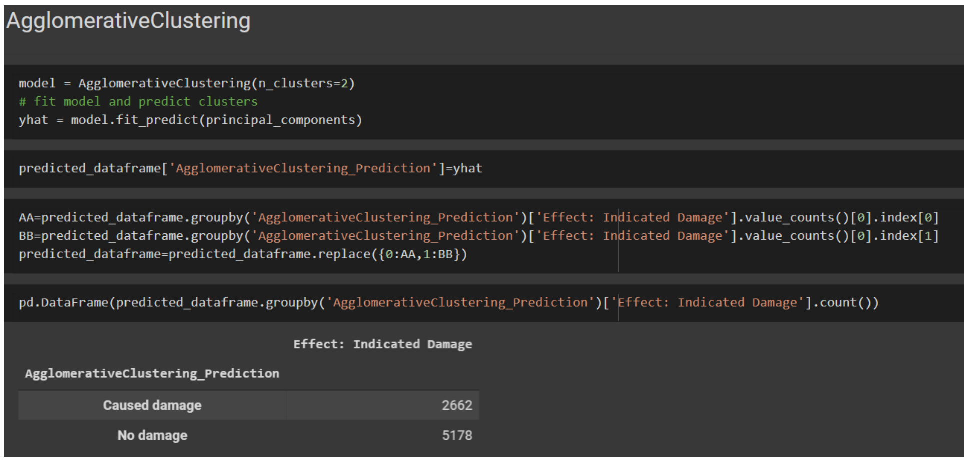


```
model = KMeans( n_clusters=2)
model.fit(principal_components)
yhat = model.predict(principal_components)

predicted_dataframe['KMeans_Prediction']=yhat

pd.DataFrame(predicted_dataframe.groupby('KMeans_Prediction')['Effect: Indicated Damage'].value_counts())
```

Figure 7. implementing K-means algorithm

To determine how well this model can predict, the k means method has been built and evaluated (Figure 7). In addition, as previously indicated, the PCA algorithm has been applied for all unsupervised algorithms in order to minimise the execution time and complexity of the algorithms. Immediately after the implementation of the k-means algorithm, the next algorithm was constructed using the other kind of machine learning algorithms available.

4.3 Agglomerative Clustering



AgglomerativeClustering

```
model = AgglomerativeClustering(n_clusters=2)
# fit model and predict clusters
yhat = model.fit_predict(principal_components)

predicted_dataframe['AgglomerativeClustering_Prediction']=yhat

AA=predicted_dataframe.groupby('AgglomerativeClustering_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[0]
BB=predicted_dataframe.groupby('AgglomerativeClustering_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[1]
predicted_dataframe=predicted_dataframe.replace({0:AA,1:BB})

pd.DataFrame(predicted_dataframe.groupby('AgglomerativeClustering_Prediction')['Effect: Indicated Damage'].count())
```

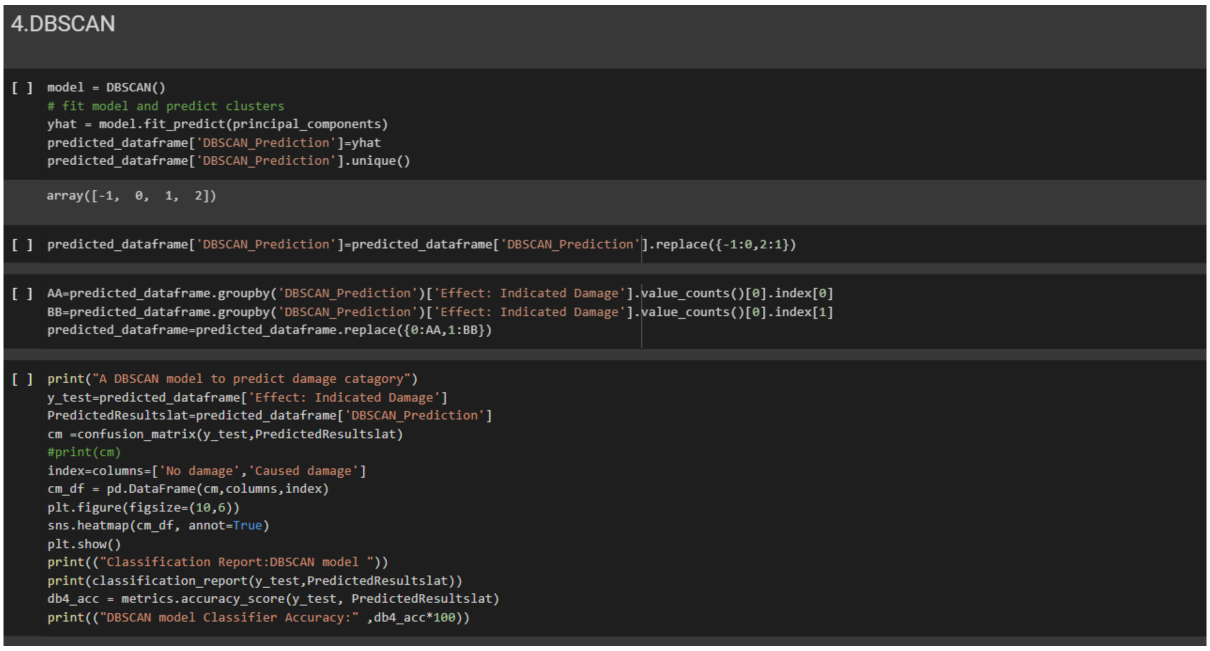
	Effect: Indicated Damage
AgglomerativeClustering_Prediction	
Caused damage	2662
No damage	5178

Figure 8. implementing Agglomerative clustering

The data has been evaluated using the agglomerative clustering model (Figure 8), which is now being implemented as the next model. Groupby function was built so that the data could be fed into the model in groups of three or four. The code was iindexed with 0 and 1 so that it could pass the values for the damage for the aircrafts that were being fed into the model correctly. The model's performance was evaluated by creating a heatmap based on the accuracy assessment, which will be refined in the next months. As the next model in this algorithm, it was implemented as the next model in this algorithm.

4.4 DBSCAN

It was necessary to assess the data after it had been put into the DBSCAN model (Figure 9) so that it could be determined to what extent it could be used to forecast the damage rate in aeroplanes. The most important information from the data was input into the model, and the same method was followed for the examination of all unsupervised machine learning algorithms, including the ones that were used in the study. After that, the information was sent into the next modelling process.... Their accuracy measures were used to evaluate the model, which was subsequently analysed. It was time to put the next model into action!

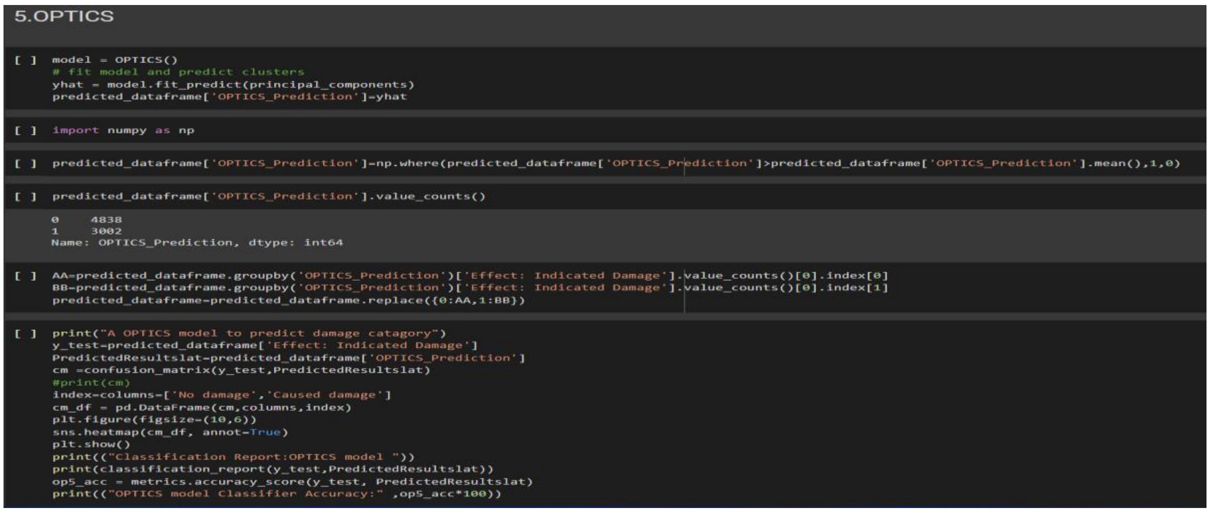


4.DBSCAN

```
[ ]  model = DBSCAN()
     # fit model and predict clusters
     yhat = model.fit_predict(principal_components)
     predicted_dataframe['DBSCAN_Prediction']=yhat
     predicted_dataframe['DBSCAN_Prediction'].unique()

     array([-1,  0,  1,  2])

[ ]  predicted_dataframe['DBSCAN_Prediction']=predicted_dataframe['DBSCAN_Prediction'].replace({-1:0,2:1})

[ ]  AA=predicted_dataframe.groupby('DBSCAN_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[0]
     BB=predicted_dataframe.groupby('DBSCAN_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[1]
     predicted_dataframe=predicted_dataframe.replace({0:AA,1:BB})

[ ]  print("A DBSCAN model to predict damage catagory")
     y_test=predicted_dataframe['Effect: Indicated Damage']
     PredictedResultslat=predicted_dataframe['DBSCAN_Prediction']
     cm =confusion_matrix(y_test,PredictedResultslat)
     #print(cm)
     index=columns=['No damage','Caused damage']
     cm_df = pd.DataFrame(cm,columns,index)
     plt.figure(figsize=(10,6))
     sns.heatmap(cm_df, annot=True)
     plt.show()
     print(("Classification Report:DBSCAN model "))
     print(classification_report(y_test,PredictedResultslat))
     db4_acc = metrics.accuracy_score(y_test, PredictedResultslat)
     print(("DBSCAN model Classifier Accuracy:" ,db4_acc*100))
```

Figure 9 : implementing and applying confusion matrix fro DBSCAN

4.5 OPTICS



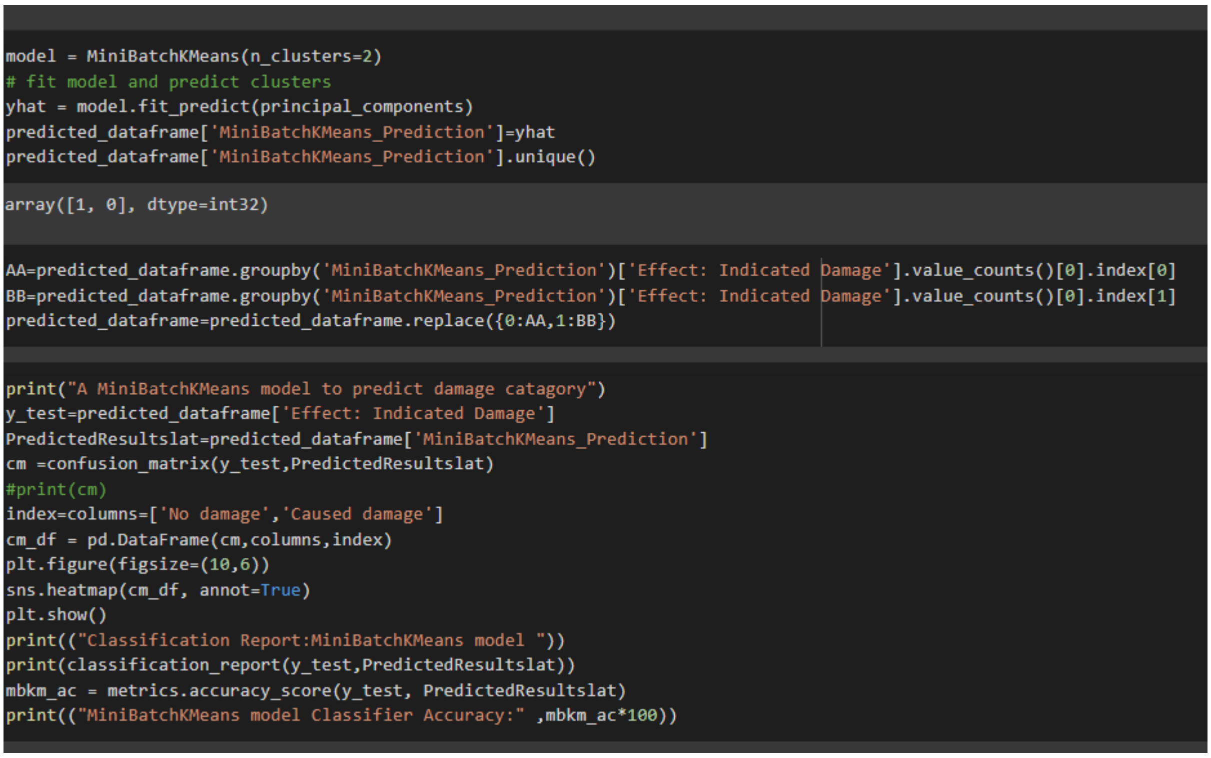
5.OPTICS

```
[ ]  model = OPTICS()
     # fit model and predict clusters
     yhat = model.fit_predict(principal_components)
     predicted_dataframe['OPTICS_Prediction']=yhat

[ ]  import numpy as np

[ ]  predicted_dataframe['OPTICS_Prediction']=np.where(predicted_dataframe['OPTICS_Prediction']>predicted_dataframe['OPTICS_Prediction'].mean(),1,0)

[ ]  predicted_dataframe['OPTICS_Prediction'].value_counts()

     0    4838
     1    3002
     Name: OPTICS_Prediction, dtype: int64

[ ]  AA=predicted_dataframe.groupby('OPTICS_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[0]
     BB=predicted_dataframe.groupby('OPTICS_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[1]
     predicted_dataframe=predicted_dataframe.replace({0:AA,1:BB})

[ ]  print("A OPTICS model to predict damage catagory")
     y_test=predicted_dataframe['Effect: Indicated Damage']
     PredictedResultslat=predicted_dataframe['OPTICS_Prediction']
     cm =confusion_matrix(y_test,PredictedResultslat)
     #print(cm)
     index=columns=['No damage','Caused damage']
     cm_df = pd.DataFrame(cm,columns,index)
     plt.figure(figsize=(10,6))
     sns.heatmap(cm_df, annot=True)
     plt.show()
     print(("Classification Report:OPTICS model "))
     print(classification_report(y_test,PredictedResultslat))
     op5_acc = metrics.accuracy_score(y_test, PredictedResultslat)
     print(("OPTICS model Classifier Accuracy:" ,op5_acc*100))
```

Figure 10. OPTICS model

While compared to other algorithms, the OPTICS algorithm's performance (Figure 10) and unique style of techniques were taken into consideration when developing the algorithm. The optics method was implemented in a similar manner as the previous algorithm, using the same computer language. Further analysis of the data was carried out using the code in order to forecast the target that was being further evaluated alongside the data. The accuracy of their models was assessed based on the overall performance of the models..

4.6 Mini- batch K means



```
model = MiniBatchKMeans(n_clusters=2)
# fit model and predict clusters
yhat = model.fit_predict(principal_components)
predicted_dataframe['MiniBatchKMeans_Prediction']=yhat
predicted_dataframe['MiniBatchKMeans_Prediction'].unique()
```

```
array([1, 0], dtype=int32)
```

```
AA=predicted_dataframe.groupby('MiniBatchKMeans_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[0]
BB=predicted_dataframe.groupby('MiniBatchKMeans_Prediction')['Effect: Indicated Damage'].value_counts()[0].index[1]
predicted_dataframe=predicted_dataframe.replace({0:AA,1:BB})
```

```
print("A MiniBatchKMeans model to predict damage catagory")
y_test=predicted_dataframe['Effect: Indicated Damage']
PredictedResultslat=predicted_dataframe['MiniBatchKMeans_Prediction']
cm =confusion_matrix(y_test,PredictedResultslat)
#print(cm)
index=columns=['No damage','Caused damage']
cm_df = pd.DataFrame(cm,columns,index)
plt.figure(figsize=(10,6))
sns.heatmap(cm_df, annot=True)
plt.show()
print(("Classification Report:MiniBatchKMeans model "))
print(classification_report(y_test,PredictedResultslat))
mbkm_ac = metrics.accuracy_score(y_test, PredictedResultslat)
print(("MiniBatchKMeans model Classifier Accuracy:" ,mbkm_ac*100))
```

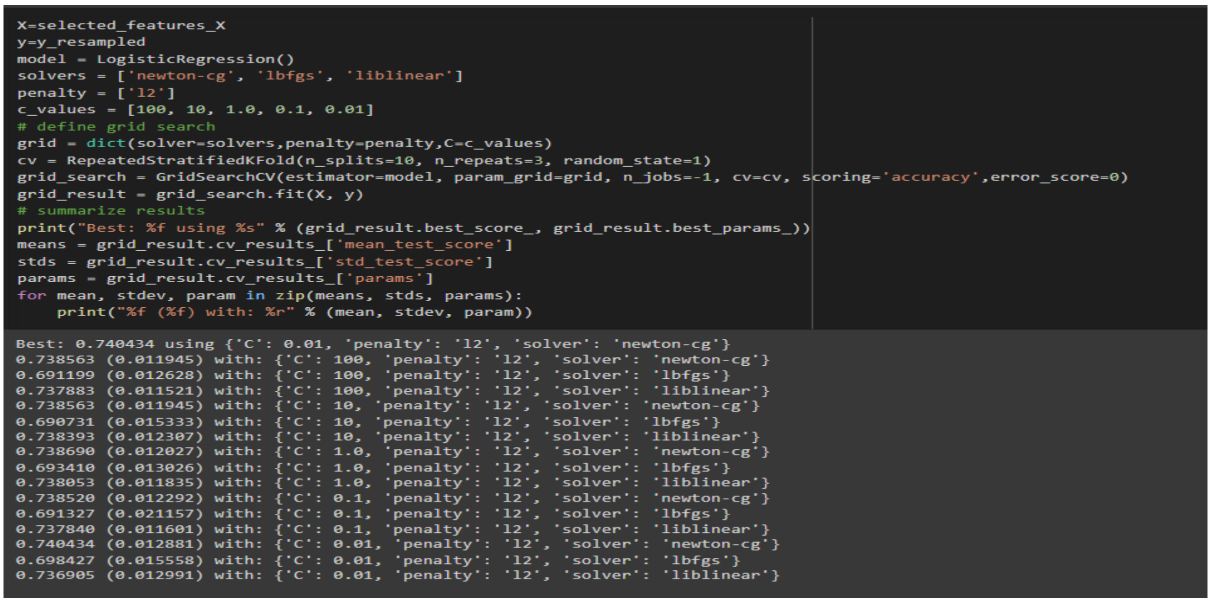
Figure 11. Mini batch K-means

Mini batch k means is a prediction method that is more accurate in terms of accuracy than the K-means algorithm, and it works in a similar way to the K-means algorithm. This was done in order to see how far the data might be analysed (Figure 11) further with the help of these models. The f1 score, precision, and recall metrics were found to be more accurate in their prediction, and it is obvious from the study that both the k-means and the Mini batch k-means did well in terms of prediction.
The data was then placed into a supervised learning technique, where each data point was subjected to parameter adjustment in order to improve accuracy as well as to determine how effective supervised learning is in terms of prediction. After then, the data was examined and predictions were made based on the data.

5 Supervised learning algoritms

5.1 Logistic regression:

The first model that was implemented in Supervised learning was to be the Logistic regression (Figure 12), once after the validated data has been moved into the model, the paramter tuning for each parameters were implemented, each parameter has a score of accuracy based on them that was actually chosen for the analysis. The data was then applied with different metrics to be evaluated. Along with the parameters the standard and mean test score has been fed into the model.



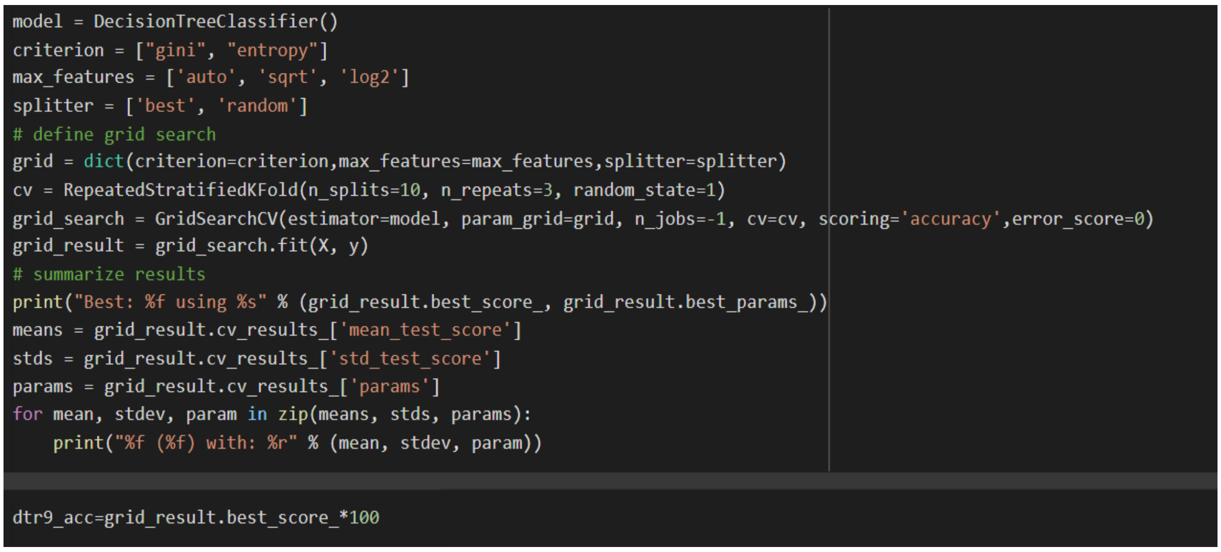
```
X=selected_features_X
y=y_resampled
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.740434 using {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.738563 (0.011945) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.691199 (0.012628) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.737883 (0.011521) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.738563 (0.011945) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.690731 (0.015333) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.738393 (0.012307) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.738690 (0.012027) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.693410 (0.013026) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.738053 (0.011835) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.738520 (0.012292) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.691327 (0.021157) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.737840 (0.011601) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.740434 (0.012881) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.698427 (0.015558) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.736905 (0.012991) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

Figure 12. Logistic regression implementation

The supervised learning being one of the more accurate in predictionns when compared to the other unsupervised learning techniques. The analysis carried out using supervised learning algorithms were more higher in results, their results were above 70 %. The whole process being carried out has the same method of implementation with different paramteres and they were tuned accordingly as to have an higher accuracy rate. Based on the metrics, the accuracy of each model has been evaluated as to check how far the data has been analyzed and have a good prediction rate among the models.

5.2 Decision tree algorithm



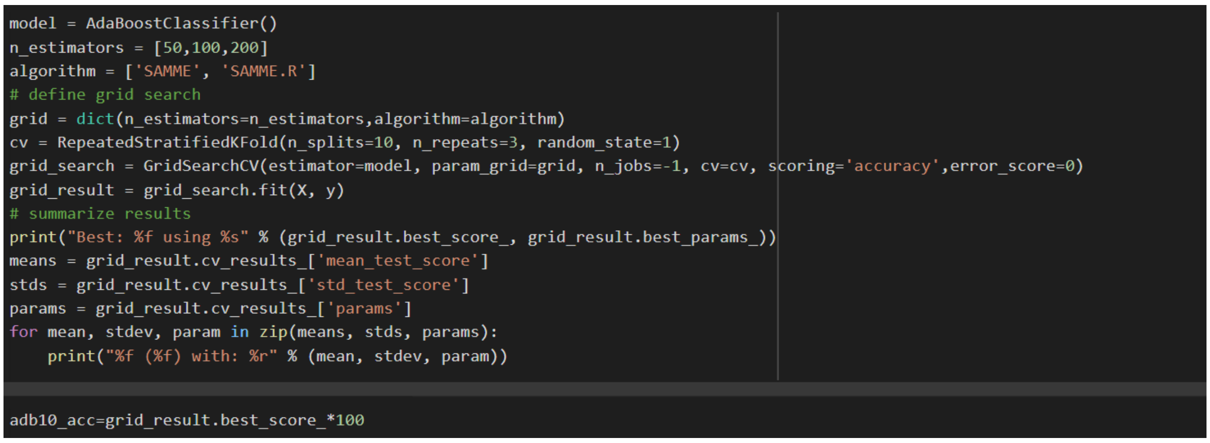
```
model = DecisionTreeClassifier()
criterion = ["gini", "entropy"]
max_features = ['auto', 'sqrt', 'log2']
splitter = ['best', 'random']
# define grid search
grid = dict(criterion=criterion,max_features=max_features,splitter=splitter)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
dtr9_acc=grid_result.best_score_*100
```

Figure 13. Decision tree classifier

The data that has been further studied has really produced the data; the prediction of each model changes, as does their score in parameter tuning; the models being run with the same methods of model are to be detailed in further detail in the next parts. The analysis that is now being conducted is aimed at anticipating the harm that the birds would do to the aircraft. Once the test results have been received and analysed, the model may be assessed. The decision tree classifier was selected since it was more distinctive in its performance. The model was then developed using the parameters that were provided by the user.

5.3 AdaBoost Classifier

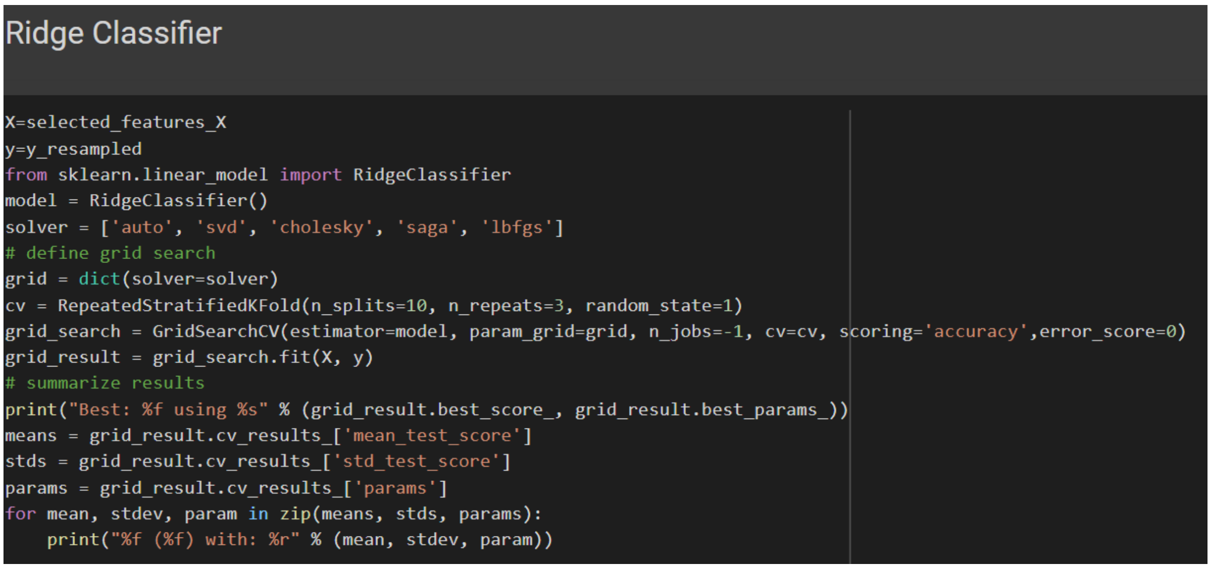


```
model = AdaBoostClassifier()
n_estimators = [50,100,200]
algorithm = ['SAMME', 'SAMME.R']
# define grid search
grid = dict(n_estimators=n_estimators,algorithm=algorithm)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

adb10_acc=grid_result.best_score_*100
```

Figure 14. AdaBoost Classifier implementation

The boosting algorithm, also known as the adaboost classifier, was designed in order to test their effectiveness, and it was also implemented in order to reduce the bias and variance. When it came to implementing the model, we used the same implementation method that we had used in the previous phases. a situation in which the algorithm was to be in the same boosting notion and the data was to be fed into and evaluated.

5.4 Ridge Classifier



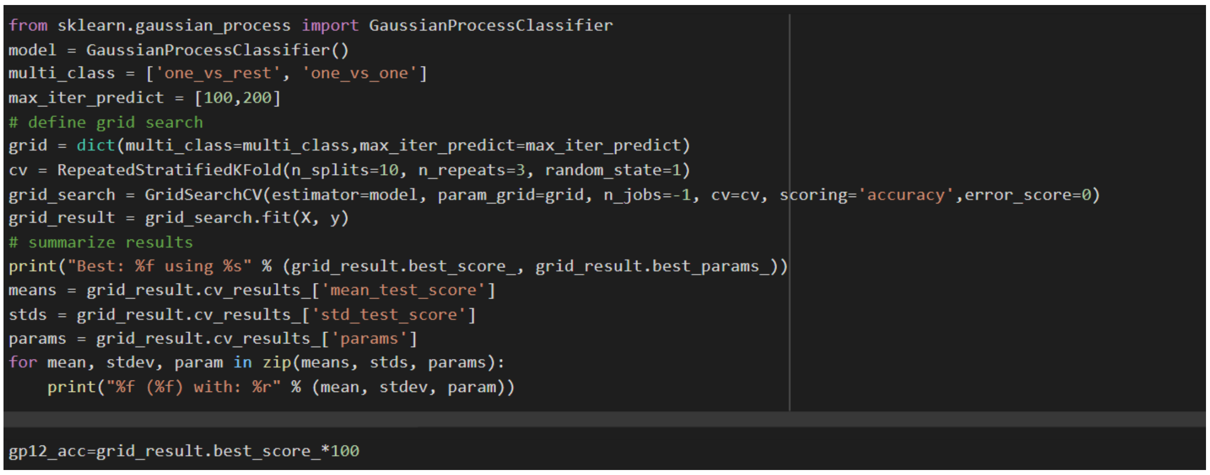
Ridge Classifier

```
X=selected_features_X
y=y_resampled
from sklearn.linear_model import RidgeClassifier
model = RidgeClassifier()
solver = ['auto', 'svd', 'cholesky', 'saga', 'lbfgs']
# define grid search
grid = dict(solver=solver)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Figure 15. Ridge Classifier implementation

When their samples of classes are dependent over the linear subspaces, which is accomplished by the ridge regreesor, the same performance is carried out in the subspaces where their samples of classes are reliant over the linear subspaces The analysis and prediction of the target using the ridge classifier will be more accurate in terms of prediction since it will be possible to see how far the data has been utilised to analyse and forecast it. The moel was assessed in accordance with the measures that were applied.

5.5 Gaussian Process Classifier

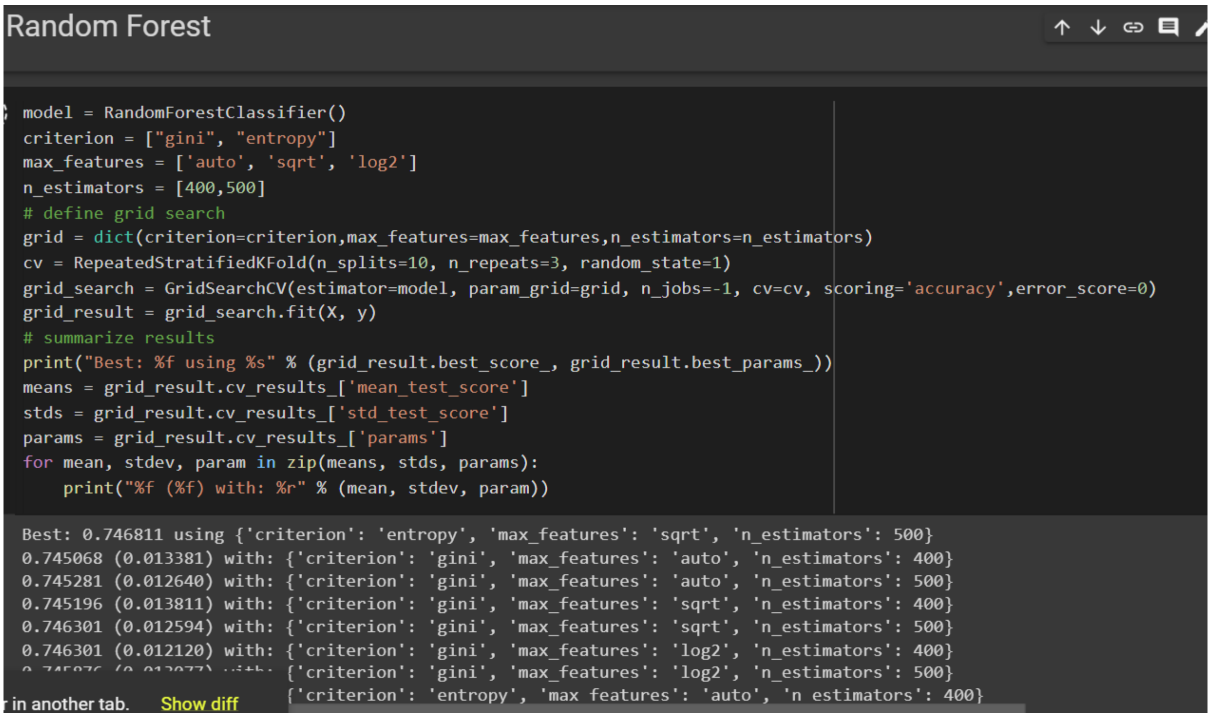


```
from sklearn.gaussian_process import GaussianProcessClassifier
model = GaussianProcessClassifier()
multi_class = ['one_vs_rest', 'one_vs_one']
max_iter_predict = [100,200]
# define grid search
grid = dict(multi_class=multi_class,max_iter_predict=max_iter_predict)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

gp12_acc=grid_result.best_score_*100
```

Figure 16. Gaussian classsifier implementation

Because the gaussian process classifier (GPC) is more accurate in terms of prediction, it is necessary to check how far the data is to be analysed using this model, which is known to be generalised concepts of the gaussian probability distribution from the gaussian process classifier (GPC). The model's performance was evaluated based on the accuracy rate that was achieved by the model in terms of prediction..

5.6 Random Forest Algorithm



Random Forest

```
model = RandomForestClassifier()
criterion = ["gini", "entropy"]
max_features = ['auto', 'sqrt', 'log2']
n_estimators = [400,500]
# define grid search
grid = dict(criterion=criterion,max_features=max_features,n_estimators=n_estimators)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.746811 using {'criterion': 'entropy', 'max_features': 'sqrt', 'n_estimators': 500}
0.745068 (0.013381) with: {'criterion': 'gini', 'max_features': 'auto', 'n_estimators': 400}
0.745281 (0.012640) with: {'criterion': 'gini', 'max_features': 'auto', 'n_estimators': 500}
0.745196 (0.013811) with: {'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators': 400}
0.746301 (0.012594) with: {'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators': 500}
0.746301 (0.012120) with: {'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 400}
[illegible] {'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 500}
in another tab.    Show diff   {'criterion': 'entropy', 'max_features': 'auto', 'n_estimators': 400}
```

Random forest classifier algorithm was implemented into the analysis as it is to be one of the best predicting algorithm. The data being fed into the model needs to be more accurate in terms of prediction and the evaluation takes place in terms of the applied metrics. Random forest being one of the possiblity of developing more uncorrelated trees were implemented using the feature boosting and randomization technique as to make the model to be more accurate in terms of prediction

References

Bata, M., Carriveau, R. and Ting, D., 2020. Short-term water demand forecasting using hybrid supervised and unsupervised machine learning model. *Smart Water*, 5(1).

Mahesh Kumar, K. and Rama Mohan Reddy, A., 2016. A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method. *Pattern Recognition*, 58, pp.39-48.

Agrawal, K., Garg, S., Sharma, S. and Patel, P., 2016. Development and validation of OPTICS based spatio-temporal clustering technique. *Information Sciences*, 369, pp.388-401.

Sun, S., Zhong, P., Xiao, H. and Wang, R., 2015. Active Learning With Gaussian Process Classifier for Hyperspectral Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, 53(4), pp.1746-1760.

Schein, A. and Ungar, L., 2007. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3), pp.235-265.

Roos, T., Wettig, H., Grünwald, P., Myllymäki, P. and Tirri, H., 2005. On Discriminative Bayesian Network Classifiers and Logistic Regression. *Machine Learning*,.