Predicting hotel bookings cancelation using logistic regression , K-NN algorithm,GNB classifier ,Random forest classifier and XGB classifier

Abas Mohammedidris
Faculty of Engineering, Environment and Computing, Coventry University
MSc Data Science and Computational Intelligence (ECT104) Stage 1
Coventry, United Kingdom
Moham911@uni.coventry.ac.uk

***Abstract*—Customers cancelling bookings will impact the revenue and will affect the demand management decisions in the hotels or in general the hospitality industry. In attempt to solve this issue, this paper presents a machine learning models that can predict the output of the bookings, whether the customer is likely to cancel or not. For this study a dataset of two hotels was used and five different classification algorithms, each algorithm was trained, tested and validated three times due to the nature of the dataset, which was slightly imbalanced and to balance it, three techniques was implemented.**

Keywords—hotel booking cancelation, class imbalance, feature reduction, data outlier, grid search ,classification

I. INTRODUCTION

In recent years , tourism was one of the most industries that have witnessed expansion in the world , which makes it a major contributor to the world economy [3] and the number of international tourist is growing continuously , evidently in 1990 the number of international tourist exceeded 400 million and in 2017 this number approached 1300 million . Just in 2017 the number grew by 7%, which makes the highest growth since the economic crises in 2009 [4]. From economic development point of view , the tourism industry has various contribution by generating income in such ways, goods consumption , tourist service employment opportunity etc and the most important sector within this industry is the hospitality sector or hotels.

The main fundamental of successful hospitality revenue such as hotels is the demand forecast [3] , and the easiest method to manage the demand forecast is by taking pre-booking for the hotel, which ease the preparation process and manging the revenue. However, the pre-booking has its positive impact as it increases the revenue forecasting , it also has a negative impact which is the booking cancelation and this could be due to uncertainty the tourist faces such as political instability , weather and natural disaster etc [5].Therefore, the hotel industry offers goods which cannot be stockpiled and the unoccupied room results to loss of revenue opportunity [6]. Hence , it is very important ,organisation and planning are fundamental as well as having an accurate tool that predict the forecasting and this can be achieved by applying some machine learning algorithms. As a solution to the mentioned issue several researchers proposed a machine-learning techniques , a study was conducted by [3] used four different machine learning algorithms whose are Random forest , support vector machine C5.0 and artificial neural network genetic algorithms (ANN GE) , the algorithms were developed based on hotel booking on Spain and comparison was performed to the models in terms of accuracy ,F1 score , precision , specificity and AUC and (ANN GE) was found to be the best in terms of all the mentioned performance measures followed by the random forest algorithms. Kernel extreme learning machine (ELM) which is a type of single-hidden layer forward neural network was proposed to the study of hotel revenue forecasting by [7] for the application of tourist arrivals with internet search index and showed a speedy and generalised process. A classification model which predict the bookings cancelation based on hotel contacted booking and booking not contacted found that the cancelation is less for the contacted booking [7], which tells that , the classification can classify using such approach.

As the state of art, literature review indicates , more machine learning models needs to be applied in order to tackle the issue and find the most appropriate model that can predict the cancelation that will help in making decision regard the hotel revenue. Therefore, on this study five machine learning models will be used whose are random forest , logistic regression , K-NN ,gaussian naïve Bayes classifier and XGB classifier.

II. THE DATASET

The dataset under consideration for this study is a hotel bookings dataset, which was obtained from ‘Kaggle’ website , and it is a combination of two hotels data, which was written by [1],the data has two main dimensions whose are , the time booking was made and the period of stay. Which means the booking can be cancelled in any time between the date of booking and the anticipated arrival data of the customer. In terms of features, the dataset contains 32 columns and 119390 instance and size-wise, it is 130MB and each instance is a customer unique booking regardless of it was checked in or cancelled. The dataset has two classes ,cancelled booking and not cancelled , therefore , the aim of this study is to use the data and develop prediction models , to classify the booking as one of the two above mentioned classes. The initial data explorations showed that , the dataset has various issue needs to be solved , before proceeding to build a machine learning models. The main problems are missing values , outliers and categorical data columns, there are three features within the datset that consist of

higher number of missing values ,whose are the company ,agent and country .Also the dataset is slightly imbalanced as it is shown in fig(II.1) , the figure also shows the percentage of the cancelled bookings versus not cancelled, the aim is to make the best usage of the entire dataset ,however ,due to the computational power limitation some columns will be dropped in the process of data preparation.

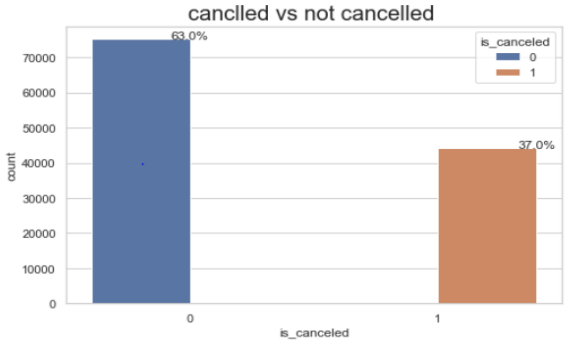

Figure II.1 cancelled booking vs not cancelled

A. *Date preparation*

To prepare the data , the first issue needs to be solved is handling the missing values, as it was addressed in the previous section , three features consist of missing values, therefore each column ,which contains missing values is handled separately in the most appropriate method for this study.

The feature company is categorical, and it contains a high number of missing values and has a high degree of cardinality which makes the model take longer to train , therefore it was removed from the dataset.

The second feature with missing values is the feature "country" , and it was found that , the country Portugal has the highest occurrence within the dataset , with an occurrence of 40.9% ,in attempt of solving the missing value in the country feature , the missing value were replaced by the country Portugal as it was assigned in most of the bookings , however , the majority of cancelled booking has the country "Portugal " assigned to it, which was proven , when the model was trained that it is a case of leakage, therefore the feature country was dropped from the modelling dataset.

The third case of missing value was the feature "agent" , according to [8],the missing values can be substituted with a chosen value based on the general characteristics of the variable such as 'zero' , 'mode' which is the most occurring value ,this method was applied to solve the previous case which is the country and for ordinal and continuous features the mean ,median , mode , maximum and minimum value .Therefore for the feature agent , the mean value was computed and the missing value was substituted by the mean value.

The fourth feature with a missing value is the feature "children" and it has four missing value, hence it is very small and will not have an impact on the modelling dataset ,it was substituted by "zero".

As it is shown in table(VII.1) in appendix(1) , the dataset has originally 32 features and more than 120,000 instances which makes it too complex for computation process , therefore , the features that do not contribute with value or introduce noise are dropped , the removed features arrival_date_year,arrival_date_month,arrival_date_week_number,arrival-day-date-month,status and reservation_status_date , the above features represents the date the booking was made and the expected arrival date ,therefore , there is not necessity for the mentioned features as there is a feature that represents the period between the booking date and the arrival date which is "lead time". Another column that was removed is "previous-booking-not-cancelled" and it was removed because same information is given by the feature "previous-cancelation" similarly the "distribution-channel" is repeated information ,which is given by the "market-segment" feature.

1) *Categorical features encoding*

Many practical datasets consist of categorical variables, which makes it challenging for analysis and some machine learning algorithms, even though some machine learning algorithms support categorical values. Therefore, it is very important to turn the categorical attributes into numerical. There are several methods for converting categorical attributes into numerical such as label encoding , one-hot-encoding ,custom binary encoding and so on , however, for this study the one-hot-encoding is used as it is the most efficient for encoding multiple features ,and as the dataset contains six categorical features at this stage and by applying one-hot-encoding or dummy variable the categorical variable, will be converted into multiple binary columns , this will add more dimensionality but it is the more appropriate for this case, as we are trying to avoid any data leakage. The process is shown in the table below for one attribute within the dataset, which is reserved-room-type.

Table II.1:one hot encoding

Reserved-room-type
A
B
C
D

One hot encoding

Reserved-room-type-A	Reserved-room-type-B	Reserved-room-type-C	Reserved-room-type-D
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

When the process of encoding the categorical variables is completed , further adjustment can be done to reduce dimensionality by removing the dummy variable, for example by removing the first column (Reserved-room-type-A), no information will be lost , as it will be represented by the zeros in the other columns ,an example of the process is illustrated in table(II.2).

Table II.2:dummy variable removal

Reserved-room-type-A	Reserved-room-type-B	Reserved-room-type-C	Reserved-room-type-D
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Dummy variable removal

Reserved-room-type-B	Reserved-room-type-C	Reserved-room-type-D
0	0	0
1	0	0
0	1	0
0	0	1
0	0	0

2) *Outlier data*

One of the things that can affect machine learning modelling is outliers, from stastics point of view outliers is an observation points far from another observation point [10], as it shown in fig(II.3) the "adr" average daily rate has big outlier , which is an observation point higher than

5000 whereas the other observation point lies around 200, therefore , it is very important to solve the issue. One of the most straightforward techniques to identify outlier is by using the (1.5×IQR) rule, this rule is based on inter-quantile range and the inter-quintile range is computed by using equation(1) and it describes , how the middle half of the data is spread-out .

IQR=Q3-Q1 Equation 1

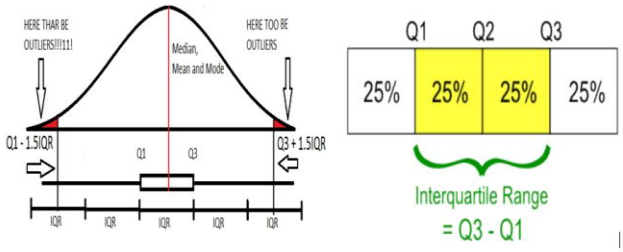

Figure II.2: IQR and 1.5IQR rule [11]

The outlier is handled by, first computing the IQR score and then filtering out the invalid values, for more details about the process, see the code in appendix (2) under the section “handling outlier”.

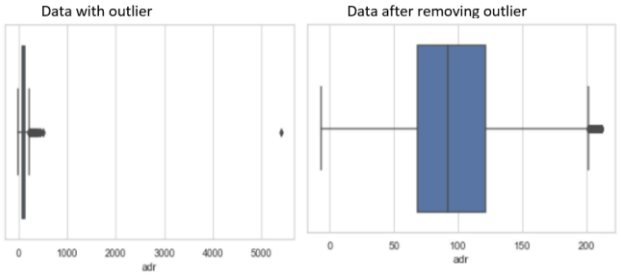

Figure II.3: adr outlier plot before and after removal

3) Class imbalance

As it is shown in fig(II.1), the data is slightly imbalanced ,class imbalance can cause a biased decision boundary towards the majority class , which leads to low and poor recognition of the minority class .In addition to learning complexity , class imbalance has also various difficulties such as small sample size which means that, the sample of minority is insufficient to train the classfier as well as (small sample disjunct), which is the representation of minority class by number of sub concepts which leads to the lack of uniform[12]. Even though, the data is not highly imbalanced, it is good to take precautions, to avoid the above mentioned difficulties ,therefore ,three different balancing techniques are deployed on this study to balance the data, whose are smotenc_oversampling ,random oversampling and random under sampling.

a) Smotenc_oversampling

Smote oversampling is method based on selecting on nearest neighbor to create synthetic instance which is governed by Euclidean distance , however , smote only works with nominal or continues features ,therefore , in the case of categorical or binary or a mixture of nominal and binary values an extension to smote method is used , which is “SMOTENC” .Smotenc allows for the use of binary or categorical by selecting the value with highest occurrence of nearest neighbors to minority class[13,14].

4) Random oversampling

Random oversampling technique is performed by random duplicating instance from the minority class to match the number of the majority class, on this study random oversampling was used to balance the data.

5) Random undersampling

The random undersampling is performed by randomly selecting instances from the majority class to be deleted to match the minority class , this class can be repeated until aimed class distribution is accomplished as an equal number for both classes , this was performed to balance the dataset

6) Data Scaling

For the purpose of data scaling, two data scaling methods were initially tested, the standard scaler and MinMax scaler, and it was found that the MinMax scaler performed better with the dataset ,therefor ,the date was scaled using minmax scaler .

B. Dimensionality Reduction

Dimensionality reduction is very important feature engineering techniques for data with higher dimensionality , some of the advantages of dimensionality reduction are fast data processing, speed by reducing the amount of the data while minimizing the loss of information, because in terms of processing , processing a data with ten dimension is not the same as processing a data with two or three dimension , another advantage of dimensionality reduction is visualization , it enables the user to visualize fewer dimension based on the analysis requirement. There are many feature reduction methods such as linear discernment analysis (LDA), principle component analysis (PCA), kernel principle component analysis (KPCA) and so on. However, for this study “PCA” is considered.

The LDA was not considered because LDA assume that the data follows the gaussian distribution, but the data of this study is not normally distributed.

1) Principle component analysis

Principle component analysis simplifies the data set in logical and useful way, this is done by identifying the features variance and sorting them by the highest variance to eliminate the redundant data. This method was implemented by taking advantage of python reach libraries.

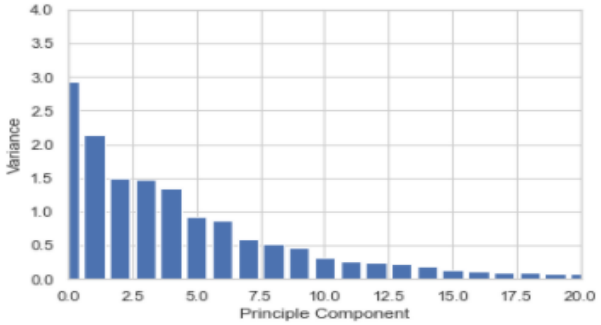

Figure II.4: PCA Individual component variance

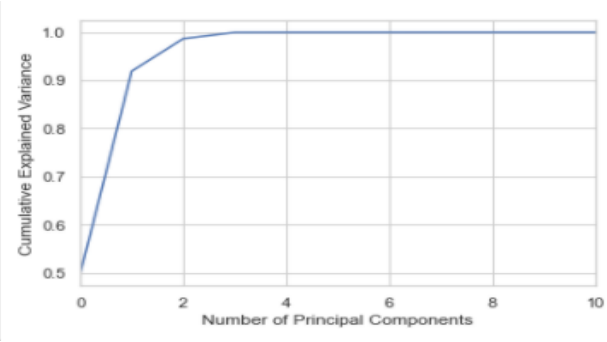

Figure II.5: Cumulative variance

The results above presented in fig(II.5, II.6) for individual component variance and cumulative component variance , it can observed that , the first two components are highly important , and from fig(II.6) we observe that two components explains 98% ,therefore, for the machine learning models , pca with two components is used.

III. MACHINE LEARNING MODELS

A. *Logistic regression*

Logistic regression is one of the techniques used to analyses a binary response data , the logistic regression is also used for classification , which in this study the logistic regression classfier is used, which takes a linear combination of more than one attribute or variable as argument of the sigmoid function ,the output of the sigmoid function is a value between 0 and 1 and the middle number is considered as the threshold to identify the outcome ,for instance if the output is greater than 0.5 is class 1 and if it is less than 0.5 class 0[15].

B. *K-nearest Neighbor (K-NN)*

The k-nearest neighbor algorithm compares the unknow instance to those belonging in the training dataset and classify it to a class based on the majority voting of nearest or by distance weighing, distance can be computed by using the Euclidean metrices [15], however , all the parameters are computed using the grid search algorithm in python .

C. *Random forest classfier*

The random forest classfier is combination of multiple tree classfier, the generation of classfier by randomly sampled independent vector from the input vector and final output of the classfier is the most popular class that is classified by the unit vote of the trees or mode.

D. *Gausian Naive bias classfier*

The gaussian naïve Bayes classfier works in the essence that takes each data point and assign it to the nearest class, however, instead of computing the nearness using the Euclidean distance, it also considers the class variance.

E. *XGB classfier*

The XGB classfier or eXtreme Gradient boosting is a boosting algorithms which is based on gradient boosted decision trees classfiers ,one of the advantages of XGB is ,it applies the best regularization methods to minimize overfitting .The XGB classfier is an open source library that provides an algorithm for machine learning under the gradient boosting technique and it is scikit-learn API, which is used for this study.

IV. RESULTS AND DISCUSSION

A. *Machine learning Models Evaluation*

The models were evaluated in terms of the techniques:

- 5 cross-validation
- Accuracy
- Precision
- Specify
- F1 measure
- Confusion matrix
- ROC_AUC curve

B. *Models results discussion*

The five machine learning models presented under the machine learning section were implemented in jupyter notebook using python libraries and each model was trained and evaluated.

For the purpose of obtaining the most accurate results, the models were trained using parameters, that were computed using grid search, as it was discussed in the previous section, the dataset was slightly imbalance. Therefore, each model was trained three times, with the three different imbalance methods results whose are smotenc_oversampling, random_oversampling and random_undersampling to evaluate and determine which class imbalance is more appropriate for the dataset.

1) *LogisticRegression*

The logistic regression was trained ,tested and validated ,the obtained results are presented in fig(IV.1, IV.2) and table(IV.1) , the parameter was computed ,using grid search. The smotenc class imbalance technique delivered a better result with LogisticRegression.

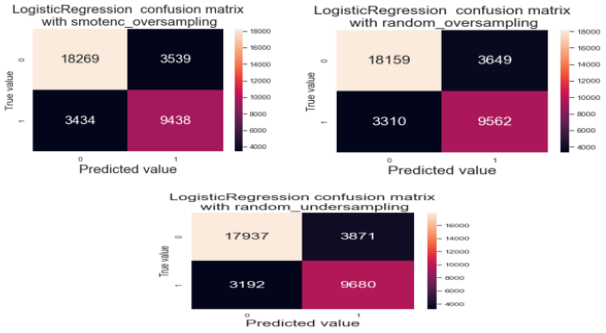

Figure IV.2: LogisticRegression Confusion matrix

Table IV.2 :LogisticRegression results

LogisticRegression			
Model parameters			
Max_iteration	10	10	10
Results			
Class Imbalance	SMOTENC_OVERSAMPLING	Random_oversampling	Random_undersampling
Cross validation 5-folds			
Accuracy	0.77	0.75	0.75
Precision	0.80	0.78	0.78
Recall/Sensitivity	0.71	0.69	0.68
F1 Measure	0.75	0.73	0.73
ROC_AUC	0.82	0.83	0.83

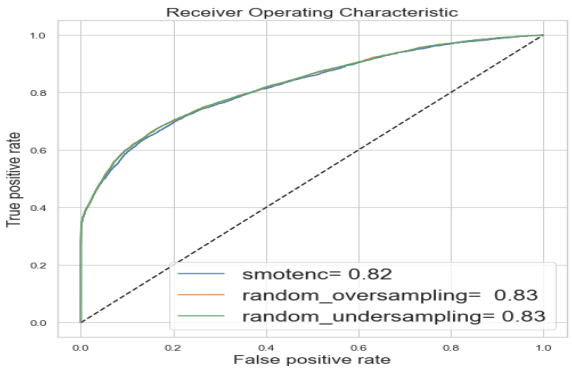

Figure IV.3: LogisticRegression ROC curve

2) K-nearest neighbor classifier

The k-nearest neighbor classfier delivered the best results in terms of accuracy, precision and sensitivity with random oversampling , in general the performance of this model is the highest among all the models that have been implemented. The parameters were computed using grid search , therefore ,it was observed that grid search gave different parameters for each technique ,for further details of how the model was tuned and implemented see appendixes(2).

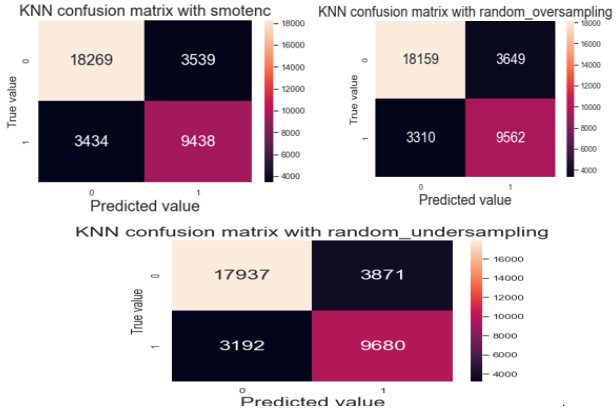

Figure IV.4: K-nearest neighbor classifier confusion matrix

Table IV.3: K-nearest neighbor classifier results and parameters

K-Nearest Neighbor			
Class Imbalance	SMOTENC_OVERSAMPLING	Random_oversampling	Random_undersampling
Model parameters			
n_neighbors	29	27	25
metric	minkowski	manhattan	minkowski
weights	distance	distance	distance
Results			
Cross validation 5-folds			
Accuracy	0.82	0.87	0.79
Precision	0.83	0.83	0.80
Recall/Sensitivity	0.78	0.92	0.79
F1 Measure	0.82	0.87	0.71
ROC_AUC	0.91	0.96	0.88

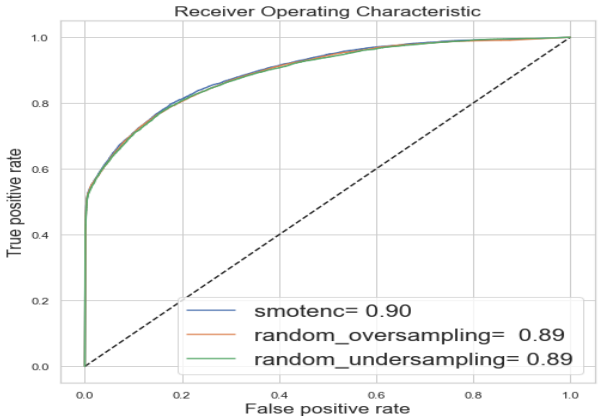

Figure IV.5 :K-nearest neighbor ROC curve

3) Gaussian Naive bayes classifier

The gaussian naïve Bayes classifier was implemented the same way the other models was, with data that was balanced with three different balancing method, however, as it is illustrated in the results figures below, the GNB classifier has delivered the poorest performance in all cases.

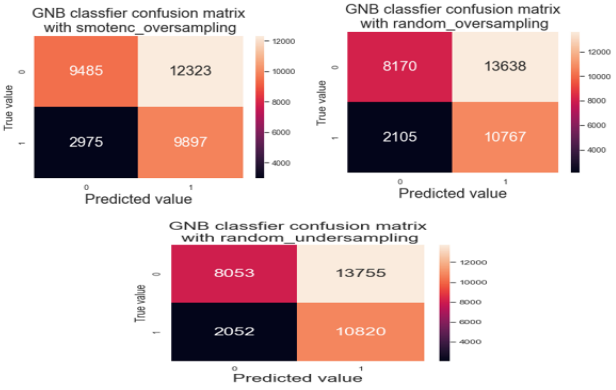

Figure IV.6: GNB confusion matrix

Table IV.4:GNB results

Gaussian Naive Bayes classifier			
Results			
Class Imbalance	SMOTENC_OVERSAMPLING	Random_oversampling	Random_undersampling
Cross validation 5-folds			
Accuracy	0.61	0.60	0.60
Precision	0.58	0.57	0.57
Recall/Sensitivity	0.78	0.83	0.83
F1 Measure	0.66	0.67	0.67
ROC_AUC	0.67	0.66	0.65

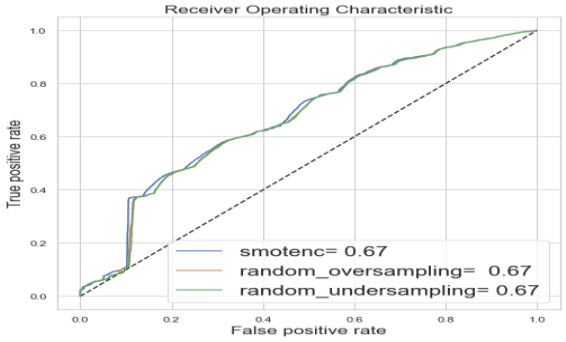

Figure IV.7: GNB ROC curve

4) Random forest classifier

Similarly, the random forest classifier was trained ,tested and validated three times and parameters were computed for all the three times as it is shown in table(IV.5), it was also evaluated three times using (cross_val_score) ,confusion matrix and roc curve the results for the evaluations are presented in fig(**IV.8, IV.9**), the model performance ,overall is acceptable, based on the obtained results.

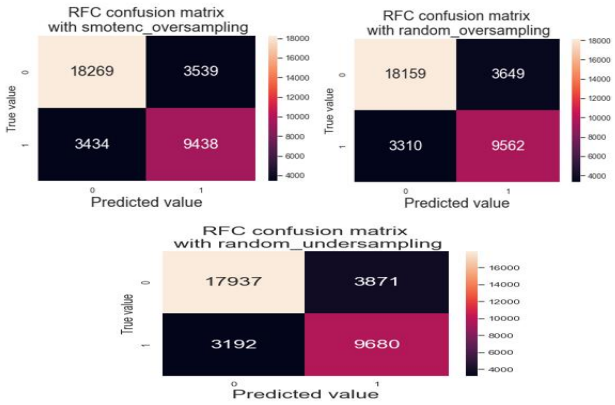

Figure IV.9: random forest classifier confusion matrix

Table IV.6:random forest classifier results and parameters

Random forest Classifier			
Class Imbalance	SMOTENC_OVERSAMPLING	Random_oversampling	Random_undersampling
Model parameters			
n_estimators	700	200	200
max_features	log2	sqrt	auto
Results			
Cross validation 5-folds			
Accuracy	0.82	0.88	0.78
Precision	0.83	0.85	0.79
Recall/Sensitivity	0.80	0.91	0.77
F1 Measure	0.81	0.88	0.78
ROC_AUC	0.90	0.95	0.87

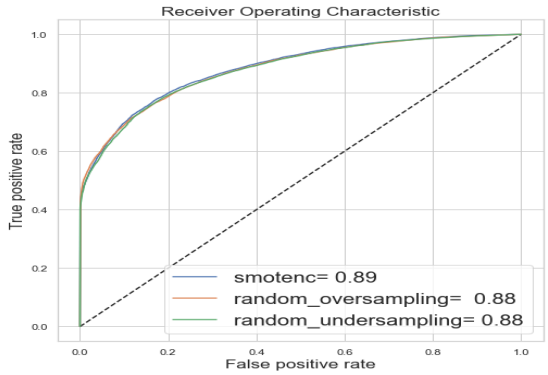

Figure IV.10:random forest classifier ROC curve

5) XGB classifier

The XGB classifier was implemented and evaluated in the same manner as the other models ,and from the obtained results ,it can be observed that this model performance close to the random forest classifier performance ,the model performed better with data balanced with smotenc balancing technique.

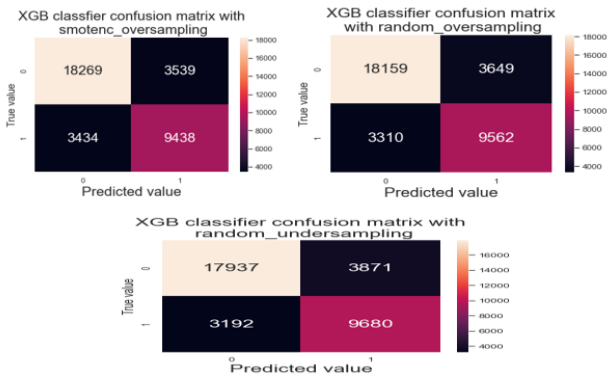

Figure IV.11:XGB classifier confusion matrix

Table IV.7:XGB classifier results and parameters

XGB Classifier			
Class Imbalance	SMOTENC_OVERSAMPLING	Random_oversampling	Random_undersampling
Model parameters			
n_estimators	180	180	140
max_depth	9	9	9
learning_rate	0.1	0.1	0.1
Results			
Cross validation 5-folds			
Accuracy	0.80	0.82	0.78
Precision	0.82	0.82	0.80
Recall/Sensitivity	0.78	0.76	0.74
F1 Measure	0.80	0.79	0.77
ROC_AUC	0.88	0.89	0.86

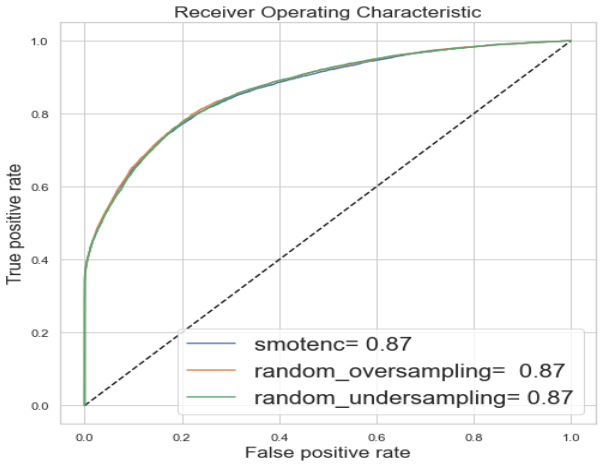

Figure IV.12;XGB classifier ROC curve

V. CONCLUSION AND FUTURE WORK

This study showed the importance and benefits of having a system that can help the demand-management make decisions in the hospitality industry. From machine learning point of view , it was observed that, the K-NN algorithm ,random forest classfier and XGB classifier delivered similar good performance, at the same it was observed that the gaussian naïve Bayes classifier delivered the poorest performance. In comparison between the three balancing technique that used , it was seeing that ,the machine learning models performed better with the dataset balanced with "smotenc oversampling" and "random oversampling".

In the future , more models can be developed to improve the performance such artificial neural network and a prototype machine learning system.

VI. REFERENCE

[1] Antonio, N., Almeida, A., & Nunes, L. (2019). Hotel booking demand datasets. *Science Direct*, *22*. Retrieved 8 November 2020, from https://www.sciencedirect.com/science/article/pii/S2352340918315191.Sánchez-Medina, A., & C-Sánchez, E. (2020). Using machine learning and big data for efficient forecasting of hotel booking cancellations. *International Journal Of Hospitality Management*, *89*, 102546. https://doi.org/10.1016/j.ijhm.2020.102546

[2] Haensel, A., & Koole, G. (2011). Booking horizon forecasting with dynamic updating: A case study of hotel reservation data. *International Journal Of Forecasting*, *27*(3), 942-960. https://doi.org/10.1016/j.ijforecast.2010.10.004

[3] *UNWTO Tourism Highlights2018 Edition*. World Tourism Organization (UNWTO). (2018). Retrieved 11 November 2020, from https://www.e-unwto.org/doi/book/10.18111/9789284419876.

[4] Chow, W., Shyu, J., & Wang, K. (1998). Developing a Forecast System for Hotel Occupancy Rate Using Integrated ARIMA Models. *Journal Of International Hospitality, Leisure & Tourism Management*, *1*(3), 55-80. https://doi.org/10.1300/j268v01n03_05

[5] Chu, F. (2009). Forecasting tourism demand with ARMA-based methods. *Tourism Management*, *30*(5), 740-751. https://doi.org/10.1016/j.tourman.2008.10.016

[6] Sun, S., Wei, Y., Tsui, K., & Wang, S. (2019). Forecasting tourist arrivals with machine learning and internet search index. *Tourism Management*, *70*, 1-10. https://doi.org/10.1016/j.tourman.2018.07.010

[7] Antonio, N., de Almeida, A., & Nunes, L. (2017). Predicting Hotel Bookings Cancellation With a Machine Learning Classification Model. *IEEE Xplore*, *6*(0-7695-6321-X/17/31.00). https://doi.org/10.1109/ICMLA.2017.00-11

[8] Refaat, M. (2010). *Data Preparation for Data Mining Using SAS* (1st ed., pp. 171-173). Elsevier Science.

[9] Ye, A. (2020). *Stop One-Hot Encoding Your Categorical Variables.*. Medium. Retrieved 23 November 2020, from https://towardsdatascience.com/stop-one-hot-encoding-your-categorical-variables-bbb0fba89809.

[10] Sharma, N. (2018). *Ways to Detect and Remove the Outliers*. Medium. Retrieved 19 November 2020, from https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba.

[11] N.d. (2017). *Inter-Quartile Range,*. Sfu.ca. Retrieved 20 November 2020, from https://www.sfu.ca/~jackd/Stat203_2011/Wk02_1_Full.pdf.

[12] Wozniak, M. *Hybrid Classifiers* (1st ed., pp. 70-75). Springer.

[13] Allen, M. (2020). *SMOTENC – Python for healthcare modelling and data science*. Python for healthcare modelling and data science. Retrieved 25 November 2020, from https://pythonhealthcare.org/tag/smotenc/.

[14] Aguilar, F. (2019). *SMOTE-NC in ML Categorization Models fo Imbalanced Datasets*. Medium. Retrieved 18 November 2020, from https://medium.com/analytics-vidhya/smote-nc-in-ml-categorization-models-fo-imbalanced-datasets-8adbdcf08c25.

[15] UrsoAntonino, A., Massimo, F., & Valentina Ravì, R. (2019). *Data Mining: Prediction Methods* (1st ed., pp. 413-430). ICAR-CNR.

[16] N.d. (2020). *How the Naive Bayes Classifier works in Machine Learning*. Dataaspirant. Retrieved 17 November 2020, from https://dataaspirant.com/naive-bayes-classifier-machine-learning/.

VII. APPENDIX 1

Table VII.1:dataset features and description

Feature	Data type	Describtion
hotel	object	Name of the hotel
Is_canceled	integer	Is the booking cancelled or not
Lead_time	integer	The time interval between booking data and expected arrival
arrival_date_year	integer	The expected arrival year
arrival_date_month	object	The expected arrival month of the year
arrival_date_week_number	integer	The expected arrival week number
arrival_date_day_of_month	integer	The expected arrival day of the month
stays_in_weekend_nights	integer	Stay during weekends
stays_in_week_nights	integer	Stay during weekdays
adults	Integer	Customer age class (adults)
children	float	Number of children accompanied with customer
babies	integer	Number of babies accompanied with customer
meal	object	Number of meals included within the booking
country	object	Customer origin country
market_segment	object	Type of customer example(direct , corporate)
distribution_channel	object	Type of customer example(direct , corporate)
is_repeated_guest	integer	Is the customer repeated guest
previous_cancellations	integer	Number of previous cancelations for the customer
previous_bookings_not_canceled	integer	Number of previous booking that is not cancelled
reserved_room_type	object	Type of room booked by customer
assigned_room_type	object	Type of room assigned to customer upon arrival
booking_changes	integer	Number of booking changes by customer
deposit_type	object	Deposit type (deposit , no deposit)
agent	float	Is the
company	float	Is the customer through company
days_in_waiting_list	integer	How many days was the customer in the waiting list
customer_type	object	Customer type example (transient , contract and so on)
adr	float	Average daily rate
required_car_parking_spaces	integer	Number of required parking spaces
total_of_special_requests	integer	Total special requests by customer
reservation_status	object	Reservation status(checked out , canceled)
reservation_status_date	object	The date of reservation status

1. Importing the Required libraries

```
]: import pandas as pd
from sklearn import preprocessing
from sklearn import svm

import itertools
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
import statsmodels.formula.api as smf

from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, confusion_matrix, auc
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA,KernelPCA

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn import preprocessing
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import confusion_matrix
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import plot_confusion_matrix
import plotly.express as px
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from imblearn.under_sampling import NearMiss # doctest: +NORMALIZE_WHITESPACE
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import learning_curve
from sklearn.metrics import matthews_corrcoef

import sklearn.metrics as metrics

from warnings import filterwarnings
filterwarnings('ignore')
%matplotlib inline
from matplotlib.pyplot import show
```

2.Data Pre-procssing

2.1 Loading Dataset

In [ ]:
```
#Loading data
df = pd.read_csv('hotel_bookings.csv')
```

2.2 cleaning and EDA

- **To have a basic undersatnding for the dataset whcih is very important ,needs to know its shaape such as columns and rows**

----------The sahpe of the dataset is shown below----------

In [ ]:
```
df.shape                        # getting the shape of the dataset
```

In [ ]:
```
print(df.columns.values)        # printing the dataset columns
```

It is very importnat to check how the features in the dataset are correlated and to do so the code in the cell below is used

In [ ]:
```
fig,axes = plt.subplots(1,1,figsize=(20,20))
correlation_mat = df.corr()

sns.heatmap(correlation_mat, annot = True,annot_kws={'size':20})

plt.show()
```

The dataset has two classes whose are

- **Canceled booking which is rerprents by 1**
- **Not_Canceled booking which is rerprents by 0**

and to see the percentage of each class the code below used

In [ ]:
```
print('Not camcelled', round(df['is_canceled'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('cancelled ', round(df['is_canceled'].value_counts()[1]/len(df) * 100,2), '% of the dataset')
```

The dataset is a combination of two hotels dataset

- **City hotel**
- **Resort hotel**

The code in the cell below is used to plot a barplot for both hotels separatley for canceled and not_canceled bookings

In [ ]:
```
cancelprediction=df.groupby(["hotel","is_canceled"]).lead_time.count().reset_index()
cancelprediction.columns=["hotel","is_canceled","count"]
ax = sns.barplot(x="hotel", y="count", hue="is_canceled", data=cancelprediction)
```

In [ ]:
```
df.info()     # The info function is used to have information about the features such as datatype
```

2.3 allocating and handling missing values

2.3.1 allocating missing values

In [ ]:
```
print("Number of NaN in each columns:", df.isnull().sum(), sep='\n')
```

In [ ]:
```
df.isna().sum()
```

Four columns have Nan values , whose are:

- **Chlidren with 4 nan values**
- **Country with 488 nan values**
- **agent with 1640 nan values**
- **Company with 112593 nan values**

The code in the cell below is to check the data manually for ('?',etc)

In [ ]:
```
for i in df.columns:
    print(i+' -> \n',df[i].unique())
    print('\n','-_'*25)
```

The code in the cell below is to convert 'reservation_status_date' column to seriestime

In [ ]:
```
df['reservation_status_date'] = pd.to_datetime(df['reservation_status_date'])
```

checking for the country that occurs most in the country column

In [ ]:
```
labels = df['country'].astype('category').cat.categories.tolist()
counts = df['country'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True) #autopct is show the % on plot
ax1.axis('equal')
plt.show()
```

2.3.2 handiling misssing values

the missing values in the agent column is replaced by the mean value as it is shown below

```
In [ ]: df['agent'].fillna(value= df['agent'].mean(),inplace=True)
        df['agent'].isnull().sum()
```

PRT is the most country that has occurance with occurance of 40.9% therefore the null is replaced by PRT

```
In [ ]: df.country.fillna(value='PRT', inplace=True)
```

handling missing values in children column

```
In [ ]: df.children.fillna(value=0.0, inplace=True)
```

```
In [ ]: #cheking again for missing values
        df.isnull().sum()
```

replacing month name by its equivalent month number

```
In [ ]: df['arrival_date_month'] = df['arrival_date_month'].replace
        (['January','February','March','April','May','June','July','August','September','October','November','December'],
         ['1','2','3','4','5','6','7','8','9','10','11','12'])
```

dropping the unnecessary columns for the study

```
In [ ]: df.drop(['arrival_date_year','arrival_date_month','arrival_date_week_number','reservation_status','arrival_date_day_of_month',
                'reservation_status_date','company','previous_bookings_not_canceled' ,'country'], axis=1, inplace=True)
```

```
In [ ]: df.head()
```

```
In [ ]: #computing columns with null values
        null_columns={}
        for i in df.columns:
            if df[i].isnull().sum()>0:
                null_columns[i]=df[i].isnull().sum()


        null_columns
```

Plotting the two classes cancleled vs not canceled with percenatge using seaborn

```
In [ ]: # plotting canceled booking versus not canceled with percentage
        sns.set(style="whitegrid")
        plt.figure(figsize=(8,5))
        total = float(len(df))
        ax = sns.countplot(x="is_canceled", hue="is_canceled", data=df)
        plt.title('canclled vs not cancelled ', fontsize=20)
        for p in ax.patches:
            percentage = '{:.1f}%'.format(100 * p.get_height()/total)
            x = p.get_x() + p.get_width()
            y = p.get_height()
            ax.annotate(percentage, (x, y),ha='center')
        plt.show()
```

2.5 handling outlier

```
In [ ]: Q1 = df['adr'].quantile(0.25)
        Q3 = df['adr'].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = (Q1 - 1.5 * IQR)
        upper_bound = (Q3 + 1.5 * IQR)
```

```
In [ ]: df1=df[(df['adr'] > lower_bound ) & (df['adr'] < upper_bound)]
```

```
In [ ]: df1['adr'].plot(kind='box',vert=False, figsize=(14,6))
```

```
In [ ]: df2=df1
```

2.4 allocating and handling categorical data

```
In [ ]: categorical = [var for var in df2.columns if df2[var].dtypes=='object']
        categorical
```

```
In [ ]: df_encoded=pd.get_dummies(df2,columns=['hotel',
         'meal',
         'market_segment',
         'distribution_channel',
         'reserved_room_type',
         'deposit_type',
        'assigned_room_type',
         'customer_type'],drop_first=True)
```

In [ ]:
```
# wcheck if there is still category data
categorical = [var for var in df_encoded.columns if df_encoded[var].dtypes=='object']
categorical
```

3.Balancinng dataset

Smote oversampling

3.1 Split data into training amd testing

In [ ]:
```
X=df_encoded.drop('is_canceled',axis=1)
Y=df_encoded['is_canceled']
print(X.shape)
print(Y.shape)
```

In [ ]:
```
#splitting data  to training and testing datset
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
```

In [ ]:
```
df_encoded.head()
```

3.2 Smote_oversampling

In [ ]:
```
from imblearn.over_sampling import SMOTENC
sm = SMOTENC([15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,
              36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54], random_state=0)
X_trian_smotenc, y_train_smotenc = sm.fit_sample(X_train, y_train.ravel())
```

In [ ]:
```
from collections import Counter
print("Before SMOTE:",Counter(y_train))
print("After SMOTE:",Counter(y_train_smotenc))
```

3.3 Random_oversampling

In [ ]:
```
ros = RandomOverSampler(random_state=0)
X_train_ros, y_train_ros = ros.fit_sample(X_train, y_train.ravel())
```

In [ ]:
```
print("Before ROS:",Counter(y_train))
print("After ROS:",Counter(y_train_ros))
```

3.4 Random_unsdersampling

In [ ]:
```
rus = RandomUnderSampler(random_state=0)
```

In [ ]:
```
X_trian_rus, y_train_rus = rus.fit_sample(X_train, y_train.ravel())
```

In [ ]:
```
print("Before RUS:",Counter(y_train))
print("After RUS:",Counter(y_train_rus))
```

4.Feature engineering

In [ ]:
```
#from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(-1,1)).fit(df_encoded)
df_encoded = scaling.transform(df_encoded)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_encoded)
```

In [ ]:
```
model = PCA()
model.fit(X)
```

In [ ]:
```
plt.plot(np.cumsum(model.explained_variance_ratio_))
plt.xlim(0,10,1)
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
```

In [ ]:
```
np.cumsum(model.explained_variance_ratio_)[2]
```

In [ ]:
```
plt.bar(range(model.n_components_), model.explained_variance_)
plt.xlabel('Principle Component')
plt.ylabel('Variance')
plt.xlim([0,20])
plt.ylim([0,4])
plt.show()
```

4.Prediction models

==========Models Evaluation Function=========

```
[ ]: def Evaluation(y_test,y_pred,title):
    # Build confusion metrics
    cm = confusion_matrix(y_test, y_pred)
    # Plot confusion matrix in a beautiful manner
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax, fmt = 'g'); #annot=True to annotate cells
    # labels, title and ticks
    ax.set_xlabel('Predicted value', fontsize=20)
    ax.xaxis.set_label_position('bottom')
    plt.title(title, fontsize=20)

    ax.set_ylabel('True value', fontsize=15)
    plt.show()

    tn,fp,fn,tp=cm.ravel()
    print('accuracy:',(tp+tn)/(tn+fp+fn+tp))
    print('precision:',(tp)/(fp+tp))
    print('recall:',(tp)/(fn+tp))
```

========== Learning Curve function ==========*

```
[ ]: %%time
# Learning curve
def learningFunction(model, X_train, y_train):
    N, train_score, val_score = learning_curve(model, X_train, y_train,
                                               train_sizes = np.linspace(0.01, 1.0, 50), cv= 5, n_jobs=-1, scoring='accuracy')

    print(N)
    plt.plot(N, train_score.mean(axis=1), label = 'train')
    plt.plot(N, val_score.mean(axis=1), label = 'validation')
    plt.xlabel('Train sizes')
    plt.legend()
```

==========ROC Curve Function ==========

==========ROC Curve Function ==========

```
In [ ]: #ROC Curve
def roc_function(classifier1,classifier2,classifier3,X_test,Y_test):
    y_pred_prob1 = classifier1.predict_proba(X_test)[:,1]
    fpr1 , tpr1, thresholds1 = roc_curve(Y_test, y_pred_prob1)
    y_pred_prob2 = classifier2.predict_proba(X_test)[:,1]
    fpr2 , tpr2, thresholds2 = roc_curve(Y_test, y_pred_prob2)


    y_pred_prob3 = classifier3.predict_proba(X_test)[:,1]
    fpr3 , tpr3, thresholds3 = roc_curve(Y_test, y_pred_prob3)
    roc_auc1=metrics.auc(fpr1 , tpr1)
    roc_auc2=metrics.auc(fpr2 , tpr2)
    roc_auc3=metrics.auc(fpr3 , tpr3)
    plt.figure(figsize=(8,8))
    leg = ax.legend(prop={"size":20})
    plt.plot([0,1],[0,1], 'k--')
    plt.plot(fpr1, tpr1, label= "smotenc= {:.2f}".format(roc_auc1))
    plt.plot(fpr2, tpr2, label= "random_oversampling=  {:.2f}".format(roc_auc2))
    plt.plot(fpr3, tpr3, label= "random_undersampling= {:.2f}".format(roc_auc3))


    plt.legend(fontsize=20, loc='best')
    plt.xlabel("False positive rate",fontsize=17)
    plt.ylabel("True positive rate",fontsize=17)
    plt.title('Receiver Operating Characteristic',fontsize=17)
    plt.show()
```

```
def cvl(model,X,y):
    cvl=cross_val_score(model, X, y, cv=5, scoring='accuracy')
    cvl1=cross_val_score(model, X, y, cv=5, scoring='accuracy').mean()
    cvl2=cross_val_score(model, X, y, cv=5, scoring='precision')
    cvl3=cross_val_score(model, X, y, cv=5, scoring='precision').mean()
    cvl4=cross_val_score(model, X, y, cv=5, scoring='f1')
    cvl5=cross_val_score(model, X, y, cv=5, scoring='f1').mean()
    cvl6=cross_val_score(model, X, y, cv=5, scoring='recall')
    cvl7=cross_val_score(model, X, y, cv=5, scoring='recall').mean()
    cvl8=cross_val_score(model, X, y, cv=5, scoring='roc_auc')
    cvl9=cross_val_score(model, X, y, cv=5, scoring='roc_auc').mean()

    print('accuracy:',cvl)
    print('==========================================================')
    print('accuracy_mean:',cvl1)
    print('==========================================================')
    print('precision:',cvl2)
    print('==========================================================')
    print('precision_mean:',cvl3)
    print('==========================================================')
    print('f1:',cvl4)
    print('==========================================================')
    print('f1_mean:',cvl5)
    print('==========================================================')
    print('recall/sensitivity:',cvl6)
    print('==========================================================')
    print('recall/sensitivity_mean:',cvl7)
    print('==========================================================')
    print('roc_auc:',cvl8)
    print('==========================================================')
    print('roc_auc_mean:',cvl9)
    print('==========================================================')
```

4.1 K-NN classfier

4.1.1 Hyperparameter tuning for K-NN classfier function

```
%%time
def KNN_param(X_trian, y_train):
    model_KNN = make_pipeline(MinMaxScaler(), PCA(), KNeighborsClassifier() )

    params = {
         'kneighborsclassifier__n_neighbors': np.arange(1, 30),
         'pca__n_components': [2],
         'kneighborsclassifier__metric': ['euclidean', 'manhattan', 'minkowski','chebyshev'],
         'kneighborsclassifier__weights': ['uniform', 'distance']
    }

    grid_KNN= GridSearchCV(model_KNN, param_grid = params, cv = 5, n_jobs=-1)
    grid_fit_KNN = grid_KNN.fit(X_trian, y_train)
    print('best parameters for KNN:',grid_fit_KNN.best_params_)
```

4.1.2 K-NN Classifier with smote oversampling data

== Compute the parameters for the KNN classfier for smote_oversampling data by calling the function ==

```
[ ]: KNN_param(X_trian_smotenc, y_train_smotenc)
```

The parameters used for this pipeline ara computed in the previuos section using gridsearch

```
[ ]: ## making pipline with the the obatined parameters using gridsearch
KNN_pipe_smote1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),
                        KNeighborsClassifier(n_neighbors=29, metric= 'minkowski' , weights='distance'))
```

```
[ ]: KNN_pipe_smote1.fit(X_trian_smotenc, y_train_smotenc)     # training the model
y_pred1 =KNN_pipe_smote1.predict(X_test)              # making predictions
```

```
[ ]: Evaluation(y_test, y_pred1 , 'KNN confusion matrix with smotenc')                    # Evaluate the model by calling the evalu
ation function
```

=========Evaluating model cross_val_score=====

```
[ ]: cvl(KNN_pipe_smote1,X_trian_smotenc, y_train_smotenc)
```

4.1.3 K-NN Classifier with Random oversampling data

== Compute the parameters for the KNN classfier for random_oversampling data by calling the function ==

```
[ ]: KNN_param(X_train_ros, y_train_ros)
```

```
[ ]: ## making pipline with the the obatined parameters using gridsearch for data with random oversampling
KNN_pipe_ros1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),
                        KNeighborsClassifier(n_neighbors=27, metric= 'manhattan' , weights='distance'))
```

```
[ ]: KNN_pipe_ros1.fit(X_train_ros, y_train_ros)     # training the model
y_pred2 =KNN_pipe_ros1.predict(X_test)              # making predictions
```

```
[ ]: Evaluation(y_test, y_pred2, 'KNN confusion matrix with random_oversampling')                    # Evaluate the model by callin
g the evaluation function
```

```
[ ]: cvl(KNN_pipe_ros1,X_train_ros, y_train_ros)
```

4.1.4 K-NN Classifier with Random undersampling data

== Compute the parameters for the KNN classfier for random_undersampling data by calling the function ==

```
[ ]: KNN_param(X_trian_rus, y_train_rus)
```

```
[ ]: ## making pipline with the the obatined parameters using gridsearch for data with random undersampling
KNN_pipe_rus1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),
                        KNeighborsClassifier(n_neighbors=25, metric= 'minkowski' , weights='distance'))
```

```
[ ]: KNN_pipe_rus1.fit(X_trian_rus, y_train_rus)     # training the model
y_pred3 =KNN_pipe_rus1.predict(X_test)              # making predictions
```

```
[ ]: Evaluation(y_test, y_pred3, 'KNN confusion matrix with random_undersampling')                    # Evaluate the model by calli
ng the evaluation function
```

```
[ ]: cvl(KNN_pipe_rus1,X_trian_rus, y_train_rus)
```

======ROC_AUC curve for KNN=====

```
[ ]: roc_function(KNN_pipe_smote1,KNN_pipe_ros1,KNN_pipe_rus1,X_test,y_test)
```

4.2 Random Forest Classfier

4.2.1 Hyperparameter tuning for Random Forest classfier function

```
[ ]: %%time
def RFC_param(X_train, y_train):
    model_RFC = make_pipeline(MinMaxScaler(), PCA(), RandomForestClassifier(n_jobs=-1,oob_score = True))

    params = {
         'randomforestclassifier__n_estimators': [200, 700],
         'pca__n_components': [2],
         'randomforestclassifier__max_features': ['auto', 'sqrt', 'log2'],
    }

    grid_RFC= GridSearchCV(model_RFC, param_grid = params, cv = 5, n_jobs=-1)
    grid_RFC.fit(X_train, y_train)
    print('best parameters for RFC:',grid_RFC.best_params_)
```

4.2.2 Random forest Classifier with smote oversampling data

== Compute the parameters for the RFC classfier for smote oversampling data by calling the function ==

```
[ ]: RFC_param(X_trian_smotenc, y_train_smotenc)
```

```
[ ]: ## making pipline with the the obatined parameters using gridsearch
RFC_pipe_smote1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),
                        RandomForestClassifier(n_estimators=700,max_features='log2',n_jobs=-1,oob_score = True))
```

```
[ ]: RFC_pipe_smote1.fit(X_trian_smotenc, y_train_smotenc)     # training the model
y_pred1 =RFC_pipe_smote1.predict(X_test)              # making predictions
```

```
[ ]: Evaluation(y_test, y_pred1, 'RFC confusion matrix\n with smotenc_oversampling')                    # Evaluate the model by cal
ling the evaluation function
```

```
[ ]: #evaluate the model using cross_val_score
cvl(RFC_pipe_smote1,X_trian_smotenc, y_train_smotenc)
```

4.2.3 Random forest Classifier with random oversampling data

```
[ ]: RFC_param(X_train_ros, y_train_ros)
```

```
[ ]: ## making pipline with the the obatined parameters using gridsearch
RFC_pipe_ros1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),
                        RandomForestClassifier(n_estimators=200,max_features='sqrt',n_jobs=-1,oob_score = True))
```

```
[ ]: RFC_pipe_ros1.fit(X_train_ros, y_train_ros)     # training the model
y_pred2 =RFC_pipe_ros1.predict(X_test)              # making predictions
```

```
[ ]: Evaluation(y_test, y_pred2, 'RFC confusion matrix\n with random_oversampling')                    # Evaluate the model by call
ing the evaluation function
```

```
[ ]: #evaluate the model using cross_val_score
cvl(RFC_pipe_ros1,X_train_ros, y_train_ros)
```

4.2.3 Random forest Classifier with random undersampling data

```
[ ]: RFC_param(X_trian_rus, y_train_rus)
```

```
[ ]: ## making pipline with the the obatined parameters using gridsearch
RFC_pipe_rus1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),
                        RandomForestClassifier(n_estimators=200,max_features='auto',n_jobs=-1,oob_score = True))
```

```
[ ]: RFC_pipe_rus1.fit(X_trian_rus, y_train_rus)     # training the model
y_pred3 =RFC_pipe_rus1.predict(X_test)              # making predictions
```

```
[ ]: Evaluation(y_test, y_pred3, 'RFC confusion matrix\n with random_undersampling')                    # Evaluate the model by
ling the evaluation function
```

```
[ ]: #evaluate the model using cross_val_score
cvl(RFC_pipe_rus1,X_trian_rus, y_train_rus)
```

roc_auc curve for random forest classfier

```
[ ]: roc_function(RFC_pipe_smote1,RFC_pipe_ros1,RFC_pipe_rus1,X_test,y_test)
```

4.3 NIAV Bias classifier

```
[ ]: %%time
def GNB(X_train,y_train,X_test):
    model_GNB = make_pipeline(MinMaxScaler(), PCA(), GaussianNB())

    params = {

         'pca__n_components': [2]
    }

    grid_GNB= GridSearchCV(model_GNB, param_grid = params, cv = 5, n_jobs=-1)
    grid_fit_KNN = grid_GNB.fit(X_train,y_train)
    y_pred =grid_GNB.predict(X_test)
    print("")
    print('----------------------------------------------------------------')
    print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
    print("")
    print('----------------------------------------------------------------')
    print(confusion_matrix(y_test, y_pred))
    print("")
    print('----------------------------------------------------------------')
    print(classification_report(y_test,y_pred))
    return grid_GNB, y_pred
```

4.3.1 NIAV bias classfier with smote_oversampling data

```
grid_GNB, y_pred = GNB(X_trian_smotenc, y_train_smotenc,X_test)  # call the niav bias classfier to train and test the data
Evaluation(y_test, y_pred, 'GNB classfier confusion matrix\n with smotenc_oversampling')               #call the evaluation fun
ction to evaluate the model
cvl(grid_GNB,X_trian_smotenc, y_train_smotenc)
```

4.3.2 NIAV bias classfier with random_oversampling data

```
grid_GNB1, y_pred = GNB(X_train_ros, y_train_ros,X_test)  # call the niav bias classfier to train and test the data
Evaluation(y_test, y_pred, 'GNB classfier confusion matrix\n with random_oversampling')               #call the evaluation func
tion to evaluate the model
cvl(grid_GNB1,X_train_ros, y_train_ros)    #call the cross_val_score function to evaluate model
```

4.3.2 NIAV bias classfier with random_oversampling data

```
grid_GNB2, y_pred = GNB(X_trian_rus, y_train_rus,X_test)  # call the niav bias classfier to train and test the data
Evaluation(y_test, y_pred, 'GNB classfier confusion matrix\n with random_undersampling')               #call the evaluation fun
ction to evaluate the model
cvl(grid_GNB2,X_trian_rus, y_train_rus)    #call the cross_val_score function to evaluate model
```

```
roc_function(grid_GNB,grid_GNB1,grid_GNB2,X_test,y_test)
```

4.4 XGB classfier

4.4.1 hyperparameter tunning function for XGB classfier

```
%%time
def XGB_param(X_train,y_train):
    model_XGB= make_pipeline(MinMaxScaler(),PCA(),XGBClassifier(objective= 'binary:logistic',
        nthread=4,
        seed=42))


    params = {
        'xgbclassifier__n_estimators':range(60, 220, 40),
        'xgbclassifier__max_depth': range (2, 10, 1),
        'pca__n_components':[2],
        'xgbclassifier__learning_rate': [0.1, 0.01, 0.05]
    }
    XGB_CV = GridSearchCV(model_XGB,param_grid=params,scoring = 'roc_auc', n_jobs = -1, cv = 5,verbose=True
    )


    print("")
    print('----------------------------------------------------------')
    XGB_CV.fit(X_train,y_train)
    print("")
    print('best parameters for XGBClassifier:',XGB_CV.best_params_)
```

4.4.2 XGB classfier with smote_oversampling data

== Compute the parameters for the XGB classfier for smote_oversampling data by calling the function ==

```
XGB_param(X_trian_smotenc, y_train_smotenc)
```

```
## making pipline with the the obatined parameters using gridsearch
XGB_pipe_smote1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),XGBClassifier(objective= 'binary:logistic',max_depth=9,
                                                                                  learning_rate=0.1,
                                                                                  n_estimators=180,
                                                                                  nthread=4,
                                                                                  seed=42))
```

4.4.3 XGB classfier with random_oversampling data

== Compute the parameters for the XGB classfier for random_oversampling data by calling the function ==

```
XGB_param(X_train_ros, y_train_ros)
```

```
## making pipline with the the obatined parameters using gridsearch
XGB_pipe_ros1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),XGBClassifier(objective= 'binary:logistic',max_depth=9,
                                                                                learning_rate=0.1,
                                                                                n_estimators=180,
                                                                                nthread=4,
                                                                                seed=42))
```

```
XGB_pipe_ros1.fit(X_train_ros, y_train_ros)     # training the model
y_pred2 =XGB_pipe_ros1.predict(X_test)              # making predictions
```

```
Evaluation(y_test, y_pred2, 'XGB classifier confusion matrix\n with random_oversampling')                    # Evaluate the mo
del by calling the evaluation function
```

```
cvl(XGB_pipe_ros1,X_train_ros, y_train_ros)    #call the cross_val_score function to evaluate model
```

4.4.4 XGB classfier with random_undersampling data

== Compute the parameters for the XGB classfier for random_undersampling data by calling the function ==

```
XGB_param(X_trian_rus, y_train_rus)
```

```
## making pipline with the the obatined parameters using gridsearch
XGB_pipe_rus1 = make_pipeline(MinMaxScaler(), PCA(n_components=2),XGBClassifier(objective= 'binary:logistic',max_depth=9,
                                                                                learning_rate=0.1,
                                                                                n_estimators=140,
                                                                                nthread=4,
                                                                                seed=42))
```

```
XGB_pipe_rus1.fit(X_trian_rus, y_train_rus)     # training the model
y_pred3 =XGB_pipe_rus1.predict(X_test)              # making predictions
```

```
Evaluation(y_test, y_pred3, 'XGB classifier confusion matrix with\n random_undersampling')                    # Evaluate the m
odel by calling the evaluation function
```

```
cvl(XGB_pipe_rus1,X_trian_rus, y_train_rus)    #call the cross_val_score function to evaluate model
```

```
roc_function(XGB_pipe_smote1,XGB_pipe_ros1,XGB_pipe_rus1,X_test,y_test)
```

4.5 logistic regression

4.5.1 Logistic regression hyper_parameter tuning function

```
## function to compute logistice regression paarameters
def LG_param(X_train,y_train):
    model_LG= make_pipeline(MinMaxScaler(),PCA(),LogisticRegression())


    params = {
        'logisticregression__max_iter': [10,100,1000],
        'pca__n_components':[2]

    }
    LG_CV = GridSearchCV(model_LG,param_grid=params, n_jobs = -1, cv = 5
    )


    print("")
    print('----------------------------------------------------------------')
    LG_CV.fit(X_train,y_train)
    print("")
    print('best parameters for LogisticRegression:',LG_CV.best_params_)
```

4.5.2 logisticRegression with smote_oversampling data

== Compute the parameters for the LogisticRegression for smote_oversampling data by calling the function ==

```
LG_param(X_trian_smote, y_train_smote)
```

```
## making pipline with the the obatined parameters using gridsearch
LG_pipe_smote1 = make_pipeline(MinMaxScaler(),PCA(),LogisticRegression(max_iter=10))
```

```
LG_pipe_smote1.fit(X_trian_smotenc, y_train_smotenc)     # training the model
y_pred1 =LG_pipe_smote1.predict(X_test)              # making predictions
```

```
Evaluation(y_test, y_pred1, 'LogisticRegression  confusion matrix\n with smotenc_oversampling')                    # Evaluate
the model by calling the evaluation function
```

```
cvl(LG_pipe_smote1,X_trian_smotenc, y_train_smotenc)    #call the cross_val_score function to evaluate model
```

4.5.3 logisticRegression with random_oversampling data

In [ ]:
```
LG_param(X_train_ros, y_train_ros)
```

In [ ]:
```
## making pipline with the the obatined parameters using gridsearch
LG_pipe_ros1 = make_pipeline(MinMaxScaler(),PCA(),LogisticRegression(max_iter=10))
```

In [ ]:
```
LG_pipe_ros1.fit(X_train_ros, y_train_ros)     # training the model
y_pred2 =LG_pipe_ros1.predict(X_test)             # making predictions
```

In [ ]:
```
Evaluation(y_test, y_pred2, 'LogisticRegression  confusion matrix\n with random_oversampling')                    # Evaluate t
he model by calling the evaluation function
```

In [ ]:
```
cvl(LG_pipe_ros1,X_train_ros, y_train_ros)    #call the cross_val_score function to evaluate model
```

4.5.4 logisticRegression with random_undersampling data

In [ ]:
```
LG_param(X_trian_rus, y_train_rus)
```

In [ ]:
```
## making pipline with the the obatined parameters using gridsearch
LG_pipe_rus1 = make_pipeline(MinMaxScaler(),PCA(),LogisticRegression(max_iter=10))
```

In [ ]:
```
LG_pipe_rus1.fit(X_trian_rus, y_train_rus)     # training the model
y_pred3=LG_pipe_rus1.predict(X_test)              # making predictions
```

In [ ]:
```
Evaluation(y_test, y_pred3, 'LogisticRegression confusion matrix\n with random_undersampling')                    # Evaluate t
he model by calling the evaluation function
```

In [ ]:
```
cvl(LG_pipe_rus1,X_trian_rus, y_train_rus)    #call the cross_val_score function to evaluate model
```

plot roc_auc curve for logisticregression

In [ ]:
```
roc_function(LG_pipe_smote1,LG_pipe_ros1,LG_pipe_rus1,X_test,y_test)
```