CONTENTS

Page no.

1. INTRODUCTION:

Machine Learning (ML) techniques are widely used in data analysis. A Machine Learning model considers a large amount of data to train the model for providing future trend. But a common of data is the presence of missing values. During model training if there are some missing values, then training will not be adequate. So, people use different missing values imputation methods to fill the values of the missing attributes.

1.1. DEFINITION OF MACHINE LEARNING

Machine learning brings the promise of deriving meaning from all of that data. Arthur C. Clarke famously once said, “Any sufficiently advanced technology is indistinguishable from magic.” Machine Learning is nothing it is computational learning using algorithms to learn from and make predictions on data. Here, we can use statistical models and probabilistic algorithms to answer questions so we can make informative decisions based on our data. In ML, there are two kinds of data –labeled data (Supervised Learning) and Unlabeled data (Unsupervised Learning). Machine learning plays a big role in analyzing and extracting information from data and often a problem of missing values is encountered, so in my project also machine learning algorithms are used.

The missing value is nothing it is the missing information in the dataset that contains the attributes due to the following few reasons the faulty sensors, data entry errors (incorrect data entered manually), even it can be due to mass update of a table with unwanted value, people do not respond to survey (or specific questions in a survey), species are rare and cannot be found or sampled and many more reasons. It is denoted by the values in the dataset like “NAN” or “nan”, or blank or sometimes “zero” etc. And it can be handled by using different handling methods either statistical methods or by probabilistic algorithms etc. If it is handled and the handling notation can be used by the Standard values like “Not Available” or “NA”.

The missing value is encountered for various reasons sometimes due to the unknown reasons during the process of data recording and transfer, several evident reasons such as imperfect procedures of manual data entry, equipment errors, and incorrect measurements.

1.2: IMPUTATION:

It is a technique that is used to compute the missing values with some substitute value to keep most of the data or information of the dataset. Imputation will increase the quality of data and would improvise prediction results. It also focuses on their advantages and limitations. Imputation preserves in all cases by replacing missing values with an estimated value based on other available information. Once all missing values have been imputed, the dataset can then be analyzed using standard techniques for complete data.

Therefore, a data-preprocessing framework is crucial to deal with inaccurate and incomplete data for ensuring high data quality through effective data cleansing. One important task in data preprocessing is the imputation of missing values as accurately as possible.

1.3. MOTIVATION:

Handling Missing Value is an important pre-processing steps of a ML Model. In literature, I found many missing value imputation methods. From the empirical study, I found that clustering-based methods may be useful in predicting missing values of a dataset. So, it motivates me to develop a clustering-based technique to impute missing values by considering all the similar objects into groups. Since, similar objects have a common pattern and from that pattern we can determine the missing quantity.

1.4. CONTRIBUTION:

The main contribution of this project is to find out the best Missing values imputation methods among them. The proposed method is based on a Clustering Based Imputation method. The method's performance is measured in terms of prediction accuracy and computational cost on some benchmark datasets having missing values. The effectiveness of the method is imputed in terms of prediction accuracy exhibited for gene expression, Kaggle, and UCI respiratory datasets, in association with well-known standard classifiers, viz., Logistic Regression, Support vector machine, Decision tree, K Nearest Neighbor, Random Forest.

1.5. PROBLEM DEFINITION:

Problems usually associated with missing values are efficiency loss, complexity in handling and analyzing the data with missing values, biased estimates, and, inefficient estimates. It is an unavoidable problem in terms that mosf the algorithm does not work with data sets having missing values. Research on a large scale has been conducted for so many years. It is still going on for developing advanced and refined methods for missing data imputations. In my project I found the comparison results when I impute the methods namely Clustering-based Imputation, mean, median, mode, ffill, bfill, zero and constant using classifiers like Logistic Regression, KNN, SVM, Decision Tree, and Random Forest are different. I found that the clustering-based imputation in *Random Forest* has the highest accuracy rate and F1_score than the remaining classifiers. And next, the *Decision Tree* is the 2nd one that performs better than the other classifiers.

Hence, in my project, my purposed method named “Clustering-based Missing Value Imputation Method” works well. So, it can apply to any dataset to handle the missing values in the dataset.

2. RELATED WORK:

The following is a summary of a brief description of the previous research, algorithm used, interpretation, and their implication on the current paper.

Geeta Chhabra* et al. (2019), [1] - In their paper provide a comprehensive overview of various traditional, modern, and hybrid approaches for handling missing data which are in use from last so many years to recently been developed by researchers. It also serves as a reference for researchers in selecting an appropriate method for their problem. In their paper, they have discussed different types of missing value imputation methods and their pros and cons.

Chengqi Zhang et. al. [2], (2007) In their paper they propose a new and efficient missing value imputation based on data clustering, called CRI (Clustering-based Random Imputation). In their approach, they fill up the missing values of an instance with those plausible values that are generated from the data similar to this instance using a kernel-based random method. Specifically, they first divide the dataset (exclude instances with missing values) into clusters. And then each of those instances with missing values is assigned to a cluster most similar to it. Finally, missing values of an instance A is thus patched up with those plausible values that are generated using a kernel-based method for those instances from A's cluster. Their experiments (some of them are with the decision tree induction system C5.0) have proved the effectiveness of their proposed method in missing value imputation tasks.

Aydilek et. al [3], (2013). In their study, they utilize a fuzzy c-means clustering hybrid approach that combines support vector regression and a genetic algorithm. In this method, the fuzzy clustering parameters, cluster size, and weighting factor are optimized and missing values are estimated. The proposed novel hybrid method yields sufficient and sensible imputation performance results. The results are compared with those of fuzzy c-means genetic algorithm imputation, support vector regression genetic algorithm imputation, and zero imputation.

Singh, et. al. [5], (2014)-The clustering is a process of grouping objects based on some similarity measure. In hierarchical clustering, the objects can be clustered based on a single linkage, average linkage, or complete linkage. In their paper, they proposed a hybrid approach of clustering based on AGNES and DIANA clustering algorithms, an extension to the standard hierarchical clustering algorithm. In the proposed algorithm, they used a single linkage as a similarity measure. The proposed clustering algorithm provides more consistent clustered results from various sets of cluster centroids with tremendous efficiency.

Maria Pampaka et al. [4] (2014) - In this paper, they provide a review of these advances and their implications for educational research. They illustrate the issues with an educational, longitudinal survey in which missing data was significant, but for which they were able to collect much of these missing data through subsequent data collection. They thus compare methods, that is, step-wise regression (basically ignoring the missing data) and MI models, with the model from the actual enhanced sample. The value of MI is discussed and the risks involved in ignoring missing data are considered. Implications for research practice are discussed.

Radhakrishnan et. al, [6] (2015) – In their paper investigate the exploit of a machine learning technique as a missing value imputation process for incomplete Hepatitis data. Mean/mode imputation, ID3 algorithm imputation, decision tree imputation, and proposed bootstrap aggregation-based imputation are used as missing value imputation and the resultant datasets are classified using KNN. The experiment reveals that classifier performance is enhanced when the Bagging-based imputation algorithm is used to foresee missing attribute values.

2.1: COMPARISON OF EXISTING METHODS OF MISSING VALUE IMPUTATION TECHNIQUES:

Table 1: Comparison Table

Research Paper:	Year	Algorithm Used	Brief Description of Research	The implication of the current paper
[6]	2013	Hybrid algorithm with Support Vector Regression & Genetics with fuzzy clustering	The results are compared with zero imputation, fuzzy means genetic algorithm, and support vector regression genetic algorithm.	The proposed hybrid algorithm has better accuracy than the single algorithm
[3]	2014	A hybrid approach of clustering based on an agglomerative and divisive approach	Clusters are compared with agglomerative and divisive clusters.	The proposed algorithm provides more consistent & efficient clusters
[5]	2014	Multiple Imputation Models	The results are compared with a complete case analysis	Multiple imputations are relatively easy and popular
[4]	2015	Machine learning techniques such as the ID3 algorithm, decision tree	Compared the results of ID3 and decision tree using bagging	Performance is enhanced after bagging
[2]	2007	Clustering-based Missing Value Imputation for	Compared the performances of the kernel	Extensive experimental results have

		Data Preprocessing.	function-based completion method with and without clustering and then using the well-known K-Means as a clustering algorithm for its simplicity	demonstrated the effectiveness of the kernel-based DI and SI method in making inferences for mean, variance, and distribution function after clustering.
[1]	2019	Hybrid techniques for missing value imputation along with their benefits and limitations.	Adoption of single, multiple imputations and hybrid techniques in various fields and their enhancement for better prediction	Analyze the imputation methods which are easy to use, more accurate, efficient, and yield unbiased estimates

3. PROPOSED METHOD:

I develop a method named "Clustering-based Imputation Method" with Classifiers to handle the missing values in the dataset. To discuss the proposed method, I used different symbols, and their meanings are shown in Table 2

Table 2: Meanings of symbols used.

Symbols	Meanings
D	Dataset
Ci	Clusters
C	Class
fi	Column/feature

Steps:

1. For imputing the missing value(s) without having redundancy, the following steps are used:
2. For a dataset D having a missing value with n class, first, we have to divide the. dataset D in n clusters, ci.
3. Then if there are any missing values in the dataset-
   i) mode is applied if the column is in categorical form
   ii) mean is applied if the column is in numerical form.

The proposed clustering-based imputing method can handle mix-type missing values. Specially, if there are both numeric and categorical type missing values then the proposed method computes mean of the feature to find the missing value whereas for categorical type, it computes mode. In my method, for a missing value attribute fi.

i) It checks the types of the attributes of the instance for which missing value is to completed.
ii) Now, the method considers k most similar instance from that the cluster of that Particular instance and compute the mean of the k similar instances of that attribute.
iii) Similarly, for categorical attribute also the missing value is computed from the k-similar instances using mode operation.

3.1. PROPOSED ALGORITHM:

To impute missing values, I have developed an algorithm using the concept of clustering. To develop my proposed method, I used the following algorithm for imputation with the missing values dataset. The required algorithm of my proposed method is given below –

ALGORITHM: Clustering-based Imputation Method

i) Input: A dataset D with missing values
ii) Output: A dataset D' without Missing Values

```
start:
   for n class: do
      divide the D into n clusters c_i
    end

    for each c_i ∈ C: do
                for each fi in the ci:
                   if ∃ missing value(s):
                          if fi=categorized:
                            missing value(s)=mode(fi)
                   end
                          if f_i = numerical
                               missing value = mean(fi)
                End
      end
stop.
```

3.2. CONCEPTUAL FRAMEWORK:

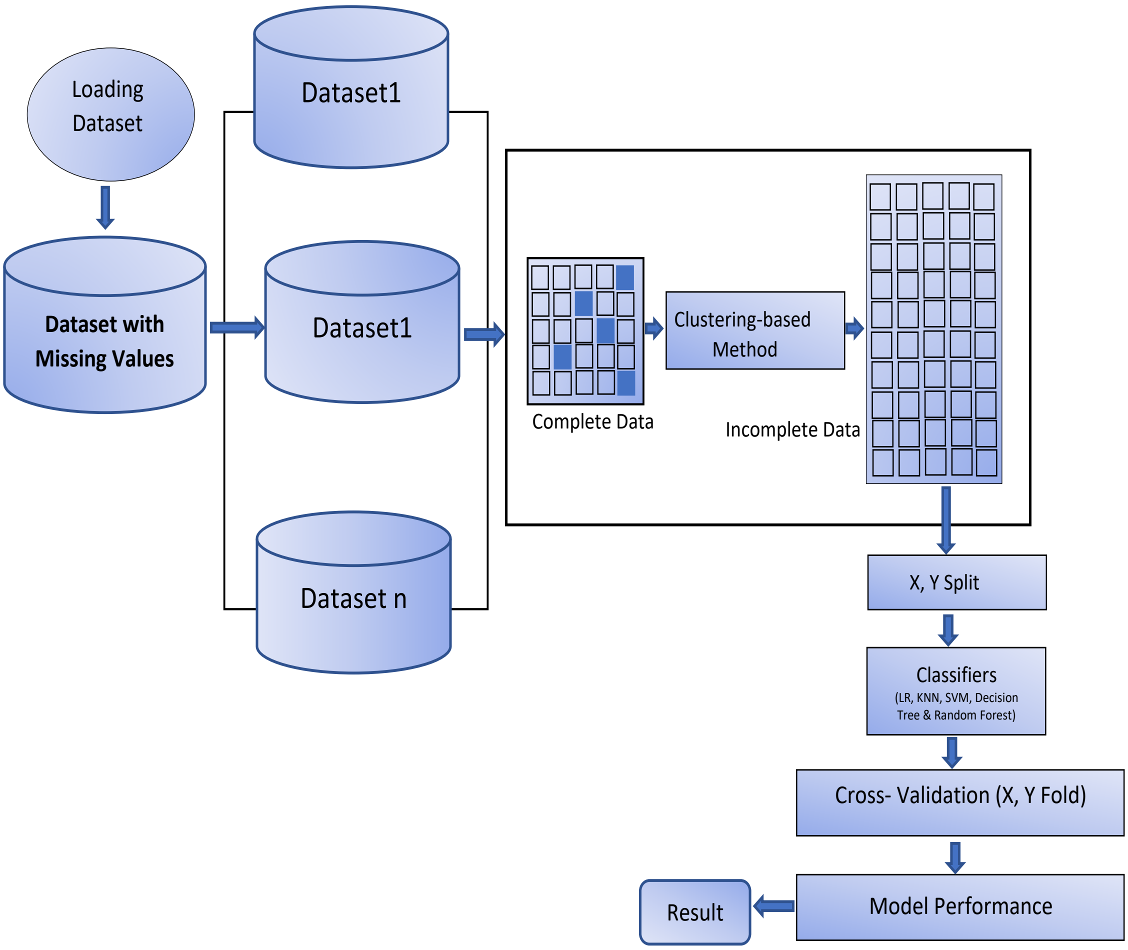


Figure 3.2(a): Conceptual Framework of my Proposed Method

4. DATA PREPROCESSING:

Data preprocessing is the process of transforming the raw data into an understandable format. It is also called data wrangling. There are some commonly used steps in data preprocessing as mentioned below –

i) Importing libraries
ii) Loading dataset
iii) Checking for missing values
iv) Checking for categorical or numerical data
v) Feature Scaling
vi) Splitting the data into training, validation, and evaluation sets

Explanation:

i) Importing libraries:
A library is a collection of modules that can be frequently called and used. Here, a lot of libraries are helpful in data preprocessing. Some of them are –

ii) Loading Dataset:
After importing the libraries, the next step is loading or importing the collected dataset. Here, the Pandas library is used to import the dataset. Mostly the datasets are available in CSV formats as they are low in size which makes them fast for processing.

iii) Checking for Missing Values:

Once the dataset is loaded, inspection is required to check whether the loaded dataset has the missing values or not. If there are, two methods of handling the missing values must apply i.e., at first, removing the entire row that contains the missing value, but there can be a possibility that we may end up losing some vital information. This can be a good approach if the size of the dataset is large. At last, if a numerical column has a missing value, then we can estimate the value by taking the mean, median, mode, etc.

iv) Checking for Categorical data:

Data in the dataset has to be in a numerical form to perform computation on it. Since Machine learning models contain complex mathematical computation, we can't feed them a non-numerical value. So, it is important to convert all the text values into numerical values. LabelEncoder() class of learned is used to covert these categorical values into numerical values.

v) Feature Scaling

Feature scaling is essential for machine learning algorithms that calculate distances between data. If not scaled the feature with a higher value range will start dominating when calculating distances, as explained intuitively in the introduction section. So algorithms that use distance calculations like K Nearest Neighbor, Regression, SVMs, etc are the ones that require feature scaling.

vi) Splitting data into training, validation, and evaluation sets

Finally, we need to split our data into three different sets, a training set to train the model, a validation set to validate the accuracy of our model, and finally test set to test the performance of our model on generic data. Before splitting the Dataset, it is important to shuffle the Dataset to avoid any biases. An ideal proportion to divide the Dataset is 60:20:20 i.e., 60% as the training set, and 20% as the test and validation set. To split the Dataset use train_test_split of sklearn. model_selection twice. Once to split the dataset into train and validation sets and then to split the remaining train dataset into train and test sets.

After doing all of these steps, the simple existing methods viz. mean, median, mode, ffill, bfill, zero and constant are imputed. And the proposed method is, then, fitted to the model using classifiers viz. Logistic Regression, K-Nearest Neighbor, Support Vector Machine, Decision Tree, and Random Forest.

5.RESULT ANALYSIS:

5.1. EXPERIMENTAL ENVIRONMENT

These experiments were carried out on a personal computer with 8GB main memory, **AMD Ryzen 5, AMD Radeon** processor, **4**GB NVIDIA **GEFORCE GTX** with 64 bits window-1**1 Single** home Operating System. The proposed method is implemented using Python of version 3.7 **in Jupyter Notebook (Anaconda).**

5.2. DATASET USED

In my experiment analysis, **8 (eight)** UCI respiratory and Kaggle datasets are used. Datasets with both numerical and categorical values with various dimensionalities and numbers of instances **having missing values** are used here. Description of the datasets are given in the table below:

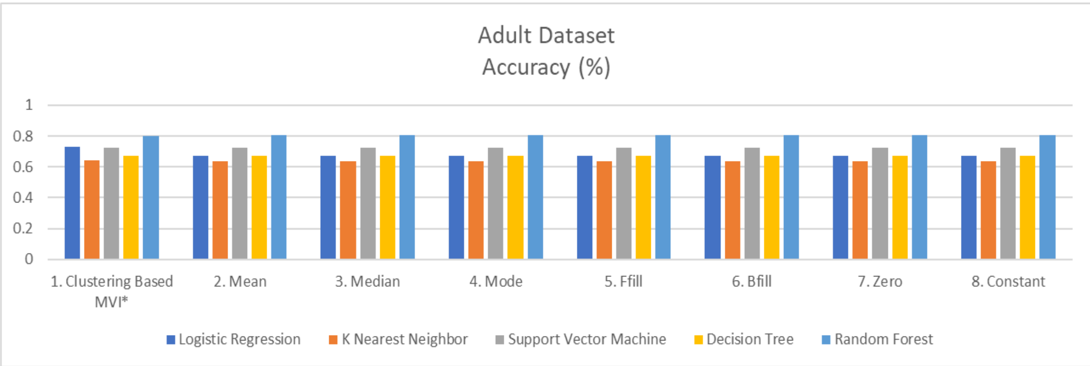
Dataset description.

Table: 3 Dataset Description

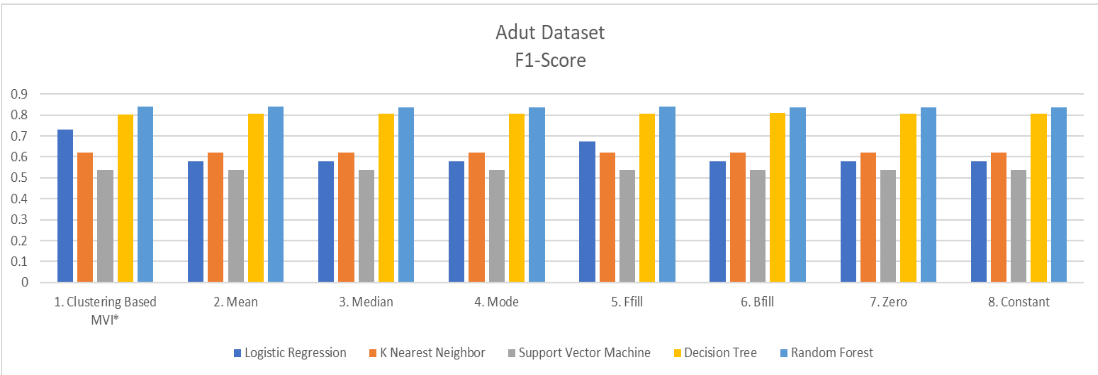
Sl.no.	**Dataset**	**No. of instances**	**No. of attributes**	**Data types**	**No. of Class Label**
1.	Adult	32562	15	Categorical	2
2.	Titanic	891	12	Categorical	2
3.	Diabetes	769	9	Categorical	2
4.	Melbourne	23545	22	Categorical	2
5.	Soybeans	683	35	categorical	4
6.	Lung_cancer	309	16	Categorical	2
7.	winequality-red	1599	12	Categorical	6
8.	winequality-white1	4898	12	Categorical	6

5.3. EMPIRICAL ANALYSIS:

The Performance of my method is analyzed and the graphs of the performance results are shown in the following figures:

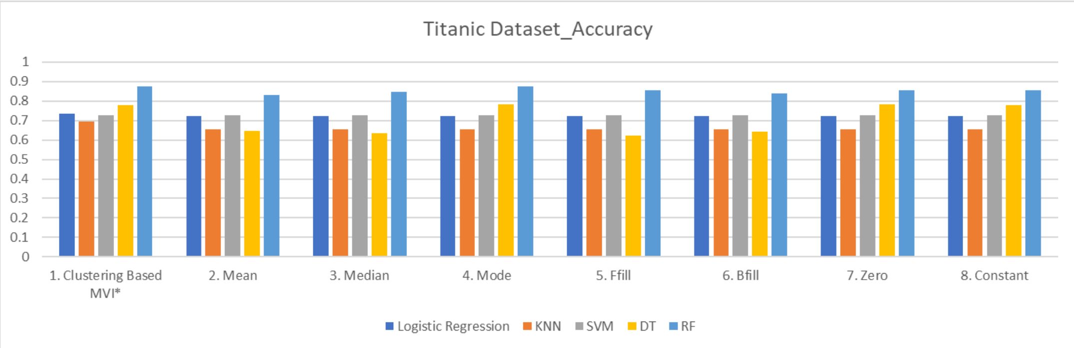


(a) Accuracy

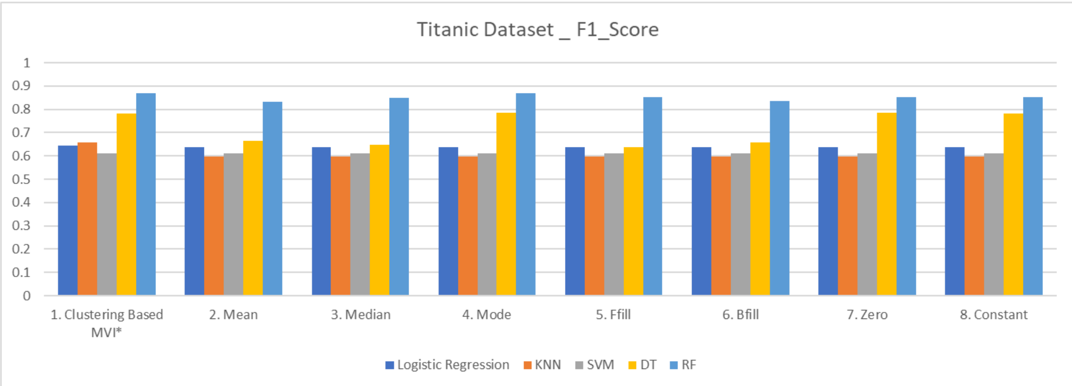


(b) F1-Score

Figure 1: Performance analysis on Adult dataset



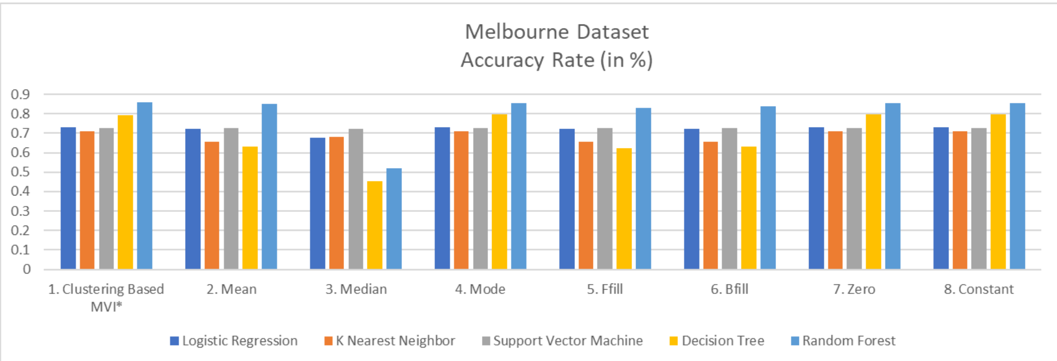
a) Accuracy

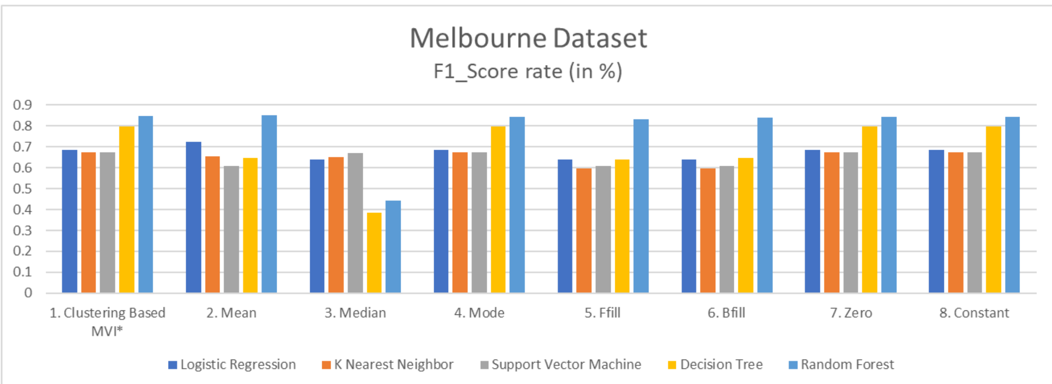


b) F1-Score

Figure 2: Performance analysis on Titanic dataset

a) Accuracy

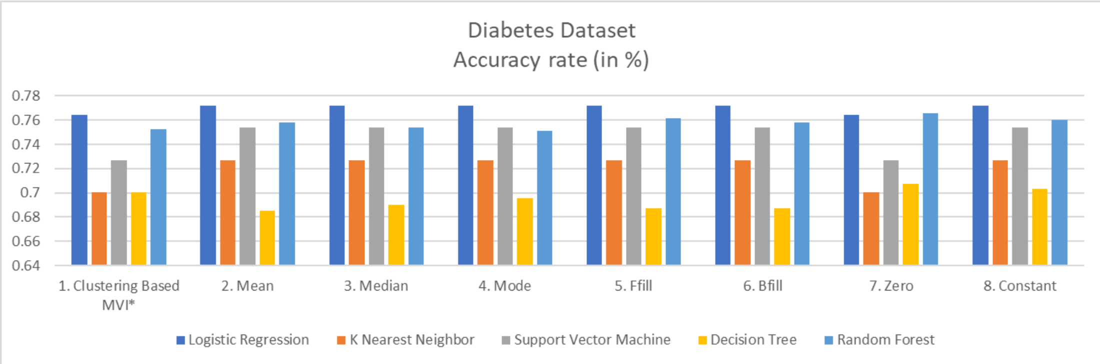


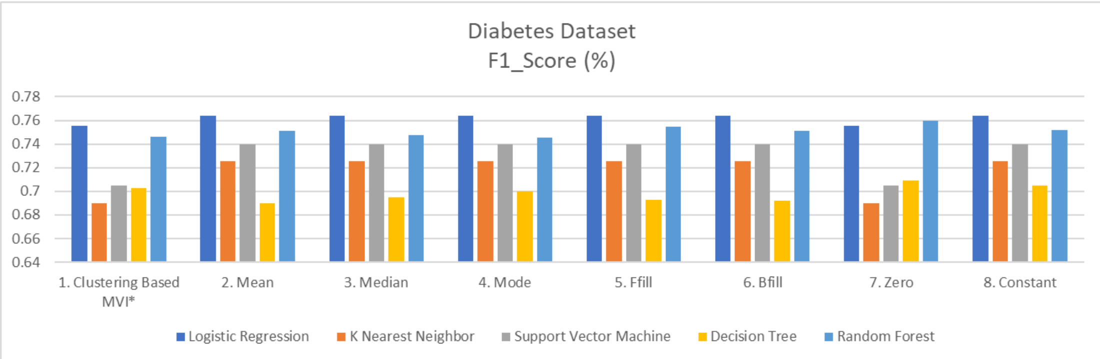


b) F1-Score

Figure 3: Performance analysis on Melbourne dataset

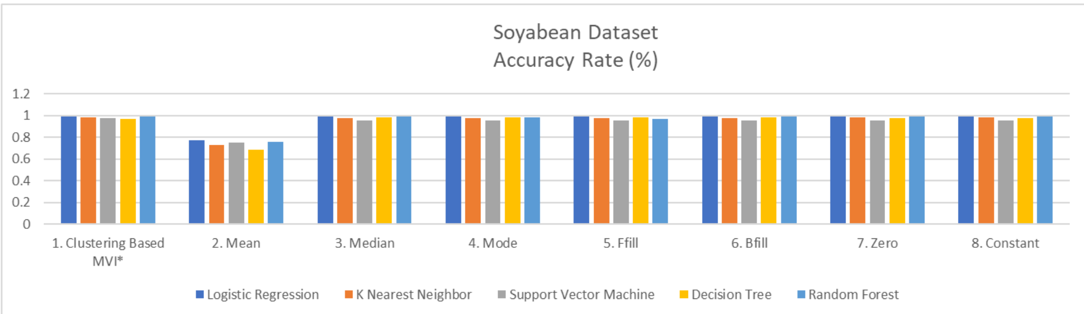
a) Accuracy



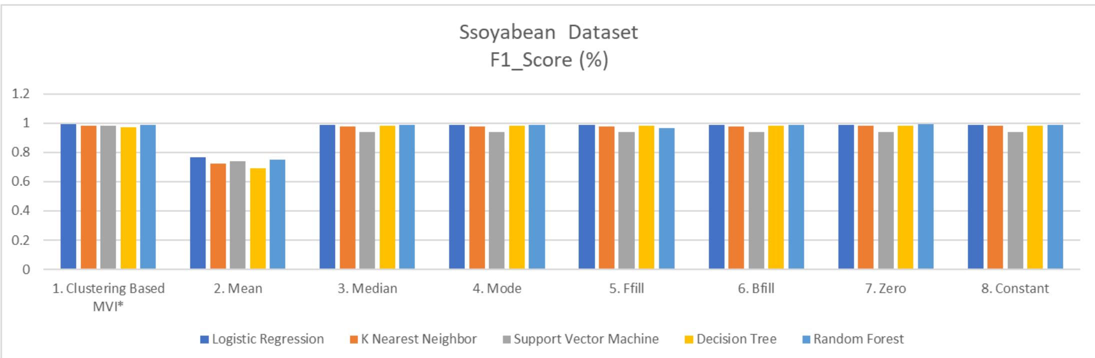


b) F1-Score

Figure 4: Performance analysis on Diabetes dataset



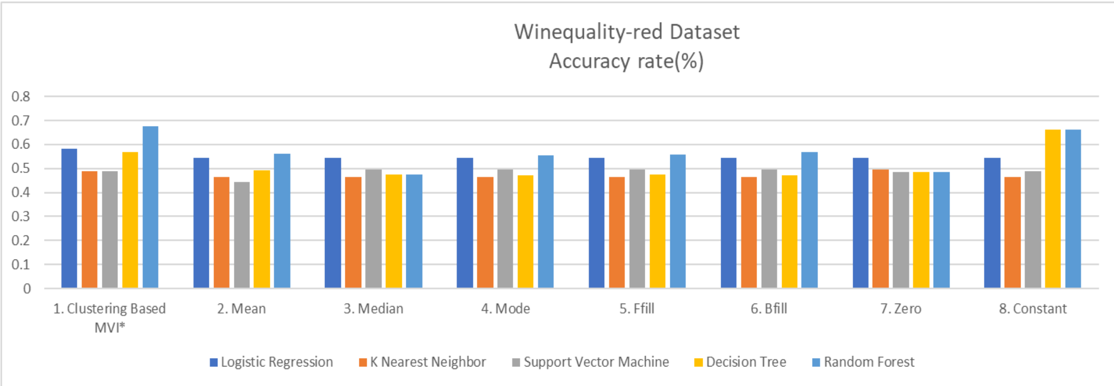
a) Accuracy

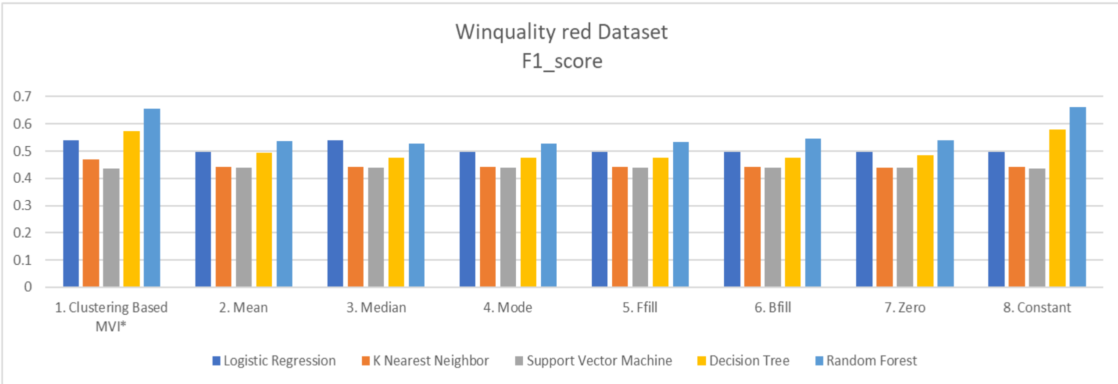


b) F1_Score

Figure 5: Performance analysis on Soyabean dataset

a) Accuracy

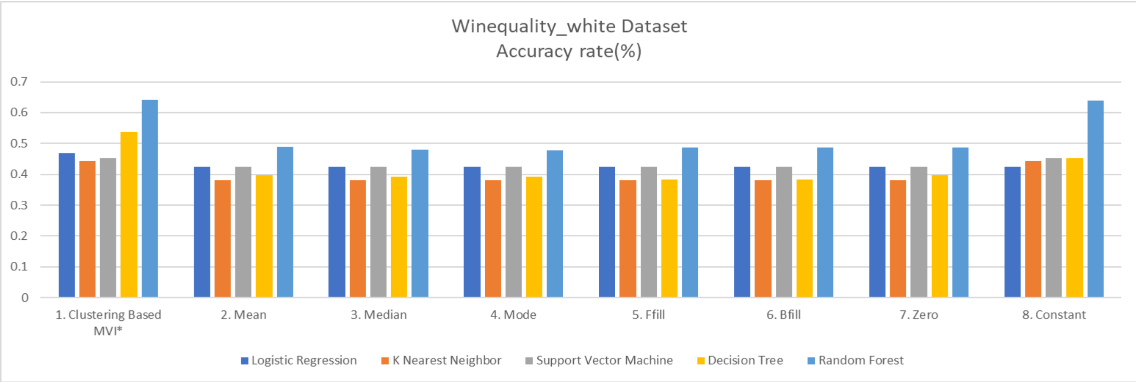


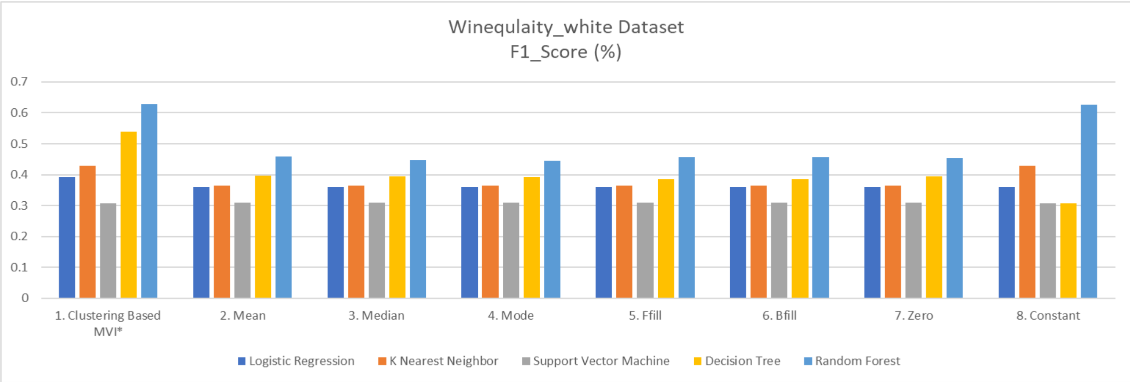


b) F1_Score

Figure 6: Performance analysis on Winequality_red dataset

a) Accuracy

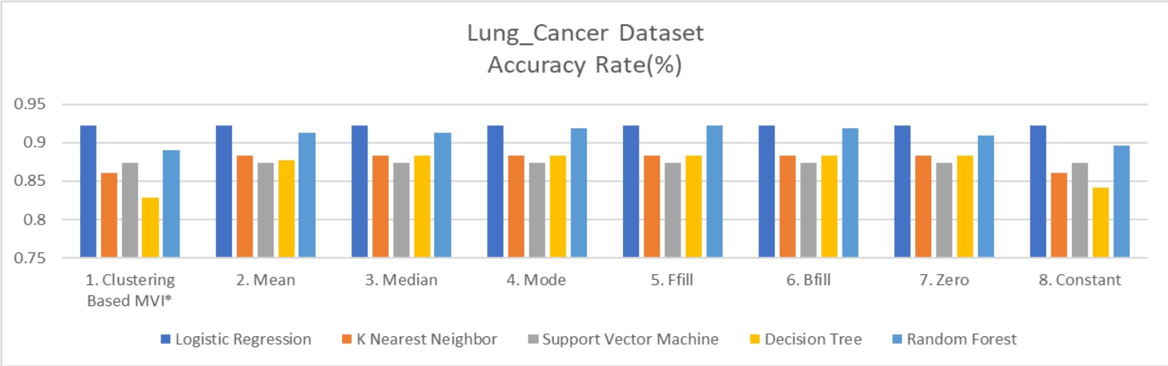


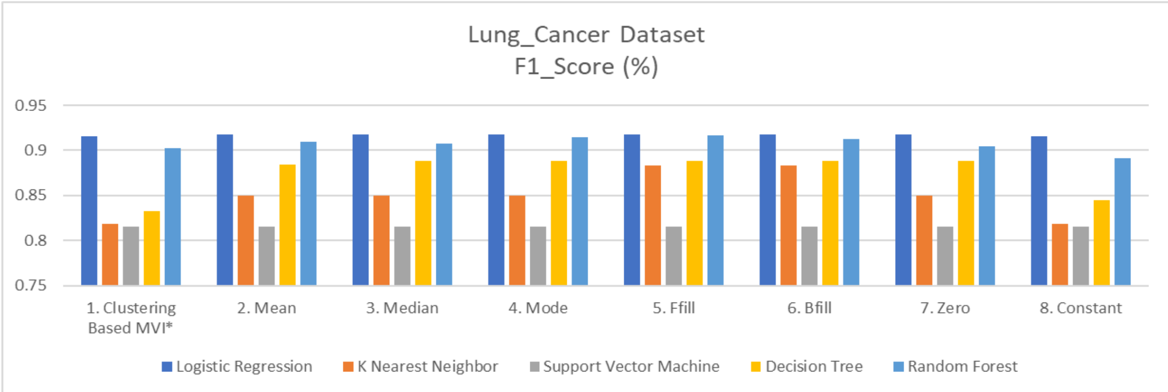


b) F1_Score

Figure 7: Performance analysis on Winequality_white dataset

a) Accuracy





b) F1_Score

Figure 8: Performance analysis on Lung_Cancer dataset

5.4. ANALYSIS OF THE RESULT:

The performance of the proposed method is tested on 8 datasets (Table: 3).

In the Adult dataset, as shown in Figures1: (a) and (b), the proposed method gives higher accuracy and f1_score than the other methods in Random Forest. It also gives almost the same accuracy as others in Logistic regression on testing my method to the adult dataset having several instances 32652 and several attributes 15 in Categorical data types with 2 class labels. But the proposed method gives less accuracy in K-nearest Neighbors than others. Not only on accuracy, my proposed method gives a higher F1_score in Random Forest and less score in Support Vector Machine.

In the Titanic dataset, as shown in Figures 2: (a) and (b), the proposed method gives higher accuracy and f1_score than the other methods in Random Forest. It also gives almost the same accuracy as others in Logistic regression on testing my method to the Titanic dataset having several instances 861 and number of attributes 12 in Categorical data types with 2 class labels. But the proposed method gives less accuracy in K-nearest neighbors than in others. Not only on accuracy, my proposed method gives a higher F1_Score in Random Forest and less score in Support Vector Machine.

In the Melbourne dataset, as shown in Figures 3: (a) and (b), the proposed method gives higher accuracy and f1_score in Random Forest than the other methods. It also gives almost the same accuracy as others in Logistic regression on testing my method to the Melbourne dataset having several instances 23545 and number of attributes 22 in Categorical data types with 2 class labels. But the proposed method gives less accuracy in the Decision Tree than others. Not only on accuracy, my proposed method gives a higher F1_Score in Random Forest and less score in Decision Tree.

In the Diabetes dataset, as shown in Figures 4: (a) and (b), the proposed method gives higher accuracy and f1_score in Logistic Regression than the other methods. It also gives almost the same accuracy as others in Random Forest on testing my method to the Diabetes dataset having several instances of 769 and number of attributes 9 in Categorical data types with 2 class labels. But the proposed method gives less accuracy in the Decision Tree than others. Not only on accuracy, but my proposed method also gives a higher F1_Score in Logistic Regression and less score in Decision Tree.

In the Soyabean dataset, as shown in Figures 5: (a) and (b), the proposed method gives higher accuracy and f1_score in Random Forest than the other methods. It also gives almost the same accuracy as others in Logistic Regression, K-nearest neighbors, and Support Vector Machine on testing my method to the Soyabean dataset having several instances 683 and number of attributes 35 in Categorical data types with 4 class labels. But the proposed method gives less accuracy in the Decision Tree than others. Not only on accuracy, my proposed method gives a higher F1_score in Random Forest but almost the same accuracy in Logistic Regression, K-Nearest Neighbor, and Support Vector Machine and less score in Decision Tree.

In the Winequality_red dataset, as shown in Figures 6: (a) and (b), the proposed method gives higher accuracy and f1_score in Random Forest than the other methods but the constant method gives the same higher accuracy in Decision Tree. It also gives almost the same accuracy as others in Logistic Regression and Decision Tree on testing my method to the Winequality_red dataset having several instances of 1599 and number of attributes 12 in Categorical data types with 6 class labels. But the proposed method gives less accuracy in Support Vector Machine than others. Not only on accuracy, my proposed method gives a higher F1_score in Random Forest but almost the same accuracy in Logistic Regression, K-Nearest Neighbor, and Support Vector Machine and less score in Support Vector Machine.

In the Winequality_white dataset, as shown in Figures 7: (a) and (b), the proposed method gives higher accuracy and f1_score in Random Forest than the other methods. It also gives almost not the same accuracy as others in Logistic Regression and Decision Tree on testing my method to the Winequality_red dataset having several instances of 1599 and number of attributes 12 in Categorical data types with 6 class labels. But the proposed method gives less accuracy in K Nearest Neighbor than others. Not only on accuracy, my proposed method gives a higher F1_Score in Random Forest but almost not the same accuracy in Logistic Regression, K-Nearest Neighbor, and Support Vector Machine and less score in Support Vector Machine.

In the Lung_Cancer dataset, as shown in Figures 8: (a) and (b), the proposed method gives higher accuracy and f1_score in Random Forest than the other methods. It also gives almost the same accuracy in Logistic Regression on testing my method to the Lung_Cancer dataset having several instances of 4898 and number of attributes 12 in Categorical data types with 6 class labels. But the proposed method gives less accuracy in the Decision Tree than others. Not only on accuracy, my proposed method gives a higher F1_Score in Logistic Regression but a lesser Score in Random Forest and less score in Support Vector Machine.

‘

6. DISCUSSION:

From the analysis of the above results, I can discuss here the proposed method gives higher accuracy and score in Random Forest and less in Decision Tree. However, in some cases of existing methods like the Constant method, the proposed method has the same accuracy rate and F1_score. In the same way, each dataset has a particular accuracy and scores high or low based on its number of instances, attributes, data types, and class labels. The experimental result also shows that the proposed method gives better and comparable performance to the other method on various datasets. The drawback of the proposed method is that, even though my method shows better accuracy and scores, some other existing proposed methods of different research scholars, give less accuracy. However, in the preprocessing of data when I tried to impute the missing values in my work, my proposed method gives higher accuracy than the existing methods like mean, median, mode, ffill, bfill, zero, and constant.

7. CONCLUSION AND FUTURE WORK:

This project introduced a Clustering-based imputation method for handling the missing values in the dataset. The method is developed to find which method is the best one using the classifiers like Logistic Regression, K Nearest Neighbors (KNN), Support Vector Machine (SVM), Decision Tree and Random Forest, etc. for handling the missing values in the dataset.

The method has been evaluated using publicly available 8 datasets from the UCI repository and Kaggle. Developing an improved imputation method based on handling the missing values to support various application domains is in progress. More experiments with other search strategies and parallel computation will be employed to validate the proposed method to estimate higher dimensions cases.

References:

1. Chhabra, G., V. Vashisht, and J. Ranjan. "A review on missing data value estimation using imputation algorithm." *Journal of Advanced Research in Dynamical and Control Systems* 11.7 (2019): 312-318.
2. Zhang, Chengqi, et al. "Clustering-based missing value imputation for data preprocessing." *2006 4th IEEE International Conference on Industrial Informatics*. IEEE, 2006.
3. Singh, Archana, and Avantika Yadav. "Hybrid Approach of Hierarchical Clustering." *World Applied Sciences Journal* 32.7 (2014): 1181-1191.
4. Radhakrishnan, Sridevi, and D. Shanmuga Priyaa. "An Ensemble approach on Missing Value Handling in Hepatitis Disease Dataset." *International Journal of Computer Applications* 130.17 (2015).
5. Pampaka, Maria, Graeme Hutcheson, and Julian Williams. "Handling missing data: analysis of a challenging data set using multiple imputations." *International Journal of Research & Method in Education* 39.1 (2016): 19-37.
6. Aydilek, Ibrahim Berkan, and Ahmet Arslan. "A hybrid method for imputation of missing values using optimized fuzzy c-means with support vector regression and a genetic algorithm." *Information Sciences* 233 (2013): 25-35.
7. Bishop, Ellen E., and Yichuan Zhao. "Rank Regression Inference Via Empirical Likelihood." (2005).
8. Stone, Mervyn. "Cross-validatory choice and assessment of statistical predictions." *Journal of the royal statistical society: Series B (Methodological)* 36.2 (1974): 111-133.
9. Gessert, G. H. "Handling missing data by using stored truth values." *ACM SIGMOD Record* 20.3 (1991): 30-42.
10. Blake, Catherine L., and Christopher J. Merz. "UCI repository of machine learning databases, 1998." (1998): 75-85.

Appendex I

Program Code:

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
from result_function import *
```

Loading Dataset

In [2]:
```
test=pd.read_csv('titanic.csv')
```

In [3]:
```
test
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. Willia	male	35.0	0	0	373450	8.0500	NaN	S

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
				m Henry								
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [4]:
```
cols = test.columns
for col in cols:
    test[col] = np.where(test[col] == 0, np.nan, test[col])
test
```

Out[4]:

In [5]:
```
classlabels=test['Sex'].unique()
```

In [6]:
```
for i in range(test.shape[1]):
    if test.iloc[:,i].isna().sum()!=0:
        print(i)
#test.isna().sum()
```

```
1
5
```

```
6
7
9
10
11
```

In [7]:

```
L=[]
```

In [8]:

```
new_df=pd.DataFrame(L, columns=test.columns)
```

In [9]:

```
new_df
```

Out[9]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

In [10]:

```
def missing_value_handling(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mean())

    return data
```

In [11]:

```
print(classlabels)
#new_df=pd.DataFrame(L, columns=df.columns)
#arr=np.array(lt)
li=[]
for i in classlabels:
    ax=test.loc[test['Sex']==i]
    hndle_ax=missing_value_handling(ax)
    hndle_arr=np.array(hndle_ax)
    for k in hndle_arr:
        li.append(k)
    #new_df.append(hndle_ax)
    #print(hndle_arr)
```

```
['male' 'female']
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1745:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```

In [12]:

```
from sklearn.utils import shuffle
#shuffled = shuffle(df)
#print(shuffled.head())
```

In [13]:

```
shuffle(pd.DataFrame(li,columns=test.columns), random_state=5)
```

Out[13]:
891 rows × 12 columns

In [14]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
import pandas as pd
import numpy as np


def missing_value_handling(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mean())

    return data


def missing_value_handling_mode(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mean())

    return data

def missing_value_handling_mean(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
```

```
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mean())

    return data


def missing_value_handling_median(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].median())

    return data


def missing_value_handling_ffill(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].ffill())

    return data


def missing_value_handling_bfill(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].bfill())

    return data

def missing_value_handling_zero(data):
    for i in range(data.shape[1]):
```

```
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].zero())

    return data


def missing_value_handling_constant(data):
    for i in range(data.shape[1]):
        #print(dataset.iloc[:,i].isnull().sum())
        if data.iloc[:,i].isnull().sum()>0:


            if data.iloc[:,i].dtype=='object':

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].mode()[0])
            else:

data.iloc[:,i]=data.iloc[:,i].fillna(data.iloc[:,i].zero())

    return data


def cluster_MH(df, class_label,rstate=5):

    classlabels=df[class_label].unique()

    li=[]
    for i in classlabels:
        ax=df.loc[df[class_label]==i]
        hndle_ax=missing_value_handling(ax)
        hndle_arr=np.array(hndle_ax)
        for k in hndle_arr:
            li.append(k)
    data=shuffle(pd.DataFrame(li,columns=df.columns),
random_state=rstate)

    return data

def Label_encoded_(data):
    label_encoder= LabelEncoder()
    for i in range(data.shape[1]):
        if data.iloc[:,i].dtypes=="object":
            #print(data.iloc[:,i].name)
            data.iloc[:,i]=label_encoder.fit_transform(data.iloc[:,i])
    return data

    #NORMALISATION
```

```
def normalize_scale(X):
    mun=X.values

    column_name=X.columns
    scale = MinMaxScaler(feature_range=(0,1))
    data_scaled = scale.fit_transform(mun)
    Scale_X=pd.DataFrame(data_scaled,columns=column_name)
    return Scale_X
```

Clustering Based Imputator

In [15]:

```
hnddata=cluster_MH(test,'Sex')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1745:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```

In [16]:

```
new_df01=Label_encoded_(hnddata)
```

In [17]:

```
from result_function import *
```

In [18]:

```
new_df01
```

Out[18]:

891 rows × 12 columns

In [19]:

```
X=new_df01.drop("Sex",axis=1)
y=new_df01['Sex']
```

In [20]:

```
result(X,y,cv=2)
```

```
Logistic Regression:
Accuracy: 0.7070590013604071    Precision:  0.69814523969044
Recall: 0.7070590013604071      F1 Score:  0.6766747640715571
MCC: 0.30675907094608357 time: 0.023675203323364258

--------------------

KNN:
Accuracy: 0.6004358341311029    Precision:  0.5606159261360912
Recall: 0.6004358341311029      F1 Score:  0.5677680913596288
MCC: 0.034583340473699445 time: 0.02011120319366455

--------------------

Support Vector Machine :
Accuracy: 0.647586537008112     Precision:  0.41936847935479526
Recall: 0.647586537008112       F1 Score:  0.5090699288017946
MCC: 0.0 time: 0.015980243682861328

--------------------
```

```
Decision Tree:
Accuracy: 0.958469793923515     Precision:  0.9592392309541933
Recall: 0.958469793923515       F1 Score:  0.9585753503734032
MCC: 0.9101176699324899 time: 0.006663799285888672

--------------------

Random Forest:
Accuracy: 0.9786819166624678    Precision:  0.9789417141216258
Recall: 0.9786819166624678      F1 Score:  0.9786471925296633
MCC: 0.9535047784894336 time: 0.1367737054824829

--------------------
```

Mean

In [21]:

```
hndata2=missing_value_handling_mean(test)
hndata2
```

Out[21]:

891 rows × 12 columns

In [22]:

```
new_df02=Label_encoded_(hndata2)
```

In [23]:

```
from result_function import *
```

In [24]:

```
new_df02
```

Out[24]:

891 rows × 12 columns

In [25]:

```
X=new_df02.drop("Sex",axis=1)
y=new_df02['Sex']
```

In [26]:

```
result(X,y,cv=2)
```

```
Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.026044607162475586

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.017635226249694824

--------------------

Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
```

```
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.01391756534576416

--------------------

Decision Tree:
Accuracy: 0.5164206177256008    Precision:  0.4964354836900938
Recall: 0.5164206177256008      F1 Score:  0.39812619722826137
MCC: 0.01436468565468709 time: 0.008367419242858887

--------------------

Random Forest:
Accuracy: 0.5029702221998287    Precision:  0.7725327187670363
Recall: 0.5029702221998287      F1 Score:  0.35327707900633487
MCC: 0.056872454435335215 time: 0.15864837169647217

--------------------
```

MEDIAN

In [27]:

```
hndata3=missing_value_handling_median(test)
hndata3
```

Out[27]:

891 rows × 12 columns

In [28]:

```
new_df03=Label_encoded_(hndata3)
```

In [29]:

```
from result_function import *
```

In [30]:

```
new_df03
```

Out[30]:

891 rows × 12 columns

In [31]:

```
X=new_df03.drop("Sex",axis=1)
y=new_df03['Sex']
```

In [32]:

```
result(X,y,cv=2)
```

```
Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.02793097496032715

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.017713069915771484
```

```
--------------------

Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.01535177230834961

--------------------

Decision Tree:
Accuracy: 0.5175416939587847    Precision:  0.49908217041749803
Recall: 0.5175416939587847      F1 Score:  0.4000943678473273
MCC: 0.01709331202951303 time: 0.008404254913330078

--------------------

Random Forest:
Accuracy: 0.50745200786013      Precision:  0.5963653937469741
Recall: 0.50745200786013        F1 Score:  0.36233860235415516
MCC: 0.046535851942309636 time: 0.17052674293518066

--------------------
```

MODE

In [33]:
```
hndata4=missing_value_handling_mode(test)
hndata4
```

Out[33]:
891 rows × 12 columns

In [34]:
```
new_df04=Label_encoded_(hndata4)
```

In [35]:
```
new_df04
```

Out[35]:
891 rows × 12 columns

In [36]:
```
X=new_df04.drop("Sex",axis=1)
y=new_df04['Sex']
```

In [37]:
```
result(X,y,cv=2)
```

```
Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.023507356643676758

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.017061710357666016

--------------------
```

```
Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.014825105667114258

--------------------

Decision Tree:
Accuracy: 0.5141784652592332    Precision:  0.4876955668784281
Recall: 0.5141784652592332      F1 Score:  0.39552649688005104
MCC: 0.006055152837578864 time: 0.0076487064361572266

--------------------

Random Forest:
Accuracy: 0.5052098553937623    Precision:  0.5960799231807857
Recall: 0.5052098553937623      F1 Score:  0.35772467303072475
MCC: 0.03924068830846471 time: 0.14934074878692627

--------------------
```

Ffill

```
hndata5=missing_value_handling_mode(test) hndata5
In [38]:
new_df05=Label_encoded_(hndata5)
new_df05
In [39]:
X=new_df05.drop("Sex",axis=1)
y=new_df05['Sex']
result(X,y,cv=2)
Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.028142929077148438

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.016590595245361328

--------------------

Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.012848973274230957
```

```
--------------------

Decision Tree:
Accuracy: 0.5175416939587847    Precision:  0.5026158579071268
Recall: 0.5175416939587847      F1 Score:  0.3987521781741791
MCC: 0.01998274940742767 time: 0.007925868034362793

--------------------

Random Forest:
Accuracy: 0.5029677029273946    Precision:  0.5957970302269331
Recall: 0.5029677029273946      F1 Score:  0.35305385513232423
MCC: 0.03032701551261585 time: 0.16259634494781494

--------------------
```

Bfill

In [40]:
```
hndata6=missing_value_handling_mode(test)
hndata6
```
Out[40]:
891 rows × 12 columns

In [41]:
```
new_df06=Label_encoded_(hndata6)
```
In [42]:
```
new_df06
```
Out[42]:
891 rows × 12 columns

In [43]:
```
X=new_df06.drop("Sex",axis=1)
y=new_df06['Sex']
```
In [44]:
```
 result(X,y,cv=2)
```

```
Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.023370862007141113

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.01845693588256836
```

```
--------------------

Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.015606164932250977

--------------------

Decision Tree:
Accuracy: 0.5175416939587847    Precision:  0.49908217041749803
Recall: 0.5175416939587847      F1 Score:  0.4000943678473273
MCC: 0.01709331202951303 time: 0.007887005805969238

--------------------

Random Forest:
Accuracy: 0.5029651836549605    Precision:  0.6666734862504411
Recall: 0.5029651836549605      F1 Score:  0.35883158641359647
MCC: 0.04032204304453557 time: 0.1586998701095581

--------------------
```

Zero

In [45]:

```
hndata7=missing_value_handling_mode(test)
hndata7
```

Out[45]:

891 rows × 12 columns

In [46]:

```
new_df07=Label_encoded_(hndata7)
```

In [47]:

```
new_df07
```

Out[47]:

891 rows × 12 columns

In [48]:

```
X=new_df07.drop("Sex",axis=1)
y=new_df07['Sex']
```

In [49]:

```
 result(X,y,cv=2)

Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.025658130645751953

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.018438339233398438

--------------------
```

```
Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.014987587928771973

--------------------

Decision Tree:
Accuracy: 0.5164206177256008    Precision:  0.4932712369077786
Recall: 0.5164206177256008      F1 Score:  0.39946220800109355
MCC: 0.011639991815460741 time: 0.007344841957092285

--------------------

Random Forest:
Accuracy: 0.5052123746661964    Precision:  0.7142304941246995
Recall: 0.5052123746661964      F1 Score:  0.3599650086789514
MCC: 0.062114374868324926 time: 0.15953242778778076

--------------------
```

Constant

In [50]:
```
hndata8=missing_value_handling_mode(test)
hndata8
```

Out[50]:
891 rows × 12 columns

In [51]:
```
new_df08=Label_encoded_(hndata8)
```

In [52]:
```
new_df08
```

Out[52]:
891 rows × 12 columns

In [53]:
```
X=new_df08.drop("Sex",axis=1)
y=new_df08['Sex']
```

In [54]:
```
 result(X,y,cv=2)
```

```
Logistic Regression:
Accuracy: 0.6274046455383685    Precision:  0.6133515098505536
Recall: 0.6274046455383685      F1 Score:  0.574184216284581
MCC: 0.12523651113768816 time: 0.023711681365966797

--------------------

KNN:
Accuracy: 0.5388396231168439    Precision:  0.5736736572114887
Recall: 0.5388396231168439      F1 Score:  0.46149117745026447
MCC: 0.03360472722455682 time: 0.017525672912597656
```

```
--------------------

Support Vector Machine :
Accuracy: 0.6464629415024941    Precision:  0.41911134556401675
Recall: 0.6464629415024941      F1 Score:  0.5085331721538399
MCC: -0.017519919308392164 time: 0.014530539512634277

--------------------

Decision Tree:
Accuracy: 0.515299541492417     Precision:  0.49054887500562677
Recall: 0.515299541492417       F1 Score:  0.3974995636195928
MCC: 0.008878207325673794 time: 0.00886237621307373

--------------------

Random Forest:
Accuracy: 0.5074545271325641    Precision:  0.7730985133631
Recall: 0.5074545271325641      F1 Score:  0.362619024882469
MCC: 0.07517329105044226 time: 0.15965425968170166

--------------------

Naive Bayes:
Accuracy: 0.6151131153322921    Precision:  0.6622117769600927
Recall: 0.6151131153322921      F1 Score:  0.5863237004008561
MCC: 0.2099747801180839 time: 0.0061779022216796875

--------------------

Stochastic Gradient Descent:
Accuracy: 0.6262357031289363    Precision:  0.5971217070980499
Recall: 0.6262357031289363      F1 Score:  0.566639337646448
MCC: 0.0802468404006349 time: 0.006747245788574219

--------------------
```

In [ ]:

Appendex II

Explanation of Existing simple Imputation Methods:

Impute / Replace Missing Values with Mean

Mean() function is used to fill the missing values in the dataset by replacing with the mean value of the entire feature column. Imputing missing data with mean values can only be done with **numerical data**.

Syntax:

1dataset.fillna(dataset.mean())

Impute / Replace Missing Values with Median

Median () function is used to fill the missing values in the dataset by replacing with the **median value** of the entire feature column. When the data is skewed, it is good to consider using the median value for replacing the missing values. And imputing missing data with median value can only be done with **numerical data**.

Syntax:

dataset.fillna(dataset.median())

Impute / Replace Missing Values with Mode

Mode () function is used to fill the missing values in the dataset by replacing with the mode value or most **frequent** value of the entire feature column. When the data is skewed, it is good to consider using mode values for replacing the missing values. And imputing missing data with **mode** values can be done with numerical and categorical data.

Syntax:

dataset.fillna(dataset.mode())

Imputer/Replace Missing Values with Ffill

ffill() function is used to fill the missing value in the dataset. ‘ffill’ stands for ‘forward fill’ and will propagate last valid observation forward.

Syntax: *Dataset.ffill(axis=None, inplace=False, limit=None, downcast=None)*

Imputer/Replace Missing Values with Bfill :

The bfill() method replaces the NULL values with the values from the next row (or next *column*, if the axis parameter is set to 'columns').

Syntax

dataset.bfill(axis, inplace, limit, downcast)

Imputer/Replace Missing Values with Zero:

Zero() function is used to fill the missing value with zero(s) in the dataset.

Syntax:

(1) For a single column using *Pandas:*

```
df['DataFrame Column'] = df['DataFrame Column'].fillna(0)
```

(2) For a single column using *NumPy:*

```
df['DataFrame Column'] = df['DataFrame Column'].replace(np.nan, 0)
```

(3) For an entire DataFrame using *Pandas:*

```
df.fillna(0)
```

(4) For an entire DataFrame using *NumPy:*

```
df.replace(np.nan,0)
```

Imputer/Replace Missing Values with Constant:

The constant () function is used to replace the missing values with either zero or any constant value.

Syntax or Usage:

```
na.constant(.x, .na)

na.inf(.x)

na.neginf(.x)

na.true(.x)

na.false(.x)

na.zero(.x)
```