

# A MACHINE LEARNING MODEL TO ANALYZE AND FORECAST BANKRUPTCY

## Team:

S.No	Name	E-mail	Role
1.	Surya Kiran Adari	SuryaKiranAdari@my.unt.edu	Designing & Researching
2.	Parithosh Girish	ParithoshGirish@my.unt.edu	Researching & Editing
3.	Kavya Gundla	KavyaGundla@my.unt.edu	Coding & Presentation
4.	Anirudh Muppidi	Anirudhmuppidi@my.unt.edu	Literature survey & Coding
5.	Sai Priya Nallamala	SaipriyaNallamala@my.unt.edu	Coding/testing & presentation

## **Workflow**

We worked collaboratively on the project by dividing the tasks like data preprocessing, visualization and predicting the results. Later, discussing the progress and updates of the project tasks. This collaborative effort has involved sharing research findings and working together on tasks to develop and improve the techniques and methodologies that improves the performance. We researched data imputation techniques and implementation of it. Those are added in the references section of this document.

## **Abstract**

The term Bankruptcy is defined as a legal proceeding in which any person or organization is incapable of repaying the loans. Bankruptcy is one of the key problems for both associations and banks. Throughout the world, academic literature and professional researchers have discussed the chance of business debts. Foretelling at the starting stage of bankruptcy may help the banks reduce their financial losses and help them to choose correct opinions. The aim of bankruptcy prediction is to evaluate a company's financial status and its potential prospects for sustained operation in the market over the long term. This field encompasses a broad range of topics in finance and econometrics, incorporating both expert insights into the phenomenon and data on successful and unsuccessful companies.

Generally, companies are evaluated using a variety of indicators that describe their business situation, which are then utilized to generate a mathematical model based on past observations. We took a bankruptcy data set from Polish companies, where synthetic characteristics were utilized to depict advanced statistics. This study focuses on the analysis of bankruptcy by using different Machine learning algorithms. This model helps us to predict whether any person or association will go bankrupt or not. Bankruptcy is one of the crucial topics that needs to be addressed in order to reduce risk of money for both organizations and banks.

## **Data Specification**

We utilized the Polish bankruptcy data from the University of California Irvine (UCI) Machine Learning Repository, which is a comprehensive collection of datasets accessible for research and learning purposes for the Machine Learning/Data Science community. This specific dataset focuses on predicting bankruptcy for Polish companies, and the information was gathered from the Emerging Markets Information Service (EMIS), a database containing data on emerging markets across the globe. The dataset consists of bankrupt companies analyzed from 2000 to 2012, and still-operating companies evaluated from 2007 to 2013. Due to the dataset's rich set of econometric indicators as attributes (features) and its large sample size of Polish companies analyzed over five distinct timeframes, it is particularly well-suited for our research on bankruptcy

prediction. Based on the data collected, we identified five classification cases that are contingent on the forecasting period.

**First year:** The information includes financial rates for the first year of the forecasting period and a class label that, after five years, indicates whether or not a company will file for bankruptcy.

**Second year :** Financial rates from the second year of the forecasting period are included in the data, and a matching class label indicates whether bankruptcy will occur after four years.

**Third year:** The information includes financial rates for the third year of the forecasting period and a class label that designates whether bankruptcy will occur after three years.

**Fourth year:** The information includes financial rates for the fourth year of the projection period and a class label that designates whether bankruptcy will occur after two years.

**Fifth year:** The information includes financial rates for the fifth year of the forecasting period and a class label that designates whether bankruptcy will occur after one year.

**Table 1:** Summary of dataset

<b>Data</b>	<b>Total Instances</b>	<b>Bankrupt Instances</b>	<b>Non-Bankrupt Instances</b>
1st Year	7027	271	6756
2nd Year	10173	400	9773
3rd Year	10503	495	10008
4th Year	9792	515	9227
5th Year	5910	410	5500

**Table 2 :** Summary of features in bankruptcy data

<b>ID</b>	<b>Description</b>	<b>ID</b>	<b>Description</b>
<b>X1</b>	net profit / total assets	<b>X33</b>	operating expenses / short-term liabilities
<b>X2</b>	total liabilities / total assets	<b>X34</b>	operating expenses / total liabilities
<b>X3</b>	working capital / total assets	<b>X35</b>	profit on sales / total assets
<b>X4</b>	current assets / short-term liabilities	<b>X36</b>	total sales / total assets

<b>X5</b>	$[(\text{cash} + \text{short-term securities} + \text{receivables} - \text{short-term liabilities}) / (\text{operating expenses} - \text{depreciation})] * 365$	<b>X37</b>	$(\text{current assets} - \text{inventories}) / \text{long-term liabilities}$
<b>X6</b>	$\text{retained earnings} / \text{total assets}$	<b>X38</b>	$\text{constant capital} / \text{total assets}$
<b>X7</b>	$\text{EBIT} / \text{total assets}$	<b>X39</b>	$\text{profit on sales} / \text{sales}$
<b>X8</b>	$\text{book value of equity} / \text{total liabilities}$	<b>X40</b>	$(\text{current assets} - \text{inventory} - \text{receivables}) / \text{short-term liabilities}$
<b>X9</b>	$\text{sales} / \text{total assets}$	<b>X41</b>	$\text{total liabilities} / ((\text{profit on operating activities} + \text{depreciation}) * (12/365))$
<b>X10</b>	$\text{equity} / \text{total assets}$	<b>X42</b>	$\text{profit on operating activities} / \text{sales}$
<b>X11</b>	$(\text{gross profit} + \text{extraordinary items} + \text{financial expenses}) / \text{total assets}$	<b>X43</b>	$\text{rotation receivables} + \text{inventory turnover in days}$
<b>X12</b>	$\text{gross profit} / \text{short-term liabilities}$	<b>X44</b>	$(\text{receivables} * 365) / \text{sales}$
<b>X13</b>	$(\text{gross profit} + \text{depreciation}) / \text{sales}$	<b>X45</b>	$\text{net profit} / \text{inventory}$
<b>X14</b>	$(\text{gross profit} + \text{interest}) / \text{total assets}$	<b>X46</b>	$(\text{current assets} - \text{inventory}) / \text{short-term Liabilities}$
<b>X15</b>	$(\text{total liabilities} * 365) / (\text{gross profit} + \text{depreciation})$	<b>X47</b>	$(\text{inventory} * 365) / \text{cost of products sold}$
<b>X16</b>	$(\text{gross profit} + \text{depreciation}) / \text{total liabilities}$	<b>X48</b>	$\text{EBITDA} (\text{profit on operating activities} - \text{depreciation}) / \text{total assets}$
<b>X17</b>	$\text{total assets} / \text{total liabilities}$	<b>X49</b>	$\text{EBITDA} (\text{profit on operating activities} - \text{depreciation}) / \text{sales}$
<b>X18</b>	$\text{gross profit} / \text{total assets}$	<b>X50</b>	$\text{current assets} / \text{total liabilities}$
<b>X19</b>	$\text{gross profit} / \text{sales}$	<b>X51</b>	$\text{short-term liabilities} / \text{total assets}$
<b>X20</b>	$(\text{inventory} * 365) / \text{sales}$	<b>X52</b>	$(\text{short-term liabilities} * 365) / \text{cost of products sold}$
<b>X21</b>	$\text{sales (n)} / \text{sales (n-1)}$	<b>X53</b>	$\text{equity} / \text{fixed assets}$
<b>X22</b>	$\text{profit on operating activities} / \text{total assets}$	<b>X54</b>	$\text{constant capital} / \text{fixed assets}$
<b>X23</b>	$\text{net profit} / \text{sales}$	<b>X55</b>	$\text{working capital}$
<b>X24</b>	$\text{gross profit (in 3 years)} / \text{total assets}$	<b>X56</b>	$(\text{sales} - \text{cost of products sold}) / \text{sales}$
<b>X25</b>	$(\text{equity} - \text{share capital}) / \text{total assets}$	<b>X57</b>	$(\text{current assets} - \text{inventory} - \text{short-term liabilities}) / (\text{sales} - \text{gross profit} - \text{depreciation})$

<b>X26</b>	(net profit + depreciation) / total liabilities	<b>X58</b>	total costs /total sales
<b>X27</b>	profit on operating activities / financial expenses	<b>X59</b>	long-term liabilities / equity
<b>X28</b>	working capital / fixed assets	<b>X60</b>	sales / inventory
<b>X29</b>	logarithm of total assets	<b>X61</b>	sales / receivables
<b>X30</b>	(total liabilities - cash) / sales	<b>X62</b>	(short-term liabilities *365) / sales
<b>X31</b>	(gross profit + interest) / sales	<b>X63</b>	sales / short-term liabilities
<b>X32</b>	(current liabilities * 365) / cost of products sold	<b>X64</b>	sales / fixed assets

As shown in the table, there are 64 features(dtype = 'float64') from X1 to X64 and a class variable y(dtype = 'int64'). Y is a predictor column which has 0 and 1 values that decides whether the company will go bankrupt or not and this can be done using supervised machine learning models.

## Project Design

The project is done using python as the programming language and the Google Colab / Jupyter Notebook as the platform for executing the code. The required libraries and python modules for this task to be done can be divided into three categories, those are for imputing techniques, machine learning models and evaluation metrics and also including the common libraries like numpy, pandas and matplotlib. Some of the main libraries that are necessary:

fancyimpute, impute- these python modules are used for handling the missing data like imputing missing values

KFold- to perform the cross-validation, for model evaluation

SMOTE- for class imbalance

Sklearn modules such as RandomForestClassifier, LogisticRegression ,DecisionTreeClassifier, GaussianNB, XGBClassifier.

Evaluation metrics like accuracy\_score, precision\_score, recall\_score to evaluate the model's performance.

### Machine learning models/algorithms used are

1.Gaussian Naïve Bayes Classifier-Naive Bayes classifier is one of the supervised learning algorithms which is based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.

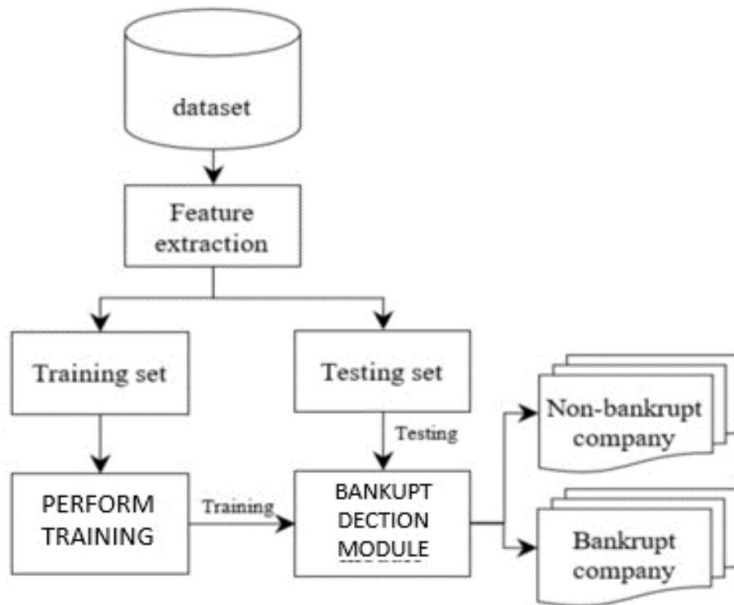
2. Logistic Regression Classifier-In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. In our implementation, we used a lambda value of 1 and applied L1 regularization. This means that we added a penalty term to the model's cost function, which encourages the model to select only the most important features and reduce overfitting. We also assigned equal weights to all the features, meaning that each feature contributes equally to the model's prediction. This approach is often used when there is no prior knowledge about the importance of the features or when we want to avoid introducing biases into the model.

3. Decision Trees Classifier-For our classification task, we create a model that predicts the value of a target variable ( $y$  = will a firm go bankrupt?) by learning simple decision rules inferred from the data features ( $x_1, x_2, \dots, x_{64}$ - all the financial distress variables of a firm). In our model, we take into account all the features available and assign equal weights to them when searching for the best split during the construction of a decision tree. This means that we evaluate each feature equally and choose the one that provides the most significant reduction in the 'Gini' index as the best split point.

4. Random Forest Classifier-A random forest is a meta estimator that fits a number of decision tree classifiers on various subsamples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. Our model uses 5 estimators, which are individual decision trees that work together to make a final prediction. During the construction of each individual decision tree, we use 'Entropy' as a measure of the quality of the split.

5. Extreme Gradient Boosting Classifier- It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. In our model, we have used 100 estimators, which are individual decision trees that work together to make a final prediction. To regularize the model and prevent overfitting, we have used a log-linear classifier with a regularization parameter (lambda) of 1.

### Design Flowchart:



### SMOTE -data imputation technique

One of the important designs of our project is SMOTE analysis for the data. As you can see in the below picture the code for SMOTE(Synthetic Minority Over-sampling Technique), it performs SMOTE oversampling on a set of imputed dataframes and returns the resulting oversampled data frames in a dictionary, where the keys are the names of the imputation methods and the values are lists of oversampled dataframes.

```
[ ] # Split the features and labels into separate dataframes for all the original dataframes
def split_dataframes_features_labels(dfs):
    feature_dfs = [dfs[i].iloc[:,0:64] for i in range(len(dfs))]
    label_dfs = [dfs[i].iloc[:,64] for i in range(len(dfs))]
    return feature_dfs, label_dfs

# Performs the SMOTE oversampling fro given dataframes.
def oversample_data_SMOTE(dfs, verbose=False):
    smote = SMOTE(sampling_strategy='auto' , random_state=42, k_neighbors=10)
    #Split the features and labels for each dataframe
    feature_dfs, label_dfs = split_dataframes_features_labels(dfs)
    resampled_feature_arrays = []
    resampled_label_arrays = []
    for i in range(len(dfs)):
        if verbose: print('Dataset: ' + str(i+1) + 'year:')
        if verbose: print('Original dataset shape {}'.format(Counter(label_dfs[i])))
        dfi_features_res, dfi_label_res = smote.fit_resample(feature_dfs[i], label_dfs[i])
        if verbose: print('Resampled dataset shape {}'.format(Counter(dfi_label_res)))
```

The screenshot shows a Jupyter Notebook interface with a file named 'project\_code.ipynb'. The code is titled '3.C.a Oversampling with SMOTE (Synthetic Minority Over Sampling Technique)'. It defines two functions: 'split\_dataframes\_features\_labels' which splits dataframes into features and labels, and 'oversample\_data\_SMOTE' which applies SMOTE oversampling. The code includes verbose print statements for debugging. The notebook status bar at the bottom indicates '0s completed at 1:26 AM'.

After applying SMOTE oversampling, the number of samples in the minority class (1.0) is increased to match the number of samples in the majority class (0.0), resulting in a balanced dataset with an equal representation of both classes as you can see below.



```
project_code.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
SMOTE Oversampling for Mean imputed dataframes

Dataset: 1year:
Original dataset shape Counter({0.0: 6756, 1.0: 271})
Resampled dataset shape Counter({0.0: 6756, 1.0: 6756})

Dataset: 2year:
Original dataset shape Counter({0.0: 9773, 1.0: 400})
Resampled dataset shape Counter({0.0: 9773, 1.0: 9773})

Dataset: 3year:
Original dataset shape Counter({0.0: 10008, 1.0: 495})
Resampled dataset shape Counter({0.0: 10008, 1.0: 10008})

Dataset: 4year:
Original dataset shape Counter({0.0: 9277, 1.0: 515})
Resampled dataset shape Counter({0.0: 9277, 1.0: 9277})

Dataset: 5year:
Original dataset shape Counter({0.0: 5500, 1.0: 410})
Resampled dataset shape Counter({0.0: 5500, 1.0: 5500})

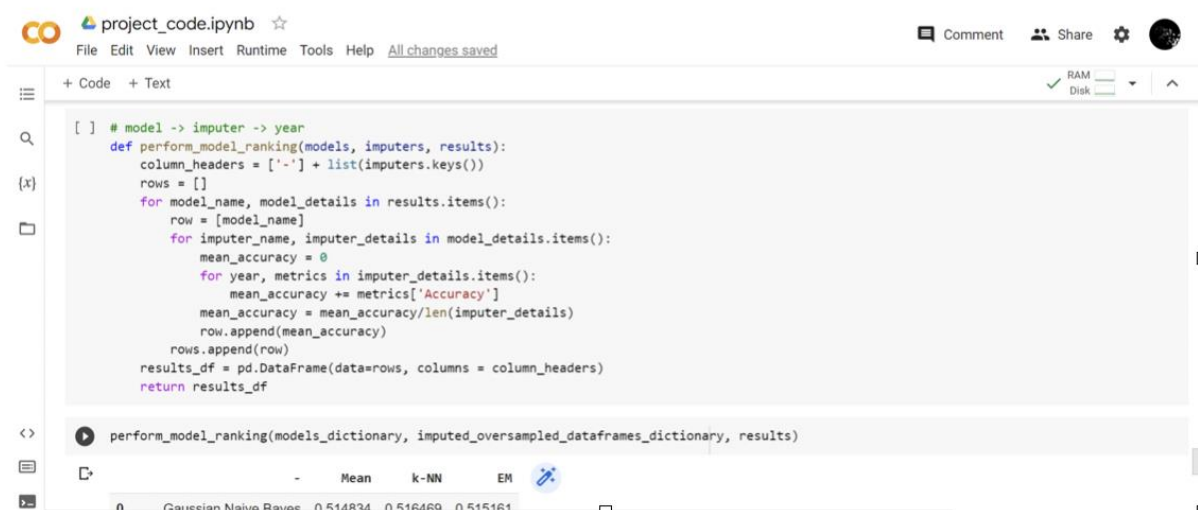
SMOTE Oversampling for k-NN imputed dataframes
```

Other techniques like k-nn, mean, EM imputation are also used for data pre-processing.

### Evaluation metrics:

The metrics for this project are accuracy, precision, recall to evaluate the model performance. Precision is used to measure the accuracy of positive predictions made by a model while recall is used to measure the ability of a model to correctly identify positive instances from the total actual positive instances.

But for ranking the model, we've only used accuracy metric as you can see in the image below.



```
project_code.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
[ ] # model -> imputer -> year
def perform_model_ranking(models, imputers, results):
    column_headers = ['-'] + list(imputers.keys())
    rows = []
    for model_name, model_details in results.items():
        row = [model_name]
        for imputer_name, imputer_details in model_details.items():
            mean_accuracy = 0
            for year, metrics in imputer_details.items():
                mean_accuracy += metrics['Accuracy']
            mean_accuracy = mean_accuracy/len(imputer_details)
            row.append(mean_accuracy)
        rows.append(row)
    results_df = pd.DataFrame(data=rows, columns = column_headers)
    return results_df

perform_model_ranking(models_dictionary, imputed_oversampled_dataframes_dictionary, results)
```

The most important functions in our code would be the different types of machine learning



algorithms that we've used and the metrics like accuracy, precision and recall for evaluating their performance. Our coding is done with displaying the best algorithm out of 5 that we've chosen and a bar plot is also shown with the algorithms against the 3 imputation techniques based on the accuracy as you can see in the image below of the model rankings.



The perspective of our project / code from the user point of view would be that our datasets are in .arff formats with object data types and the first step we have done is to import and organize the data, the second step is data analysis and pre-processing (using imputation techniques), and then data modeling using 5 classifiers and finally model analysis and visualizing the results. From the results the users/readers can know that extreme gradient boosting algorithm have the more accuracy among others which means it classifies the bankrupt companies and non-bankrupt companies accurately high among other classifiers.

## Project Milestones

There are some achievements that mark significant progress towards project completion. They are listed below:

### Milestones:

1. Data is collected as .arff files from UCI repository
2. Finished Implementation of code and tested it's functionality
3. Implemented more than 2 data imputation techniques and executed all of them successfully.
4. Used 5 supervised machine learning algorithms and compared them to find the best fit that results in high performance.
5. Conducted weekly meetings to discuss the progress for any further enhancements.

### Incremental Features:

1. Added imputed techniques to improve performance
2. Implemented SMOTE analysis to deal with data imbalance issues.
3. Created code for manually choosing the files rather than uploading files.
4. Implemented Sparsity matrix, correlation matrix to find the relationship between missing values in the dataset.

## Project Results

Our findings are structured in the following way: Initially, we present the accuracy scores of five models that we have tested. We display accuracy in a graph that plots each of the three imputation(Mean, KNN, EM) techniques against the five models(Naive Bayes' , Random Forest, Decision Tree, Logistic Regression, XGBoost) used.

As noted in the results section, the Gradient Boost Classifier with Mean Imputation method is the most effective model that we have tested. Table 3 presents an overview of the mean accuracies of different models and imputation methods across all datasets from each year. We were surprised to find that the Mean imputation technique outperformed other more sophisticated techniques like EM, despite being a basic method. The EXtreme Gradient Boosting model emerged as the most accurate model for all imputation techniques, with Random Forest coming in second place. Third place is Decision Tree. Our analysis in the results section showed that Logistic Regression performed slightly better than the Gaussian Naïve Bayes model, which ranked fourth. Finally, Gaussian Naïve Bayes model had the worst performance among all models, which ranked fifth.

**Table 3 :** Accuracies over all years' given in dataset

	Mean	K-NN	EM
<b>Gaussian Naïve Bayes</b>	0.514834	0.516469	0.515161
<b>Logistic Regression</b>	0.715885	0.717098	0.706942
<b>Decision Tree</b>	0.929784	0.904210	0.923568
<b>Random Forest</b>	0.953388	0.937045	0.945997
<b>ExtremeGradient Boosting</b>	0.989260	0.985345	0.989093

## **Conclusion**

This section provides a summary of the work done in the project so far. The project involved building five classification models, namely Gaussian Naïve Bayes, Logistic Regression, Decision Trees, Random Forests, and Extreme Gradient Boosting classifiers. The training data was balanced by oversampling the minority class labels using Synthetic Minority Oversampling technique. Missing values in the data were handled by using three imputation techniques: Mean, k-Nearest Neighbors (k-NN) and, Expectation-Maximization (EM). The main challenge faced in the project was dealing with missing or sparse data, which is a common issue in bankruptcy prediction because companies don't operate on the same timelines. The features used for bankruptcy prediction are not straightforward and require thorough study and validation. The findings of the project have been documented, and the best bankruptcy prediction model has been suggested.

## **Future work**

In this section, we aim to discuss potential areas for future research in bankruptcy prediction. It is possible to reduce the dimensionality of features, but in datasets like the Polish bankruptcy data with high missing data, it can be difficult to extract relevant features. Features that are dropped might have a significant impact on predictions if the data was less sparse. Therefore, if future bankruptcy prediction data can be collected with less sparsity, all the techniques mentioned in this section could be applied to generate more accurate predictive models.

## **Repository / Archive:**

The github link to access all our work is provided.

<https://github.com/saipriya128/Project5502>

## **Appendix:**

### **Code:**

#### **Installing the libraries:**

```
pip install fancyimpute
```

```
pip install impyute
```

#### **importing the modules:**

```
import warnings
```

```
warnings.filterwarnings("ignore")
```

```
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
import numpy as np
```

```
import pandas as pd
```

```
%matplotlib inline
```

```
from scipy.io import arff
```

```
import missingno as msno
```

```
import fancyimpute
```

```

import impyute as impy
from sklearn.impute import SimpleImputer
from sklearn.model_selection import KFold
from collections import Counter
from collections import OrderedDict
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import random
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
Importing and organizing the data
from scipy.io import arff
def load_arff_raw_data():
    N=5
    from google.colab import files
    uploaded = files.upload()
    return [arff.loadarff(str(i+1) + 'year.arff') for i in range(N)]
def load_dataframes():
    return [pd.DataFrame(data_i_year[0]) for data_i_year in load_arff_raw_data()]
def set_new_headers(dataframes):
    cols = ['X' + str(i+1) for i in range(len(dataframes[0].columns)-1)]
    cols.append('Y')
    for df in dataframes:
        df.columns = cols
    dataframes = load_dataframes()

```

**# Set the new headers for the dataframes. The new headers will have the renamed set of feature (X1 to X64)**

```

set_new_headers(dataframes)
# print the first 5 rows of a dataset 'year1'
dataframes[0].head()

```

**# Convert the dtypes of all the columns (other than the class label columns) to float.**

```

def convert_columns_type_float(dfs):
    for i in range(5):
        index = 1
        while(index<=63):
            colname = dfs[i].columns[index]
            col = getattr(dfs[i], colname)
            dfs[i][colname] = col.astype(float)
            index+=1

```

```

convert_columns_type_float(dataframes)

```

**# The class labels for all the dataframes are originally in object type.**

**# Convert them to int types**

```

def convert_class_label_type_int(dfs):
    for i in range(len(dfs)):
        col = getattr(dfs[i], 'Y')
        dfs[i]['Y'] = col.astype(int)

```

```

convert_class_label_type_int(dataframes)

```

Data Analysis and Preprocessing

**# Get Clean dataframes by dropping all the rows which have missing values**

```
def drop_nan_rows(dataframes, verbose=False):
    clean_dataframes = [df.dropna(axis=0, how='any') for df in dataframes]
    if verbose:
        for i in range(len(dataframes)):
            print(str(i+1)+'year:', 'Original Length=', len(dataframes[i]), '\tCleaned Length=', len(clean_dataframes[i]), '\tMissing Data=', len(dataframes[i])-len(clean_dataframes[i]))
    return clean_dataframes
```

**# Doing a quick analysis of how many missing values are there in each of the 5 dataframes**

```
nan_dropped_dataframes = drop_nan_rows(dataframes, verbose=True)
```

**# generate the sparsity matrix (figure) for all the dataframes**

```
def generate_sparsity_matrix(dfs):
    for i in range(5):
        missing_df_i = dfs[i].columns[dfs[i].isnull().any()].tolist()
        msno.matrix(dfs[i][missing_df_i], figsize=(20,5))
```

```
generate_sparsity_matrix(dataframes)
```

**# generate the heatmap for all the dataframes**

```
def generate_heatmap(dfs):
    for i in range(5):
        missing_df_i = dfs[i].columns[dfs[i].isnull().any()].tolist()
        msno.heatmap(dfs[i][missing_df_i], figsize=(20,20))
```

```
generate_heatmap(dataframes)
```

**Mean Imputation**

```
def perform_mean_imputation(dfs):
    # Construct an imputer with strategy as 'mean', to mean-impute along the columns
    imputer = SimpleImputer(missing_values=np.nan, strategy='mean', fill_value=None, verbose=0, copy=True)
    mean_imputed_dfs = [pd.DataFrame(imputer.fit_transform(df)) for df in dfs]
    for i in range(len(dfs)):
        mean_imputed_dfs[i].columns = dfs[i].columns
    return mean_imputed_dfs
```

```
mean_imputed_dataframes = perform_mean_imputation(dataframes)
```

**k-Nearest Neighbors (k-NN) Imputation**

```
def perform_knn_imputation(dfs):
    knn_imputed_datasets = [fancyimpute.KNN(k=100, verbose=True).fit_transform(dfs[i]) for i in range(len(dfs))]
    return [pd.DataFrame(data=knn_imputed_datasets[i]) for i in range(len(dfs))]
```

```
knn_imputed_dataframes = perform_knn_imputation(dataframes)
```

```
set_new_headers(knn_imputed_dataframes)
```

**Expectation-Maximization (EM) Imputation**

```
def perform_EM_imputation(dfs):
    em_imputed_datasets = [impy.imputation.cs.em(dfs[i].values, loops=50, dtype='cont') for i in range(len(dfs))]
    return [pd.DataFrame(data=em_imputed_datasets[i]) for i in range(len(dfs))]
```

```
em_imputed_dataframes = perform_EM_imputation(dataframes)
```

```
set_new_headers(em_imputed_dataframes)
```

```
imputed_dataframes_dictionary = OrderedDict()
```

```
imputed_dataframes_dictionary['Mean'] = mean_imputed_dataframes
```

```
imputed_dataframes_dictionary['k-NN'] = knn_imputed_dataframes
```

```
imputed_dataframes_dictionary['EM'] = em_imputed_dataframes
```

```
def check_data_imbalance(dfs):
```

```
    for i in range(len(dfs)):
        print('Dataset: '+str(i+1)+'year')
```

```

print(dfs[i].groupby('Y').size())
minority_percent = (dfs[i]['Y'].tolist().count(1) / len(dfs[i]['Y'].tolist()))*100
print('Minority (label 1) percentage: ' + str(minority_percent) + '%')
print('-'*64)

```

check\_data\_imbalance(dataframes)

### **Oversampling with SMOTE (Synthetic Minority Over Sampling Technique)**

**# Split the features and labels into separate dataframes for all the original dataframes**

```

def split_dataframes_features_labels(dfs):
    feature_dfs = [dfs[i].iloc[:,0:64] for i in range(len(dfs))]
    label_dfs = [dfs[i].iloc[:,64] for i in range(len(dfs))]
    return feature_dfs, label_dfs

```

**# Performs the SMOTE oversampling for given dataframes.**

```

def oversample_data_SMOTE(dfs, verbose=False):
    smote = SMOTE(sampling_strategy='auto', random_state=42, k_neighbors=10)
    #Split the features and labels for each dataframe
    feature_dfs, label_dfs = split_dataframes_features_labels(dfs)
    resampled_feature_arrays = []
    resampled_label_arrays = []
    for i in range(len(dfs)):
        if verbose: print('Dataset: ' + str(i+1) + ' year:')
        if verbose: print('Original dataset shape {}'.format(Counter(label_dfs[i])))
        dfi_features_res, dfi_label_res = smote.fit_resample(feature_dfs[i], label_dfs[i])
        if verbose: print('Resampled dataset shape {}'.format(Counter(dfi_label_res)))
        # Append the resampled feature and label arrays of ith dataframe to their respective list of arrays
        resampled_feature_arrays.append(dfi_features_res)
        resampled_label_arrays.append(dfi_label_res)
    return resampled_feature_arrays, resampled_label_arrays

def restructure_arrays_to_dataframes(feature_arrays, label_arrays):
    resampled_dfs = []
    for i in range(len(feature_arrays)):
        feature_df = pd.DataFrame(data=feature_arrays[i])
        label_df = pd.DataFrame(data=label_arrays[i])
        label_df.columns=['Y']
        resampled_dfs.append(feature_df.join(label_df))
    set_new_headers(resampled_dfs)
    return resampled_dfs

```

**# Perform SMOTE oversampling on all the imputed dataframes, and return them in a dictionary.**

```

def perform_oversampling_on_imputed_dataframes(df_dict):
    imputed_oversampled_dataframes_dictionary = OrderedDict()
    for key,dfs in df_dict.items():
        print('SMOTE Oversampling for ' + key + ' imputed dataframes\n')
        smote_feature_arrays, smote_label_arrays = oversample_data_SMOTE(dfs, verbose=True)
        oversampled_dataframes = restructure_arrays_to_dataframes(smote_feature_arrays, smote_label_arrays)
        imputed_oversampled_dataframes_dictionary[key] = oversampled_dataframes
        print('-'*100)
    return imputed_oversampled_dataframes_dictionary

```

```

imputed_oversampled_dataframes_dictionary = perform_oversampling_on_imputed_dataframes(imputed_dataframes_dictionary)

```

### **Data Modeling: Building Classification Models**

#### **K-Fold Cross Validation**

```

def prepare_kfold_cv_data(k, X, y, verbose=False):
    X = X.values

```

```

y = y.values
kf = KFold(n_splits=k, shuffle=True, random_state=42)
X_train = []
y_train = []
X_test = []
y_test = []

for train_index, test_index in kf.split(X):
    X_train.append(X[train_index])
    y_train.append(y[train_index])
    X_test.append(X[test_index])
    y_test.append(y[test_index])
return X_train, y_train, X_test, y_test

```

**MODELS**

**# Gaussian Naive Bayes classifier**  
gnb\_classifier = GaussianNB()

**# Logistic Regression classifier**  
lr\_classifier = LogisticRegression(C=0.01,penalty='l1',solver='liblinear', random\_state = 0)

**# Decision Tree Classifier**  
dt\_classifier = DecisionTreeClassifier(random\_state=42)

**# Random Forest Classifier**  
rf\_classifier = RandomForestClassifier(n\_estimators = 5, criterion = 'entropy')

**# eXtreme Gradient Boosting Classifier (XGBClassifier)**  
xgb\_classifier = XGBClassifier()

**# creating a dictionary of models**  
models\_dictionary = OrderedDict()

models\_dictionary['Gaussian Naive Bayes'] = gnb\_classifier  
models\_dictionary['Logistic Regression'] = lr\_classifier  
models\_dictionary['Decision Tree'] = dt\_classifier  
models\_dictionary['Extreme Gradient Boosting'] = xgb\_classifier  
models\_dictionary['Random Forest'] = rf\_classifier

**# perform data modeling**  
def perform\_data\_modeling(\_models\_, \_imputers\_, verbose=False, k\_folds=5):

# 5 Models  
# 3 Imputers  
# 5 datasets (for 5 years)  
# 3 metrics, averaged over all the K-Folds  
model\_results = OrderedDict()

# Iterate over the models  
for model\_name, clf in \_models\_.items():  
 if verbose: print("-"\*120, "\n", "Model: " + '\033[1m' + model\_name + '\033[0m' + " Classifier")  
 imputer\_results = OrderedDict()  
 for imputer\_name, dataframes\_list in \_imputers\_.items():  
 if verbose: print("\tImputer Technique: " + '\033[1m' + imputer\_name + '\033[0m')  
 feature\_dfs, label\_dfs = split\_dataframes\_features\_labels(dataframes\_list)

year\_results = OrderedDict()

# Iterate over dataframe\_list individually  
for df\_index in range(len(dataframes\_list)):  
 if verbose: print("\t\tDataset: " + '\033[1m' + str(df\_index+1) + 'year' + '\033[0m')  
 X\_train\_list, y\_train\_list, X\_test\_list, y\_test\_list = prepare\_kfold\_cv\_data(k\_folds, feature\_dfs[df\_index],  
label\_dfs[df\_index], verbose)

```

metrics_results = OrderedDict()
accuracy_list = np.zeros([k_folds])
precision_list = np.zeros([k_folds,2])
recall_list = np.zeros([k_folds,2])
TN_list = np.zeros([k_folds])
FP_list = np.zeros([k_folds])
FN_list = np.zeros([k_folds])
TP_list = np.zeros([k_folds])

# Iterate over all the k-folds
for k_index in range(k_folds):
    X_train = X_train_list[k_index]
    y_train = y_train_list[k_index]
    X_test = X_test_list[k_index]
    y_test = y_test_list[k_index]

    # Fit the model and
    clf = clf.fit(X_train, y_train)
    y_test_predicted = clf.predict(X_test)

    #code for calculating accuracy
    _accuracy_ = accuracy_score(y_test, y_test_predicted, normalize=True)
    accuracy_list[k_index] = _accuracy_

    #code for calculating recall
    _recalls_ = recall_score(y_test, y_test_predicted, average=None)
    recall_list[k_index] = _recalls_

    #code for calculating precision
    _precisions_ = precision_score(y_test, y_test_predicted, average=None)
    precision_list[k_index] = _precisions_

# creating a metrics dictionary
metrics_results['Accuracy'] = np.mean(accuracy_list)
metrics_results['Precisions'] = np.mean(precision_list, axis=0)
metrics_results['Recalls'] = np.mean(recall_list, axis=0)
metrics_results['TN'] = np.mean(TN_list)
metrics_results['FP'] = np.mean(FP_list)
metrics_results['FN'] = np.mean(FN_list)
metrics_results['TP'] = np.mean(TP_list)

if verbose:
    print("\t\t\tAccuracy:', metrics_results['Accuracy'])
    print("\t\t\tPrecision:', metrics_results['Precisions'])
    print("\t\t\tRecall:', metrics_results['Recalls'])

year_results[str(df_index+1)+'year'] = metrics_results

imputer_results[imputer_name] = year_results

model_results[model_name] = imputer_results

return model_results

```



```
results = perform_data_modeling(models_dictionary, imputed_oversampled_dataframes_dictionary, verbose=True,
k_folds=5)
```

### Model Analysis

```
# model -> imputer -> year
```

```
def perform_model_ranking(models, imputers, results):
    column_headers = ['-'] + list(imputers.keys())
    rows = []
    for model_name, model_details in results.items():
        row = [model_name]
        for imputer_name, imputer_details in model_details.items():
            mean_accuracy = 0
            for year, metrics in imputer_details.items():
                mean_accuracy += metrics['Accuracy']
            mean_accuracy = mean_accuracy/len(imputer_details)
            row.append(mean_accuracy)
        rows.append(row)
    results_df = pd.DataFrame(data=rows, columns = column_headers)
    return results_df
```

```
perform_model_ranking(models_dictionary, imputed_oversampled_dataframes_dictionary, results)
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
X = ['Gaussian','LR','Decision','EG', 'Random Forest']
mean_y= [0.514834,0.715885,0.929784,0.989260,0.953388]
knn_y= [0.516469,0.717196,0.904210,0.985345,0.937045]
em_y=[0.515161,0.706942,0.923568,0.989093,0.945997]
```

```
X_axis = np.arange(len(X))
```

```
plt.bar(X_axis,mean_y, 0.4, label = 'Mean')
plt.bar(X_axis + 0.25,knn_y, 0.4, label = 'KNN')
plt.bar(X_axis + 0.5,em_y, 0.4, label = 'EM')
```

```
plt.xticks(X_axis, X)
plt.xlabel("Algorithms")
plt.ylabel("Imputations")
plt.title("Graph of Accuracy")
plt.legend()
plt.show()
```

## References

1.Altman, E.L, "Financial Ratios, Discriminant Analysis and the Prediction of Corporate Bankruptcy", The Journal of Finance, Vol. 23(4), pp.589-609, 1968

(In this paper, Altman proposed a statistical model using discriminant analysis to predict the likelihood of corporate bankruptcy based on financial ratios)

2. Talha Mahboob Alam, "Corporate Bankruptcy Prediction: An Approach Towards Better Corporate World", The Computer Journal, Vol. 64(11), pp.1731–1746, 2021

(The paper presents a method that incorporates machine learning techniques, specifically decision tree, random forest, and gradient boosting, for predicting the likelihood of a corporation facing bankruptcy.)

3. Maciej Zieba, "Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction", Expert Systems with Applications, Vol. 58, pp. 93-101, 2016

(The paper proposes a method that combines gradient boosting techniques with synthetic feature generation to improve the performance of bankruptcy prediction models)

We also used google/YouTube as a platform to know more about a particular technique or how it can be useful for the predictions , some of the links are provided below.

1. <https://youtu.be/HX>

(implementing KNN Imputer in Python, a technique by which we can effortlessly impute missing values in a dataset by looking at neighboring values.)

2. [https://youtu.be/dkXB8HH\\_4-k](https://youtu.be/dkXB8HH_4-k)

(this video explain about how to handle imbalanced datasets using SMOTE technique)

3. <https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes9fe30ff6776a?gi=c4dbec0b2d9>

(The blog post likely provides a brief overview or summary of various machine learning models in a concise manner, potentially covering a wide range of algorithms used in machine learning, such as linear regression, decision trees, random forests, support vector machines, k-nearest neighbors, naive Bayes, gradient boosting, and neural networks, among others)

**Our project is different from the papers which we have researched** or took reference from or we can say that those papers influenced us to do this project like we have adapted the gradient boosting techniques and also how to improve the model performance but apart from this we have proposed 3 different data cleaning techniques and one data imbalance technique SMOTE so the model performance can improve to give accurate results on classifying the bankrupt and non-bankrupt companies.