ANTICIPATING BUSINESS BANKRUPTCY

Bankruptcy prediction has been a subject of interest for almost a century, and it still ranks high among hottest topics in economics. The aim of predicting financial distress is to develop a predictive model that combines various econometric measures and allows us to foresee the financial condition of a firm.

The purpose of the bankruptcy prediction is to assess the financial condition of a company and its future perspectives within the context of long-term operation on the market.

The dataset is about bankruptcy prediction of Polish companies. The data was collected from Emerging Markets Information Service (EMIS), which is a database containing information on emerging markets around the world. The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated for year 2007.Basing on the collected data five classification cases were distinguished, that depends on the forecasting period: The data contains financial rates from 1st year of the forecasting period and corresponding class label that indicates bankruptcy status.

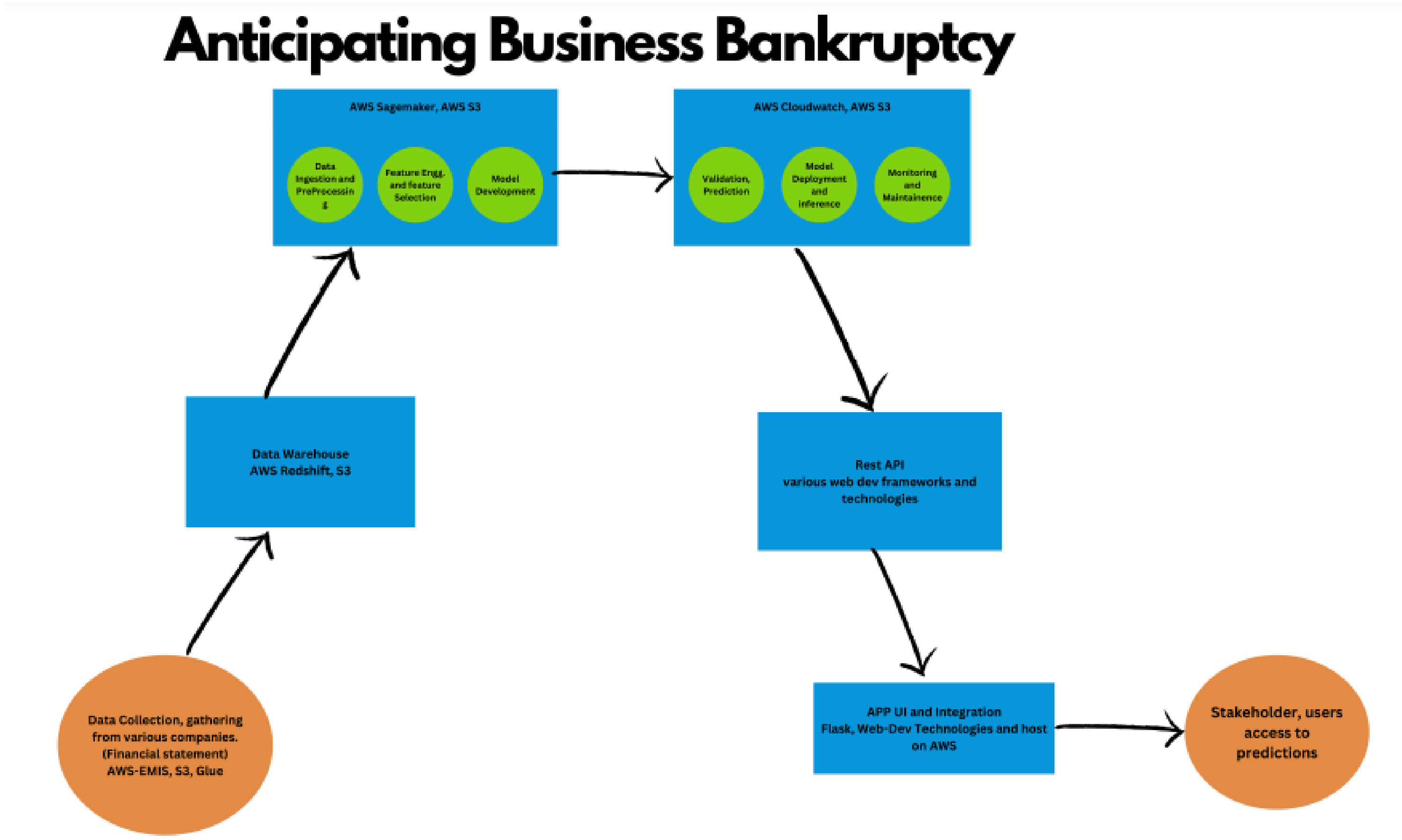
Aim:

The purpose of the bankruptcy prediction is to assess the financial condition of a company and its future perspectives within the context of long-term operation on the market.

Technical Architecture:

In this architecture, a user inputs the relevant engagement metrics such as ATTR1 to ATTR10 scores. The Flask application is responsible for handling the request and returning the prediction that if company can bankrupt or not in the forecasting period.

The Flask application then passes the inputs metrics to the trained algorithm model, which processes the input data and returns a predict based on the learned relationship between the input metrics and repayment interval.



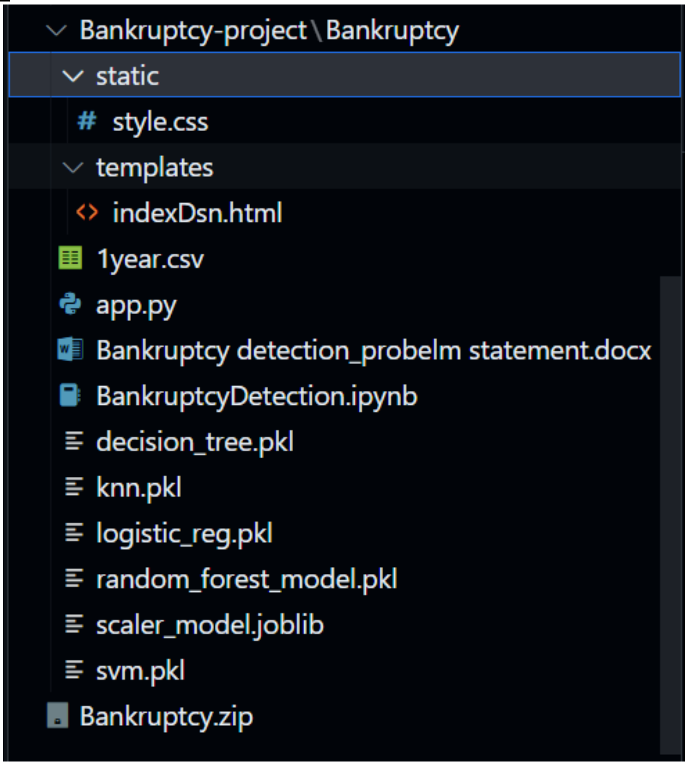
Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have completed all the activities listed below:

• Define problem / Problem understanding
o Specify the business problem
o Business Requirements
o Social or Business Impact
• Data Collection and Preparation
o Collect the dataset
o Data Preparation
• Exploratory Data Analysis
o Descriptive statistical
o Visual Analysis
• Model Building
o Train-test split
o Training and testing the Models using multiple algorithms
• Performance Testing
o Comparing model accuracy
o Graphical representation of the model comparison.
• Model Deployment
o Save the best model
o Test the model
o Integrate with Web Framework
o GUI
• Project Demonstration & Documentation

Project Structure:



- The data obtained is in csv files is split for training and testing.
- We have built a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- model.pkl is the saved model. This will further be used in the Flask integration.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

The aim of the bankruptcy prediction is to evaluate a company's financial situation and its prospects for the future in the context of its long-term activity on the market.

Activity 2: Business Requirements

1. **Financial Planning Enhancement**: Precise bankruptcy predictions support businesses in refining their financial planning strategies. This capability enables organizations to foresee potential issues with cash flow, pinpoint areas requiring improvement, and take proactive measures to enhance overall financial health.
2. **Early Warning Mechanism**: Bankruptcy prediction models act as an advanced warning system, notifying companies about potential financial distress before it reaches a critical stage. This early detection empowers organizations to promptly implement corrective measures, such as debt restructuring, contract renegotiation, or cost-saving initiatives.
3. **Risk Mitigation Focus**: Predicting bankruptcy not only helps in risk assessment but also shifts the focus towards actively mitigating these risks. Businesses can develop and execute strategies to address financial challenges, ensuring a more resilient and secure financial landscape.

Activity 4: Social or Business Impact.

Social Impact:

- **Job losses:** When a company goes bankrupt, it often leads to layoffs and job losses for its employees. Predicting bankruptcy helps mitigate the impact by allowing employees to prepare for potential job transitions in advance. It also enables governments, labor organizations, and affected individuals to develop strategies for reemployment and support.

- **Investor protection:** Bankruptcy prediction plays a vital role in investor protection. When investors, shareholders, or creditors can anticipate a company's financial distress, they can take appropriate measures to protect their investments or minimize losses. It enables them to diversify their portfolios, sell stocks, or adjust their financial positions in a timely manner.

Business Impact:

- **Industry dynamics and competition:** Bankruptcies can reshape industry dynamics and alter competitive landscapes. Predicting bankruptcy enables businesses to anticipate market shifts, identify potential acquisition opportunities, or adjust their strategies to gain a competitive advantage. It fosters a more dynamic business environment by facilitating adaptive responses to market changes.

- **Financial system stability:** The stability of the financial system is crucial for economic growth and prosperity. Predicting company bankruptcies contributes to financial system stability by identifying potential risks and vulnerabilities. It helps financial institutions assess their exposure, manage risk, and make informed lending decisions. By avoiding excessive risk concentration, the likelihood of systemic financial crises can be reduced.

Milestone 2: Data Collection and Preparation:

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

Activity 1: Collect the dataset

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: **https://www.kaggle.com/datasets/bhadaneeraj/bankruptcy-detection**

About this Data

Attribute Information:

Feature	Description	Financial term
Attr1	net profit / total assets	Profitability Ratio
Attr2	total liabilities / total assets	Leverage Ratio
Attr3	working capital / total assets	Efficiency Ratio
Attr4	current assets / short-term liabilities	Current Ratio
Attr5	[(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365	Cash Conversion Cycle
Attr6	retained earnings / total assets	Retention Ratio
Attr7	EBIT / total assets	EBIT Margin or Return on Assets (ROA)
Attr8	book value of equity / total liabilities	Equity Multiplier
Attr9	sales / total assets	Asset Turnover Ratio
Attr10	equity / total assets	Equity Ratio

Activity 1.1: Importing the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sbn
import pickle
import joblib

from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import zscore
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```python
df = pd.read_csv('1year.csv')
df.drop(df.iloc[:, 10:64], inplace=True, axis=1)
df
```

[8] ✓ 0.0s

	Attr1	Attr2	Attr3	Attr4	Attr5	Attr6	Attr7	Attr8	Attr9	Attr10	class
0	0.20055	0.37951	0.39641	2.0472	32.351	0.38825	0.24976	1.3305	1.1389	0.50494	0
1	0.20912	0.49988	0.47225	1.9447	14.786	0	0.25834	0.99601	1.6996	0.49788	0
2	0.24866	0.69592	0.26713	1.5548	-1.1523	0	0.30906	0.43695	1.309	0.30408	0
3	0.081483	0.30734	0.45879	2.4928	51.952	0.14988	0.092704	1.8661	1.0571	0.57353	0
4	0.18732	0.61323	0.2296	1.4063	-7.3128	0.18732	0.18732	0.6307	1.1559	0.38677	0
...	...	...	...	...	...	...	...	...	...	...	...
7007	0.038665	0.071884	0.48884	7.8004	221.01	0.038665	0.045892	11.068	1.0765	0.7956	1
7008	0.001091	0.8516	0.003463	1.0086	-44.467	0.086248	0.001091	0.17429	1.0297	0.14842	1
7009	-0.091442	0.7055	-0.047216	0.92568	-7.2952	0	-0.090374	0.41744	9.1345	0.2945	1
7010	0.13809	3.3357	-2.364	0.29128	-88.382	-3.3963	0.13809	-0.70021	9.9852	-2.3357	1
7011	0.098271	0.8333	0.000426	1.0005	-43.191	0	0.12838	0.20019	2.5144	0.16682	1

7012 rows × 11 columns

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Getting the Preliminary Information about the Dataset
- Handling missing values
- Dropping Unwanted column

Activity 2.1: Getting the Preliminary Information about the Dataset

i.e. Non-Null, Count , Dtype

Let's find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```python
print("Shape of the DataFrame:", df.shape)
print("Number of Columns:", len(df.columns))
print("Number of Rows:", len(df))
```

[9] ✓ 0.0s

```
Shape of the DataFrame: (7012, 11)
Number of Columns: 11
Number of Rows: 7012
```

```python
df.info()
```

[10] ✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7012 entries, 0 to 7011
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Attr1   7012 non-null   object
 1   Attr2   7012 non-null   object
 2   Attr3   7012 non-null   object
 3   Attr4   7012 non-null   object
 4   Attr5   7012 non-null   object
 5   Attr6   7012 non-null   object
 6   Attr7   7012 non-null   object
 7   Attr8   7012 non-null   object
 8   Attr9   7012 non-null   object
 9   Attr10  7012 non-null   object
 10  class   7012 non-null   int64 
dtypes: int64(1), object(10)
memory usage: 602.7+ KB
```

Activity 2.2: Handling missing values:

Check and number of missing values and its percentage for the all the columns and filled the missing values only for required columns (Input).

For checking the null values, df.isnull() function is used. To sum those null values, we use .sum() function.

```
# Checking for missing values
missing_values = df.isnull().sum()
print(missing_values)
```

[11] ✓ 0.0s

```
Attr1      0
Attr2      0
Attr3      0
Attr4      0
Attr5      0
Attr6      0
Attr7      0
Attr8      0
Attr9      0
Attr10     0
class      0
dtype: int64
```

When we checked for the " ? " in the dataset we can easily found it. So, the meaning of the "

? " was same as Null or NA, so we identified and replaced all the ? with NAN As we checked above, we were not able to see any missing values in all the column but there were some columns.

```
(df.eq('?')).any()
```

[9]

```
Attr1       True
Attr2       True
Attr3       True
Attr4       True
Attr5       True
Attr6       True
Attr7       True
Attr8       True
Attr9       True
Attr10      True
class      False
dtype: bool
```

```
(df.eq('?')).sum()
```

[10]

```
Attr1      3
Attr2      3
Attr3      3
Attr4     30
Attr5      8
Attr6      3
Attr7      3
Attr8     25
Attr9      1
Attr10     3
class      0
dtype: int64
```

Filling '?' with NaN and then doing the graphical analysis / EDA

```
# Checking for ? values and fill them
df.replace('?',np.NAN,inplace=True)
```

```
[12]
(df.eq('?')).sum()
```

```
Attr1      0
Attr2      0
Attr3      0
Attr4      0
Attr5      0
Attr6      0
Attr7      0
Attr8      0
Attr9      0
Attr10     0
class      0
dtype: int64
```

```
[13]
df.isnull().sum()
```

```
Attr1       3
Attr2       3
Attr3       3
Attr4      30
Attr5       8
Attr6       3
Attr7       3
Attr8      25
Attr9       1
Attr10      3
class       0
dtype: int64
```

Now we replaced the 82 null values from age column with the median of the column.

Filling in Missing values with median

```
[36]
df = df.fillna(df.median())
for column in df.columns:
    df[column] = df[column].fillna(df[column].median())
```

```
[37]
df.isnull().sum()
```

```
Attr1     0
Attr3     0
Attr4     0
Attr5     0
Attr6     0
Attr7     0
Attr8     0
Attr9     0
Attr10    0
class     0
dtype: int64
```

Activity 2.3: Dropping Unwanted column

In the dataset, we need all the columns, as every column is important for prediction

Milestone 3: Exploratory Data Analysis

Activity 3.1: Descriptive Statistical

Descriptive analysis is to study of the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this describe function we can understand the unique, top, and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
{column:len(df[column].unique()) for column in df.columns}
```

[20] ✓ 0.0s

```
{'Attr1': 6618,
 'Attr3': 6691,
 'Attr4': 6274,
 'Attr5': 6806,
 'Attr6': 4202,
 'Attr7': 6661,
 'Attr8': 6671,
 'Attr9': 5500,
 'Attr10': 6621,
 'class': 2}
```

```
print("Data Information")
df.info()
```

[21] ✓ 0.0s

```
Data Information
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7012 entries, 0 to 7011
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Attr1   7012 non-null   object
 1   Attr3   7012 non-null   object
 2   Attr4   7012 non-null   object
 3   Attr5   7012 non-null   object
 4   Attr6   7012 non-null   object
 5   Attr7   7012 non-null   object
 6   Attr8   7012 non-null   object
 7   Attr9   7012 non-null   object
 8   Attr10  7012 non-null   object
 9   class   7012 non-null   int64
dtypes: int64(1), object(9)
memory usage: 547.9+ KB
```

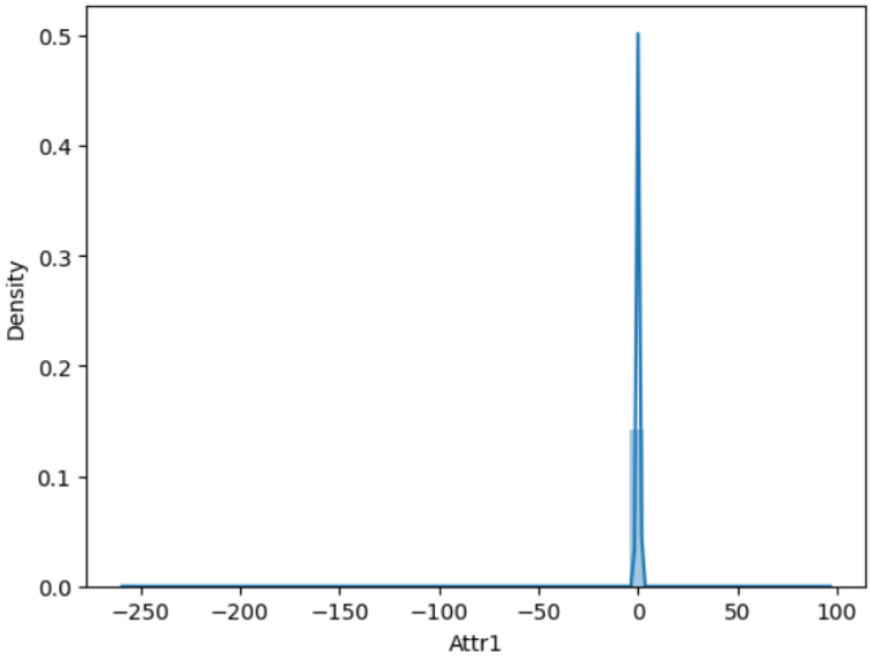
Activity 3.2: Visualisation

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

1. Univariate Analysis

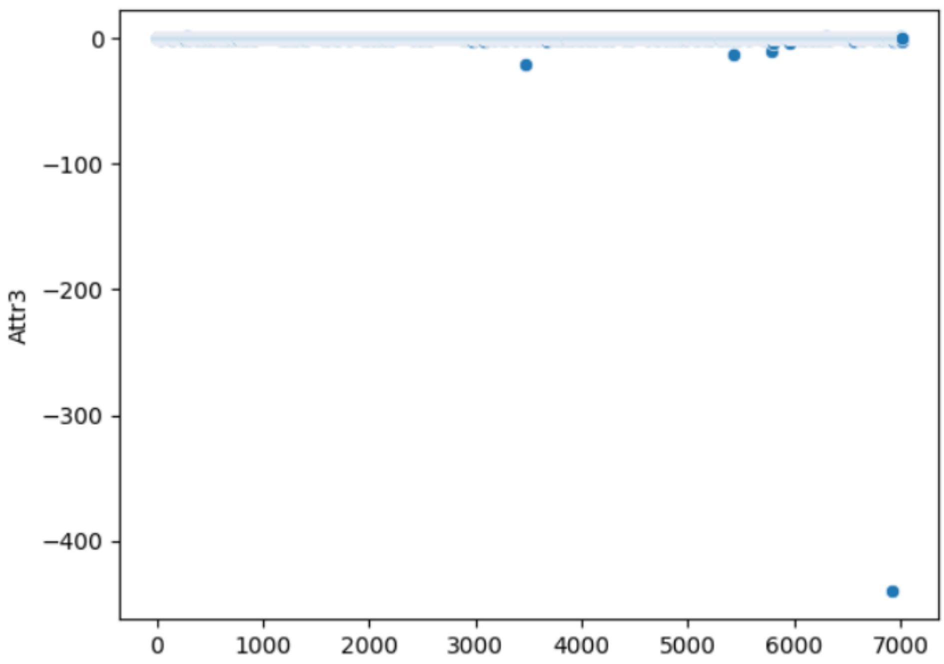
```
[16]: sbn.distplot(df['Attr1'])
```

```
[16]: <Axes: xlabel='Attr1', ylabel='Density'>
```



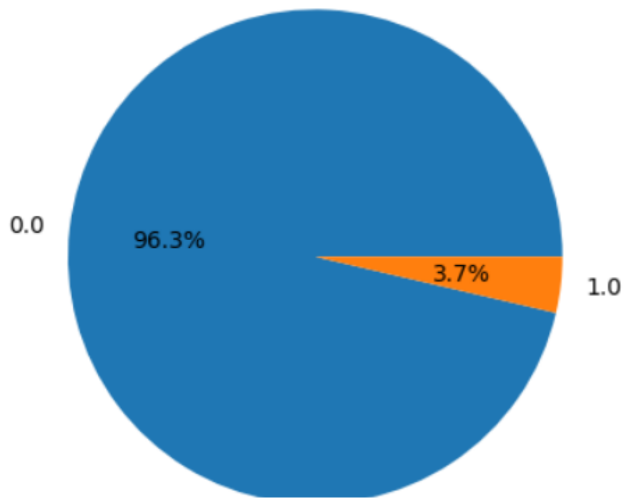

```
[18]: df = df.astype(float)
      sbn.scatterplot(df['Attr3'])
```

```
[18]: <Axes: ylabel='Attr3'>
```



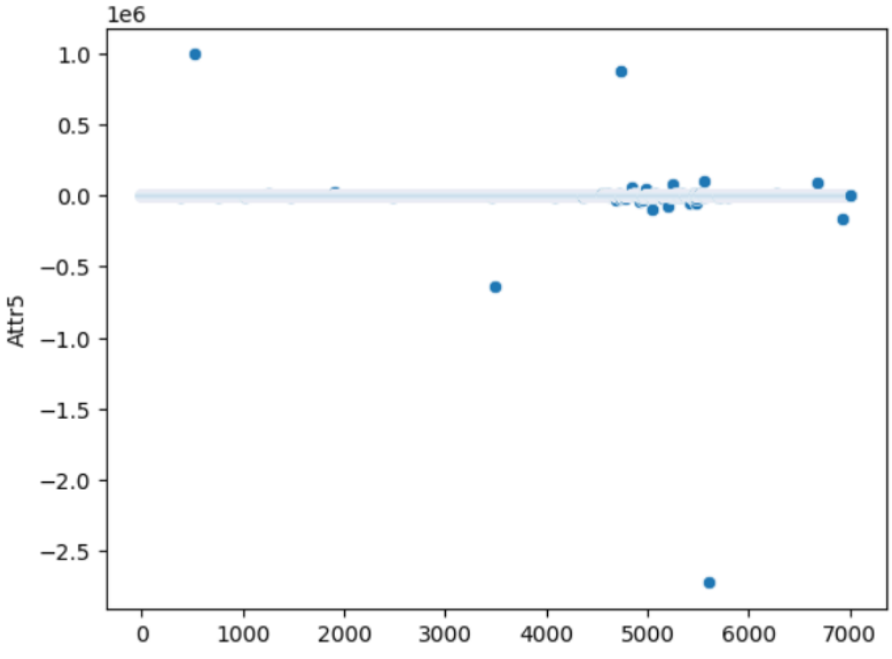
[19]:
```
plt.pie(df["class"].value_counts(), labels = df["class"].unique(),autopct ='%1.1f%%')
```

[19]:
```
([<matplotlib.patches.Wedge at 0x28beb60e210>,
  <matplotlib.patches.Wedge at 0x28beb66fb50>],
 [Text(-1.0927725884889667, 0.12588911727041197, '0.0'),
  Text(1.0927725825956707, -0.12588916842678877, '1.0')],
 [Text(-0.5960577755394363, 0.06866679123840651, '96.3%'),
  Text(0.5960577723249112, -0.06866681914188477, '3.7%')])
```



[20]:
```
sbn.scatterplot(df['Attr5'])
```

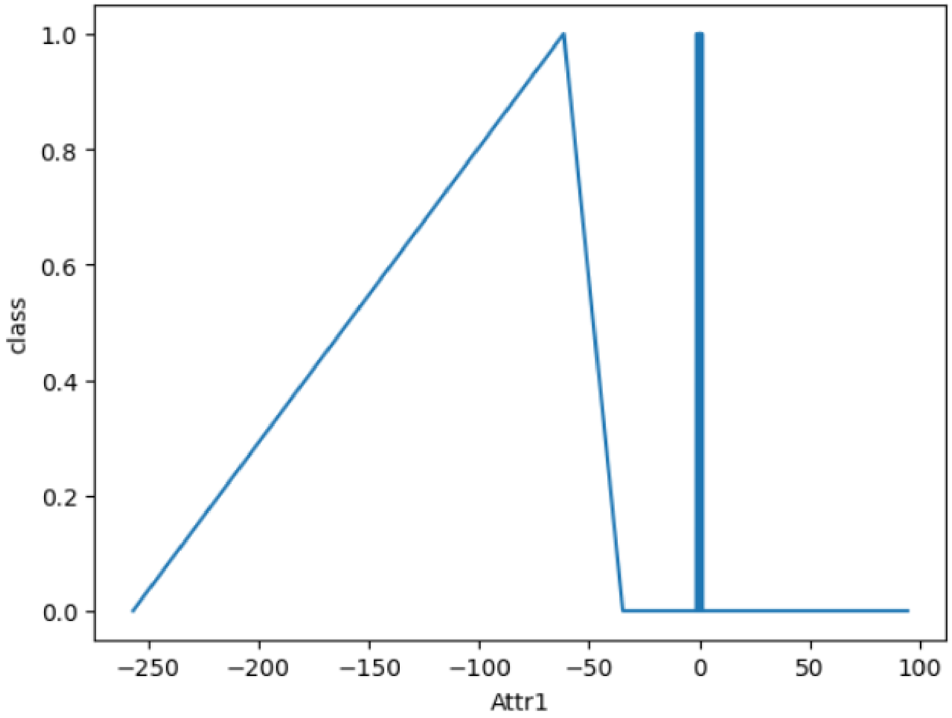
[20]:
```
<Axes: ylabel='Attr5'>
```



2. Bivariate Analysis

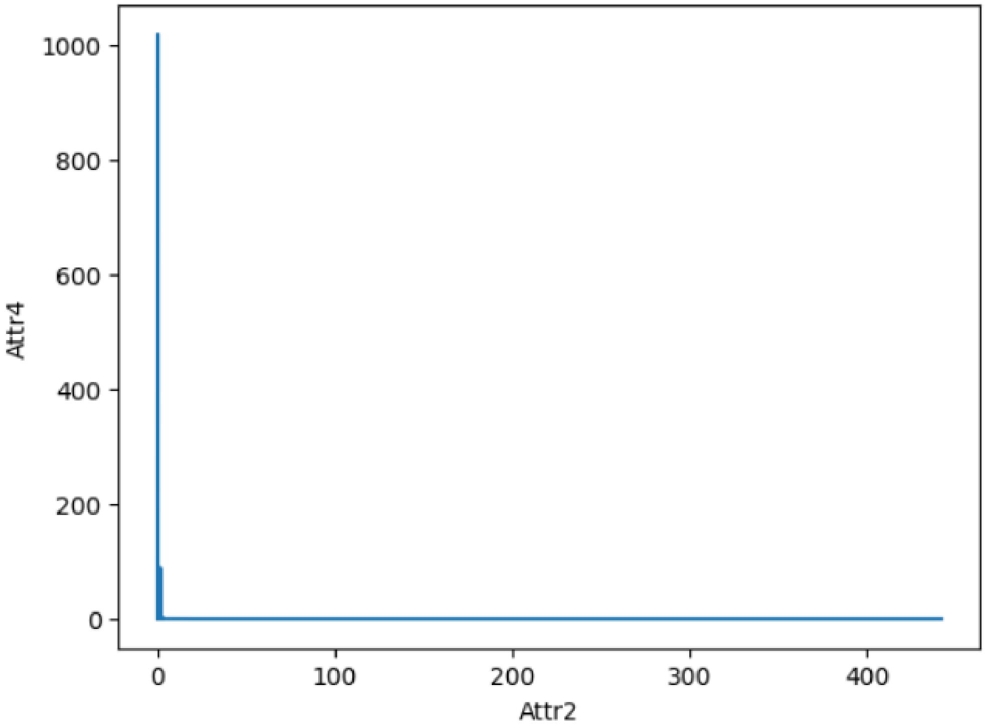
```
[24]: sbn.lineplot(x=df['Attr1'],y=df['class'])
```

```
[24]: <Axes: xlabel='Attr1', ylabel='class'>
```

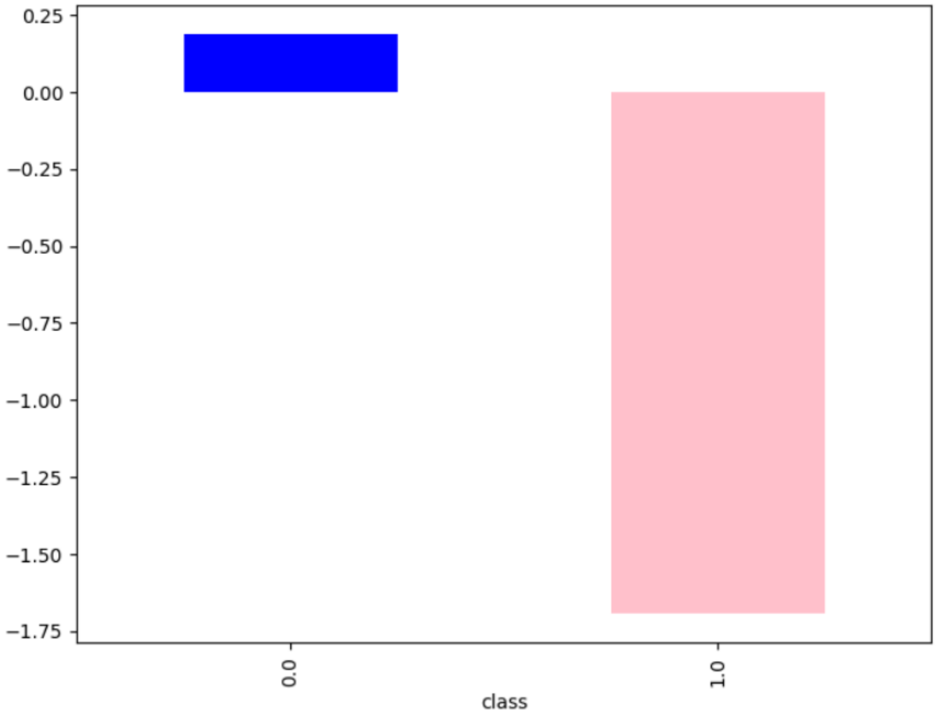



```
[25]: sbn.lineplot(x=df['Attr2'],y=df['Attr4'])
```

```
[25]: <Axes: xlabel='Attr2', ylabel='Attr4'>
```

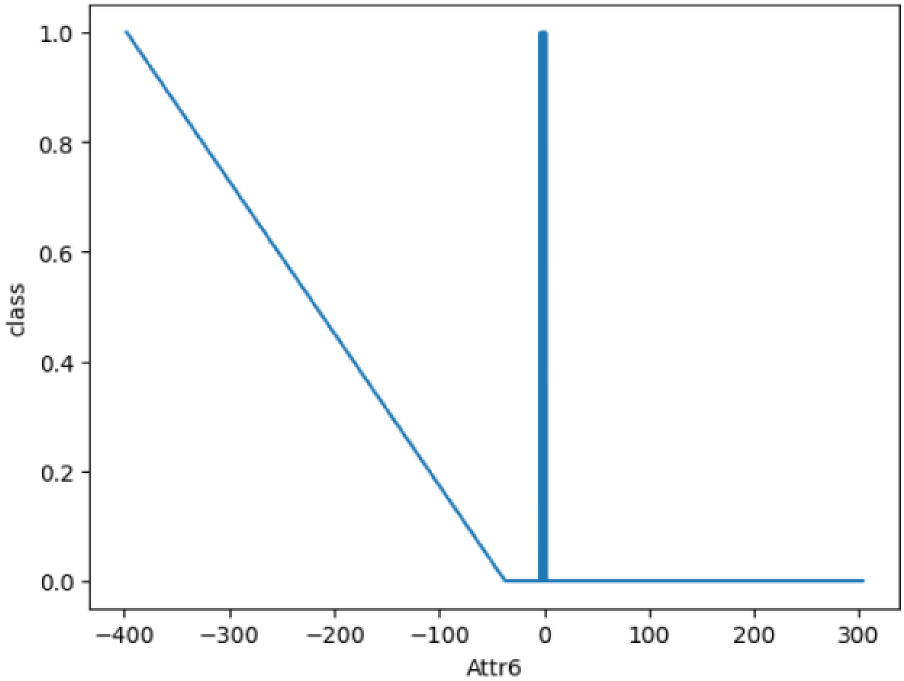


[26]:
```
grouped_data = df.groupby('class')['Attr3'].mean()
plt.figure(figsize=(8, 6))
grouped_data.plot(kind='bar', color=['blue', 'pink'])
plt.show()
```

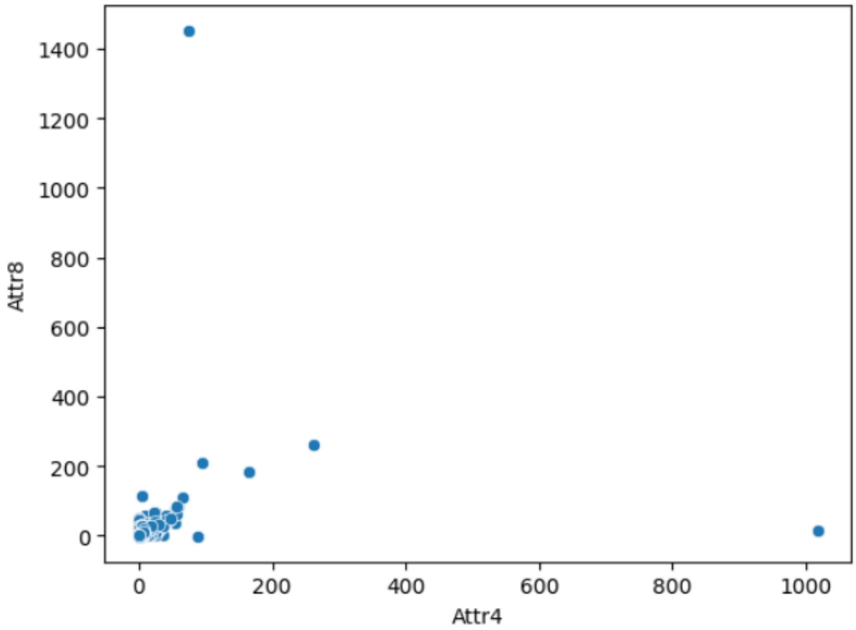


[27]:
```
sbn.lineplot(x=df['Attr6'],y=df['class'])
```

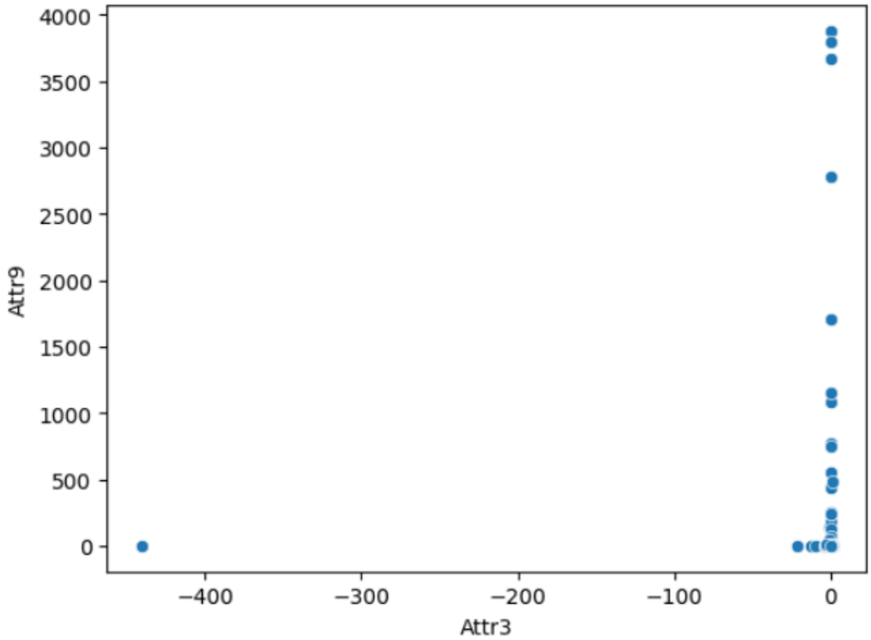
[27]: <Axes: xlabel='Attr6', ylabel='class'>



[28]:
```
sbn.scatterplot(x='Attr4',y='Attr8',data=df)
plt.show()
```

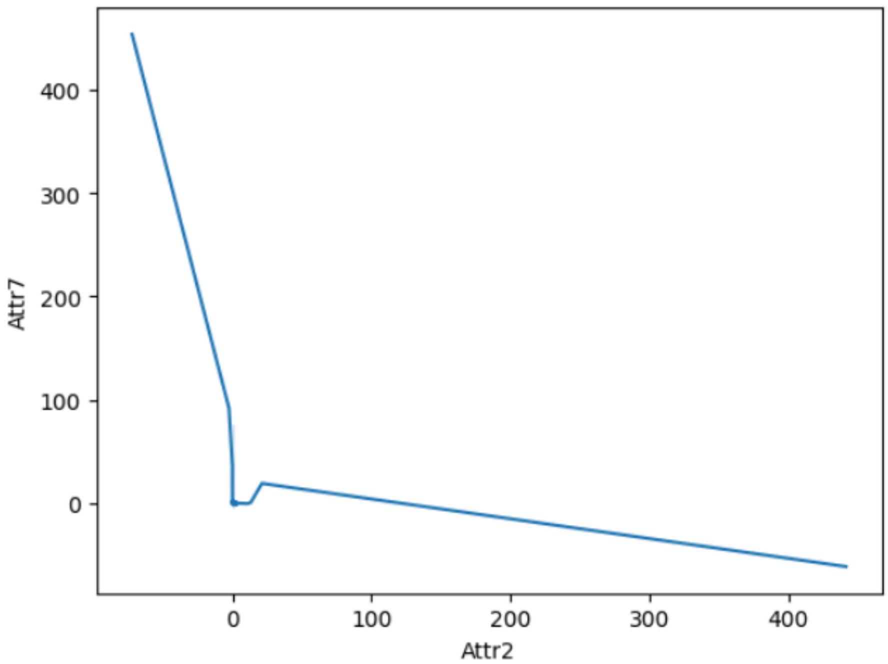


[32]:
```
sbn.scatterplot(x='Attr3',y='Attr9',data=df)
plt.show()
```



```
[29]: sbn.lineplot(x=df['Attr2'],y=df['Attr7'])
```

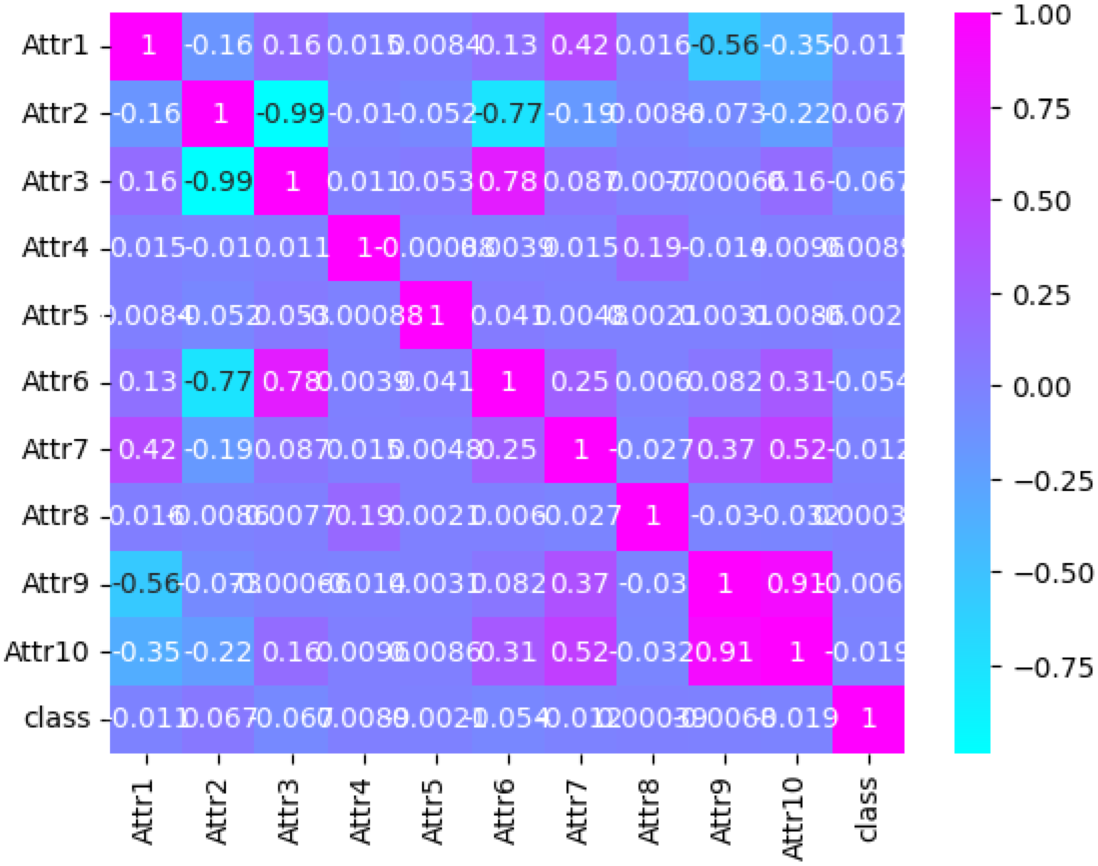
```
[29]: <Axes: xlabel='Attr2', ylabel='Attr7'>
```



3. Multivariate Analysis

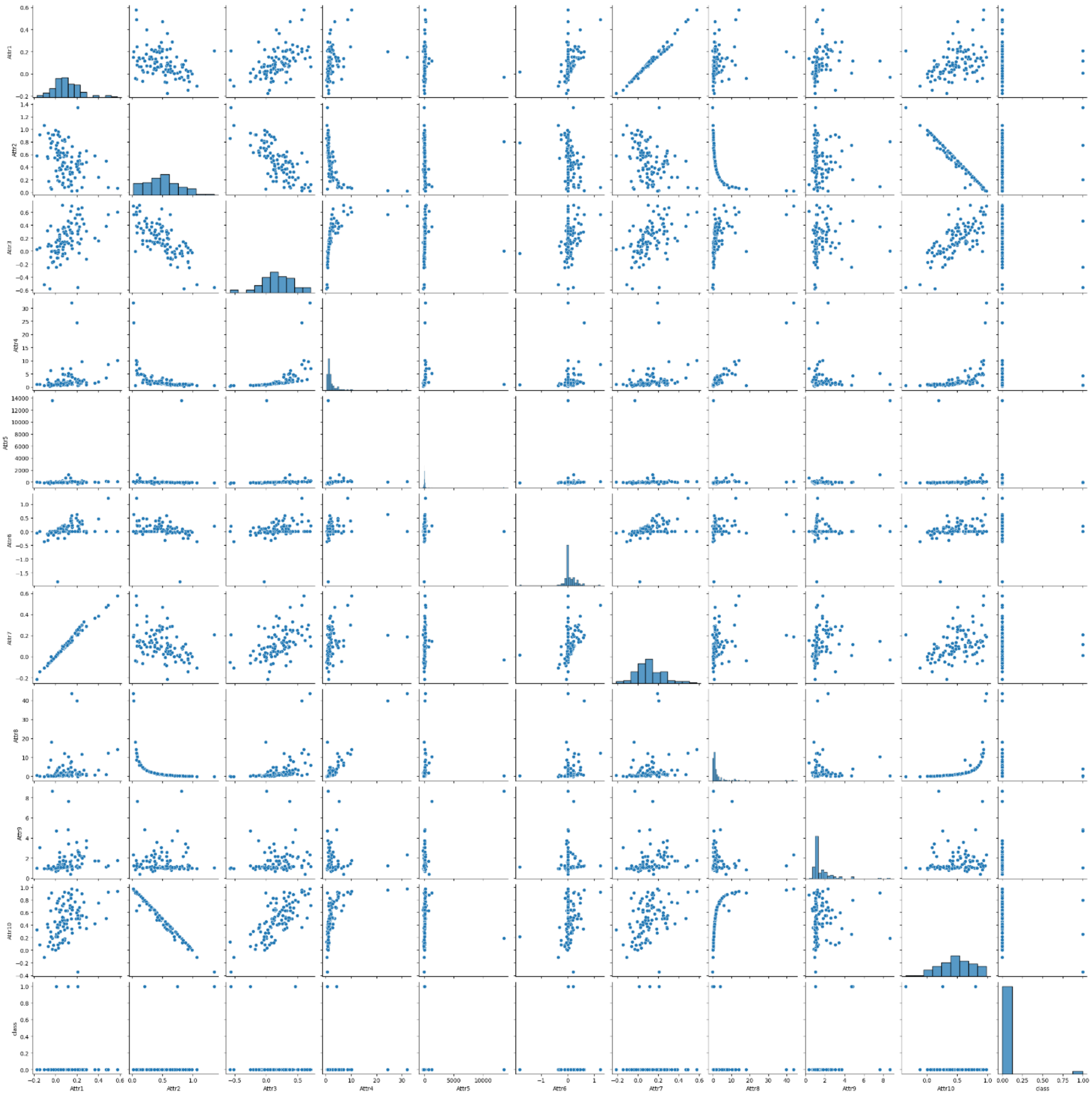
```
[33]: sbn.heatmap(df.corr(),cmap="cool",annot=True)
```

```
[33]: <Axes: >
```



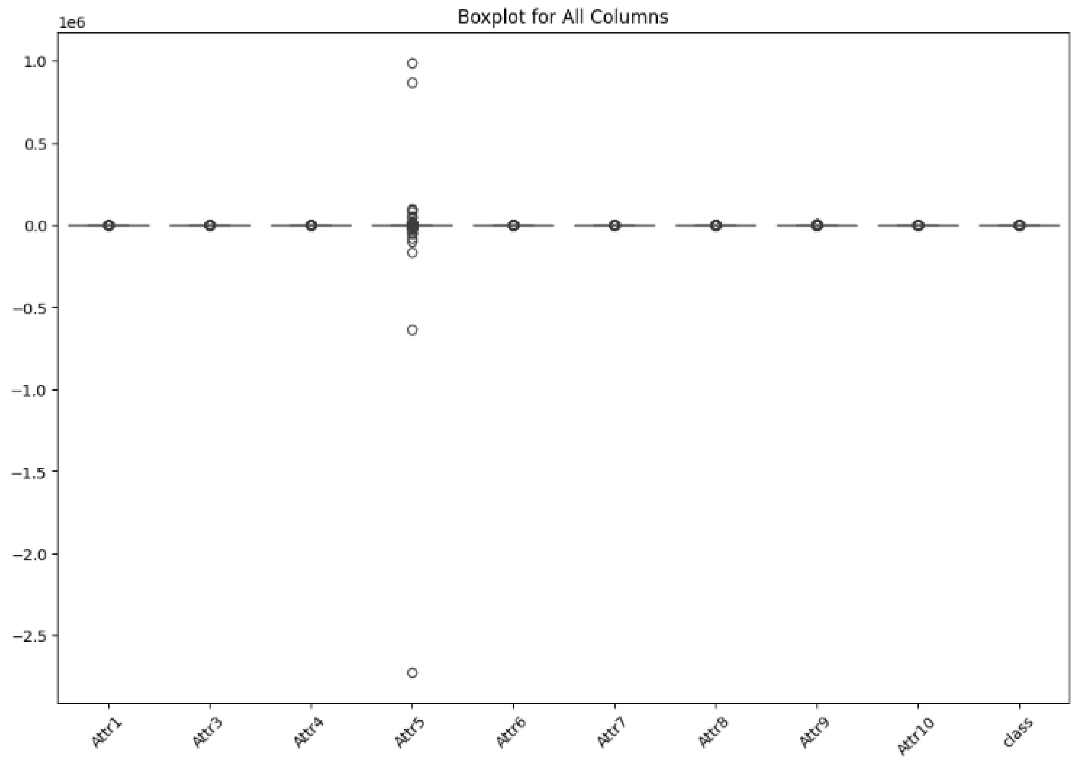
```
]: sbn.pairplot(df.sample(100))
```

```
]: <seaborn.axisgrid.PairGrid at 0x202697e27d0>
```

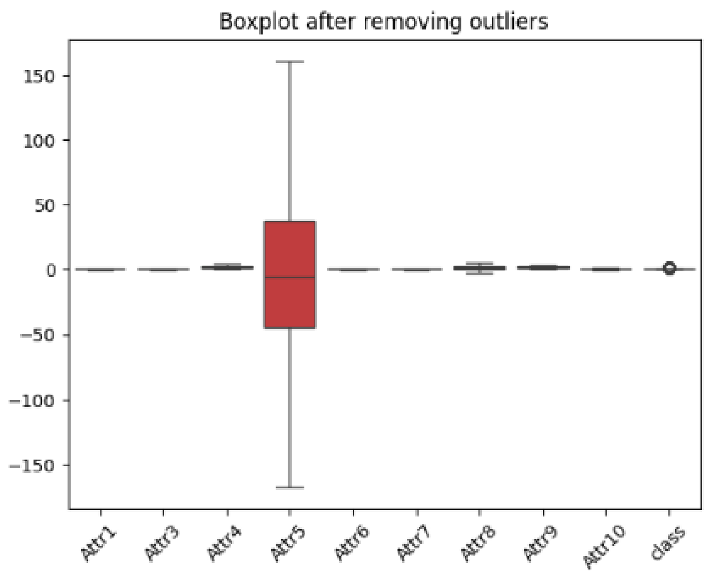


4. Boxplot and outlier removal

```
[38]: # Making a boxplot to see if there are any outliers
plt.figure(figsize=(12, 8))
sbn.boxplot(data=df)
plt.xticks(rotation=45)
plt.title('Boxplot for All Columns')
plt.show()
```




```
[39]: cols = ["Attr1","Attr3","Attr4","Attr5","Attr6","Attr7","Attr8","Attr9","Attr10"]
for i in cols:
  q1 = df[i].quantile(0.25)
  q3 = df[i].quantile(0.75)
  iqr = q3 - q1
  uL = q3 + 1.5*iqr
  lL = q1 - 1.5*iqr
  df[i] = np.where(df[i]>uL,uL,np.where(df[i]<lL,lL,df[i]))
sbn.boxplot(data=df)
plt.xticks(rotation=45)
plt.title('Boxplot after removing outliers')
plt.show()
```

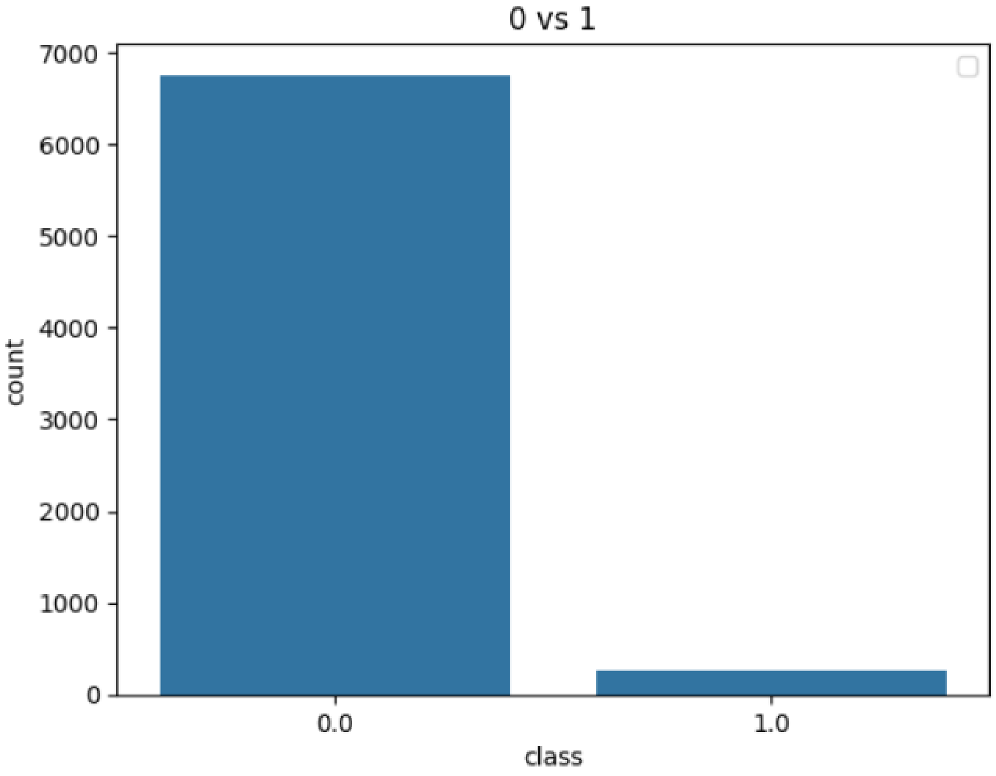


Activity 3.3 Class Analysis:

In the bar graph we can see that most of the people are Female, so we need to balance the dataset.

```
[23]: sbn.barplot(x =df["class"].value_counts().index,y =df["class"].value_counts())
      plt.title('0 vs 1')
      plt.legend()
      plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an unders
```



Milestone 4: Model Building

Activity 1 Dealing with Imbalanced data

```
[74]: from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = df.drop("class", axis=1)
y = df["class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Shape before SMOTE - X_train:", X_train.shape, "y_train:", y_train.shape)
print("Target distribution before SMOTE:\n", y_train.value_counts())
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print("\nShape after SMOTE - X_train_resampled:", X_train_resampled.shape, "y_train_resampled:", y_train_resampled.shape)
print("Target distribution after SMOTE:\n", y_train_resampled.value_counts())


#scale
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)
```

```
Shape before SMOTE - X_train: (5609, 9) y_train: (5609,)
Target distribution before SMOTE:
 class
0.0    5404
1.0     205
Name: count, dtype: int64

Shape after SMOTE - X_train_resampled: (10808, 9) y_train_resampled: (10808,)
Target distribution after SMOTE:
 class
0.0    5404
1.0    5404
Name: count, dtype: int64
```

Activity 2 Train-test split

Now let's split the Dataset into train and test sets. First split the dataset into X and y and then split the data set.

Here X and y variables are created. On X variable, data is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing X, y, test_size, random_state.

In the current project we have below columns as X variable and y variables (Target variable): "Class"

[75]:
```
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = df.drop("class", axis=1)
y = df["class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X)
print(y)
```

```
         Attr1     Attr3     Attr4       Attr5     Attr6     Attr7    Attr8  \
0     0.200550  0.396410  2.047200   32.351000  0.367437  0.249760  1.33050
1     0.209120  0.472250  1.944700   14.786000  0.000000  0.258340  0.99601
2     0.248660  0.267130  1.554800   -1.152300  0.000000  0.309060  0.43695
3     0.081483  0.458790  2.492800   51.952000  0.149880  0.092704  1.86610
4     0.187320  0.229600  1.406300   -7.312800  0.187320  0.187320  0.63070
...        ...       ...       ...         ...       ...       ...      ...
7007  0.038665  0.488840  4.537275  161.047625  0.038665  0.045892  4.97102
7008  0.001091  0.003463  1.008600  -44.467000  0.086248  0.001091  0.17429
7009 -0.091442 -0.047216  0.925680   -7.295200  0.000000 -0.090374  0.41744
7010  0.138090 -0.476192  0.291280  -88.382000 -0.220463  0.138090 -0.70021
7011  0.098271  0.000426  1.000500  -43.191000  0.000000  0.128380  0.20019


         Attr9    Attr10
0     1.138900  0.504940
1     1.699600  0.497880
2     1.309000  0.304080
3     1.057100  0.573530
4     1.155900  0.386770
...        ...       ...
7007  1.076500  0.795600
7008  1.029700  0.148420
7009  3.776512  0.294500
7010  3.776512 -0.260321
7011  2.514400  0.166820

[7012 rows x 9 columns]
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
       ...
7007    1.0
7008    1.0
7009    1.0
7010    1.0
7011    1.0
Name: class, Length: 7012, dtype: float64
```

[76]:
```
print("Length of X_train:", len(X_train))
print("Length of X_test:", len(X_test))
print("Length of y_train:", len(y_train))
print("Length of y_test:", len(y_test))
```

```
Length of X_train: 5609
Length of X_test: 1403
Length of y_train: 5609
Length of y_test: 1403
```

Activity 4: Training and testing the models using multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best models are saved based on their performance.

Activity 4.1 Support Vector Classifier:

A function named "svm" is created and train and test data are passed as the parameters. Inside the function, "SVC" algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model accuracy is calculated.

```
[57]: from sklearn.svm import SVC
      svm = SVC(kernel='rbf', random_state=0)
      svm.fit(X_train_scaled, y_train_resampled)
```

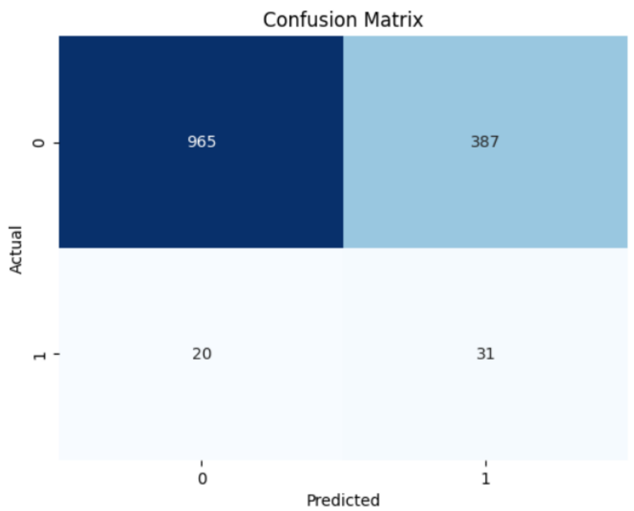
```
[57]:  ▾ SVC
      SVC(random_state=0)
```

```
[58]: y_pred4_train = svm.predict(X_train_scaled)
      y_pred4_test = svm.predict(X_test_scaled)
```

```
[59]: print('Training Accuracy for SVM = ', accuracy_score(y_train_resampled,y_pred4_train))

      Training Accuracy for SVM =  0.8035714285714286
```

```
[60]: print('Testing Accuracy for SVM = ', accuracy_score(y_test,y_pred4_test))

      Testing Accuracy for SVM =  0.7099073414112615
```

```
[63]: print("===================SVM==================")
      print(classification_report(y_test, y_pred4_test))
      from sklearn.metrics import confusion_matrix
      import matplotlib.pyplot as plt


      cm = confusion_matrix(y_test, y_pred4_test)

      # Create a heatmap using Seaborn
      sbn.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
      plt.xlabel('Predicted')
      plt.ylabel('Actual')
      plt.title('Confusion Matrix')
      plt.show()

===================SVM==================
              precision    recall  f1-score   support

         0.0       0.98      0.71      0.83      1352
         1.0       0.07      0.61      0.13        51

    accuracy                           0.71      1403
   macro avg       0.53      0.66      0.48      1403
weighted avg       0.95      0.71      0.80      1403
```



Activity 4.2: Decision Tree Classifier:

A function named "dt" is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, overfitting and accuracy is calculated.

```
[62]: from sklearn.tree import DecisionTreeClassifier
      dt = DecisionTreeClassifier()
      dt.fit(X_train_scaled, y_train_resampled)
```

```
[62]: ▾ DecisionTreeClassifier
      DecisionTreeClassifier()
```

```
[63]: y_pred5_train = dt.predict(X_train_scaled)
      y_pred5_test = dt.predict(X_test_scaled)
```

```
[64]: print('Training Accuracy for DecisionTree = ', accuracy_score(y_train_resampled,y_pred5_train))
```

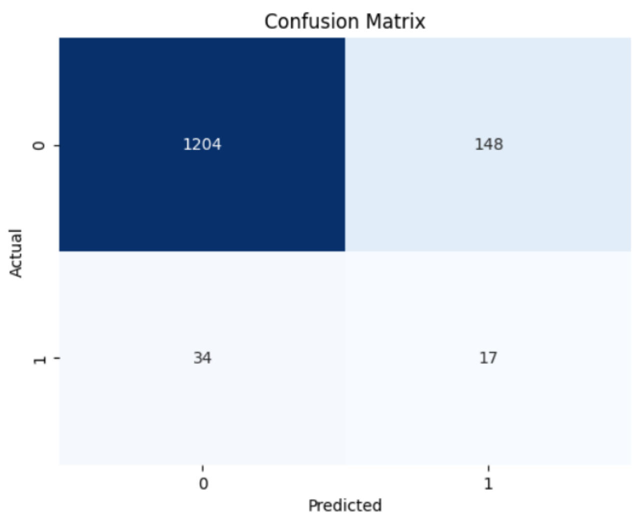
```
Training Accuracy for DecisionTree =  0.9999074759437454
```

```
[65]: print('Testing Accuracy for DecisionTree = ', accuracy_score(y_test,y_pred5_test))
```

```
Testing Accuracy for DecisionTree =  0.8752672843905915
```

```
[68]: print("===================Decision Trees===================")
      print(classification_report(y_test, y_pred5_test))
      from sklearn.metrics import confusion_matrix
      import matplotlib.pyplot as plt


      cm = confusion_matrix(y_test, y_pred5_test)

      # Create a heatmap using Seaborn
      sbn.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
      plt.xlabel('Predicted')
      plt.ylabel('Actual')
      plt.title('Confusion Matrix')
      plt.show()
```

```
===================Decision Trees====================
              precision    recall  f1-score   support

         0.0       0.97      0.89      0.93      1352
         1.0       0.10      0.33      0.16        51

    accuracy                           0.87      1403
   macro avg       0.54      0.61      0.54      1403
weighted avg       0.94      0.87      0.90      1403
```



Activity 4.3 Random Forest Classifier:

A function named "rfc" is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialised and training data is passed tothe model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model accuracy is calculated.

[43]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
```

[44]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(criterion='entropy')
rfc.fit(X_train_scaled, y_train_resampled)
y_pred1_train = rfc.predict(X_train_scaled)
y_pred1_test = rfc.predict(X_test_scaled)
```

[45]:
```
print('Training Accuracy for RandomForest = ', accuracy_score(y_train_resampled,y_pred1_train))
```

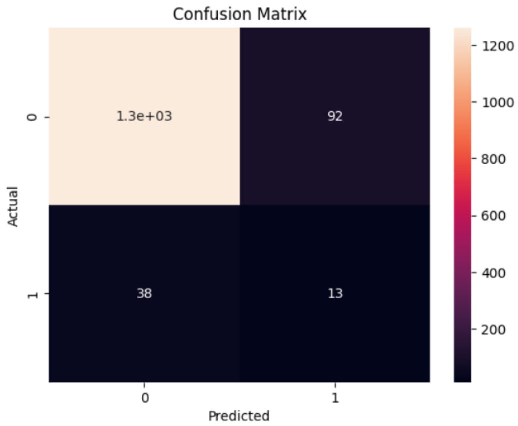
```
Training Accuracy for RandomForest =  0.9999074759437454
```

[46]:
```
print('Testing Accuracy for RandomForest = ', accuracy_score(y_test,y_pred1_test))
```

```
Testing Accuracy for RandomForest =  0.9073414112615823
```

[49]:
```
print("===================Random Forest Classifier===================")
print(classification_report(y_test, y_pred1_test))
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt


cm=confusion_matrix(y_test, y_pred1_test)
# Create a heatmap using Seaborn
sbn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
===================Random Forest Classifier===================
              precision    recall  f1-score   support

         0.0       0.97      0.93      0.95      1352
         1.0       0.12      0.25      0.17        51

    accuracy                           0.91      1403
   macro avg       0.55      0.59      0.56      1403
weighted avg       0.94      0.91      0.92      1403
```



Activity 4.4 K N N :

A function named "knn" is created, taking training and testing data as parameters. Inside the function, a K-Nearest Neighbors classifier is initialized, and the training data is used to train the model using the .fit() function. The test data is then predicted with the .predict() function, and the predictions are stored in a new variable. To evaluate the model, the accuracy is calculated using the accuracy_score metric.

```
[48]: from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train_resampled)
y_pred2_train = knn.predict(X_train_scaled)
y_pred2_test = knn.predict(X_test_scaled)
```

```
[49]: print('Training Accuracy for KNN = ', accuracy_score(y_train_resampled,y_pred2_train))
```

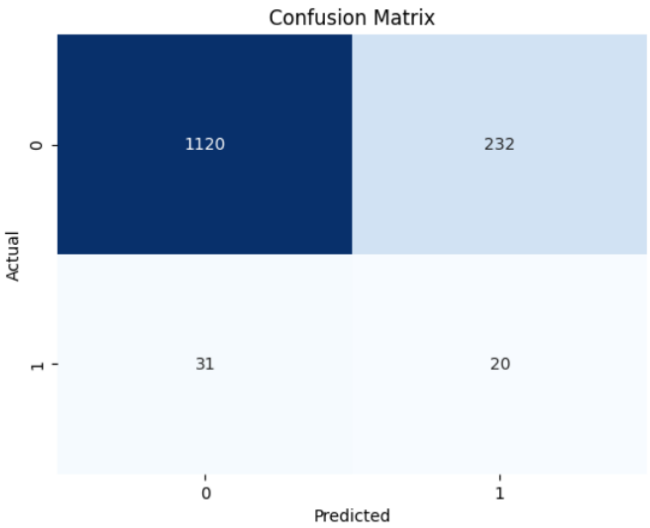
```
Training Accuracy for KNN =  0.9394892672094745
```

```
[50]: print('Testing Accuracy for KNN = ', accuracy_score(y_test,y_pred2_test))
```

```
Testing Accuracy for KNN =  0.812544547398432
```

```
[53]: print("===================KNN===================")
print(classification_report(y_test, y_pred2_test))
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt


cm = confusion_matrix(y_test, y_pred2_test)

# Create a heatmap using Seaborn
sbn.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
===================KNN===================
              precision    recall  f1-score   support

         0.0       0.97      0.83      0.89      1352
         1.0       0.08      0.39      0.13        51

    accuracy                           0.81      1403
   macro avg       0.53      0.61      0.51      1403
weighted avg       0.94      0.81      0.87      1403
```



Activity 4.5 Logistic Regression:

A function named "logReg" is defined to handle logistic regression modeling. Similar to the previous function, it takes training and testing data as parameters. Inside the function, a Logistic Regression classifier is initialized, and the model is trained using the .fit() function with the training data. Subsequently, the test data is used for prediction with the .predict() function, and the predictions are stored in a new variable.

```
[52]: from sklearn.linear_model import LogisticRegression
```

```
[53]: logReg=LogisticRegression(max_iter=500)
      logReg.fit(X_train_scaled, y_train_resampled)
      y_pred3_train = logReg.predict(X_train_scaled)
      y_pred3_test = logReg.predict(X_test_scaled)
```

```
[54]: print('Training Accuracy for Logistic Reg = ', accuracy_score(y_train_resampled,y_pred3_train))
```

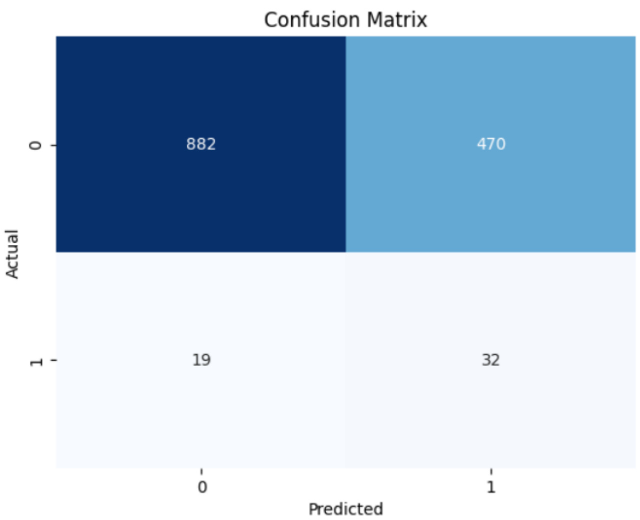
```
Training Accuracy for Logistic Reg =  0.6871761658031088
```

```
[55]: print('Testing Accuracy for Logistic Reg = ', accuracy_score(y_test,y_pred3_test))
```

```
Testing Accuracy for Logistic Reg =  0.6514611546685674
```

```
[58]: print("===================Logistic Regression===================")
      print(classification_report(y_test, y_pred3_test))
      from sklearn.metrics import confusion_matrix
      import matplotlib.pyplot as plt


      cm = confusion_matrix(y_test, y_pred3_test)

      # Create a heatmap using Seaborn
      sbn.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
      plt.xlabel('Predicted')
      plt.ylabel('Actual')
      plt.title('Confusion Matrix')
      plt.show()
```

```
===================Logistic Regression===================
              precision    recall  f1-score   support

         0.0       0.98      0.65      0.78      1352
         1.0       0.06      0.63      0.12        51

    accuracy                           0.65      1403
   macro avg       0.52      0.64      0.45      1403
weighted avg       0.95      0.65      0.76      1403
```



Milestone 5 : Performance Testing

Activity 1: Comparing all the Models.

For comparing the above five models, the accuracy_df function is used.

Below is the accuracy comparison of all the models and we can clearly see that accuracy for Logistics Regression, Decision Tree and Random Forest is 95 percent so we can take any of this model for our classification purpose.

```
[67]: acc_DT = accuracy_score(y_test, y_pred5_test)
      acc_RF = accuracy_score(y_test, y_pred1_test)
      acc_KNN = accuracy_score(y_test, y_pred2_test)
      acc_SVM = accuracy_score(y_test, y_pred4_test)
      acc_LR = accuracy_score(y_test, y_pred3_test)
```

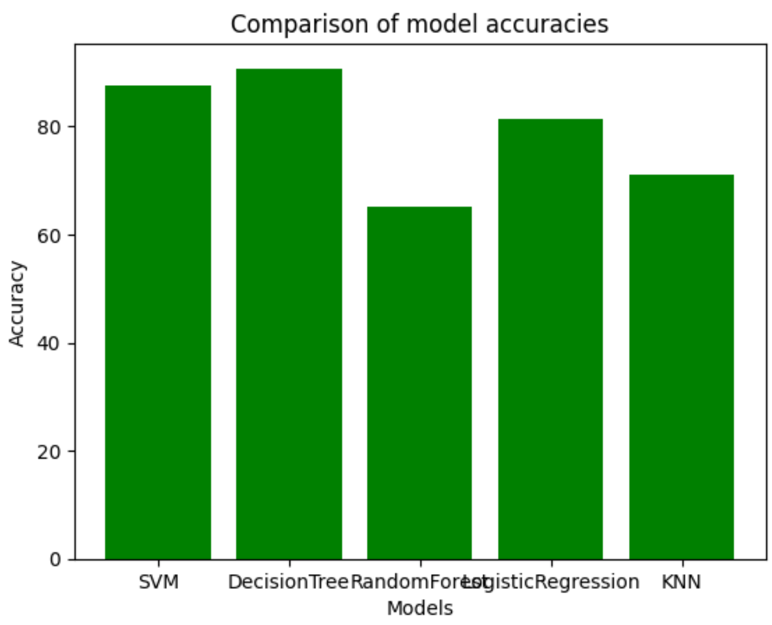
```
[68]: accuracy_df = pd.DataFrame({
          'Model': ['DecisionTree', 'RandomForest', 'LogisticRegression', 'KNN', 'SVM'],
          'Accuracy': [acc_DT*100, acc_RF*100, acc_LR*100, acc_KNN*100, acc_SVM*100]
      })
      print(accuracy_df)
```

```
                Model   Accuracy
0        DecisionTree  87.526728
1        RandomForest  90.734141
2  LogisticRegression  65.146115
3                 KNN  81.254455
4                 SVM  70.990734
```

Activity 2: Graphical representation of the model comparison.

```
[69]: models = ['SVM', 'DecisionTree', 'RandomForest', 'LogisticRegression', 'KNN']
      accuracies = [acc_DT*100, acc_RF*100, acc_LR*100, acc_KNN*100, acc_SVM*100]
      plt.bar(models, accuracies, color='green')
      #add title and axis labels
      plt.title('Comparison of model accuracies')
      plt.xlabel('Models')
      plt.ylabel('Accuracy')
```

```
[69]: Text(0, 0.5, 'Accuracy')
```



Milestone 6: Model Deployment

Activity 1: Save and load the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
•[70]: #saving models with pickle
with open('random_forest_model.pkl', 'wb') as file:
    pickle.dump(rfc, file)
import joblib
joblib.dump(scaler, 'scaler_model.joblib')
```

We save the model using the pickle library into a file named random_forest_model.pkl

Activity 2: Test the model:

Let's test the model first in python notebook itself.

As we have 10 features in this model, let's check the output by giving all the inputs.

```
[ ]: l= [-0.091442, -0.047216, 0.92568, -7.2952, 0, -0.090374, 0.41744, 9.1345, 0.2945]

[81]: model=pickle.load(open("random_forest_model.pkl",'rb'))
print(model.predict([l]))

[0.]
```

We can see above that out model has predicted "0", that means model has classified this as Bankruptcy Detection of Company.

Hence, we can conclude that, out model is giving the accurate results.

Activity 3: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks:

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 3.1: Building HTML pages:

For this project HTML files are crated and saved in the templates folder.

- indexDsn.html

```
templates
  indexDsn.html
```

Activity 3.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

```python
from flask import Flask, render_template, request, jsonify
import joblib
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

app = Flask(__name__)

model1 = joblib.load('Bankruptcy/knn.pkl')
model2 = joblib.load('Bankruptcy/svm.pkl')
model3 = joblib.load('Bankruptcy/decision_tree.pkl')
model4 = joblib.load('Bankruptcy/logistic_reg.pkl')
model5 = joblib.load('Bankruptcy/random_forest_model.pkl')


@app.route('/')
def index():
    return render_template('indexDsn.html')


@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.get_json(force=True)
        selected_model = data['selectedModel']
        attribute_values =data['attributeValues']
        attribute_values = np.array(attribute_values).reshape(1, -1)
        scaler = joblib.load("Bankruptcy\scaler_model.joblib")
        attribute_values = scaler.transform(attribute_values)
```

bank.py | app.py | BankruptcyDetection.ipynb | 1year.csv | index.html | Project.ipynb

Bankruptcy-project > Bankruptcy > app.py > ...

```python
def predict():
    try:
        data = request.get_json(force=True)
        selected_model = data['selectedModel']
        attribute_values =data['attributeValues']
        attribute_values = np.array(attribute_values).reshape(1, -1)
        scaler = joblib.load("Bankruptcy\scaler_model.joblib")
        attribute_values = scaler.transform(attribute_values)
        app.logger.info(f"Received prediction request for model: {selected_model}")

        if selected_model == 'model1':
            used_model = "Model 1 (KNN)"
            prediction=(model1.predict(attribute_values))
            prediction_label = "will be" if prediction[0] == 1 else "will not be"
            app.logger.info(f"Prediction using {used_model}: {prediction}")

        elif selected_model == 'model2':
            used_model = "Model 2 (SVM)"
            prediction=(model2.predict(attribute_values))
            prediction_label = "will be" if prediction[0] == 1 else "will not be"
            app.logger.info(f"Prediction using {used_model}: {prediction}")

        elif selected_model == 'model3':
            used_model = "Model 3 (Decision Tree)"
            prediction=(model3.predict(attribute_values))
            prediction_label = "will be" if prediction[0] == 1 else "will not be"
            app.logger.info(f"Prediction using {used_model}: {prediction}")

```

bank.py | app.py | BankruptcyDetection.ipynb | 1year.csv | index.html | Project.ipynb

Bankruptcy-project > Bankruptcy > app.py > ...

```python
            prediction=(model3.predict(attribute_values))
            prediction_label = "will be" if prediction[0] == 1 else "will not be"
            app.logger.info(f"Prediction using {used_model}: {prediction}")

        elif selected_model == 'model4':
            used_model = "Model 4 (Logistic Regression)"
            prediction=(model4.predict(attribute_values))
            prediction_label = "will be" if prediction[0] == 1 else "will not be"
            app.logger.info(f"Prediction using {used_model}: {prediction}")

        elif selected_model == 'model5':
            used_model = "Model 5 (Random Forest)"
            prediction=(model5.predict(attribute_values))
            prediction_label = "will be" if prediction[0] == 1 else "will not be"
            app.logger.info(f"Prediction using {used_model}: {prediction}")

        else:
            return jsonify(error="Invalid model selected")

        return jsonify(prediction=int(prediction[0]),usedModel=used_model,predictionLabel=prediction_label)
    except Exception as e:
        app.logger.error(f"Error in prediction: {str(e)}")
        return jsonify(error=str(e))

if __name__ == '__main__':
    app.run(debug=True)

```

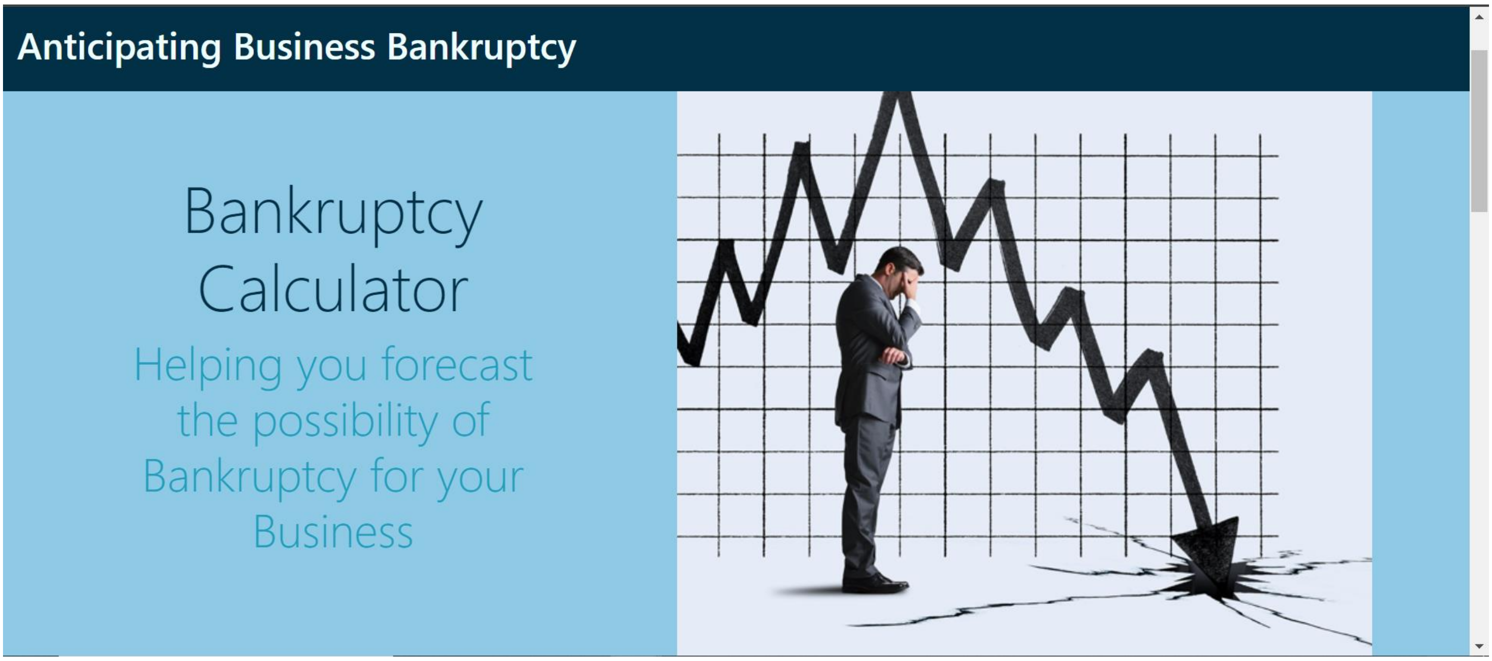
Ln 10, Col 43 Spaces: 4 UTF-8 CRLF Python 3.11.2 64-bit Go Live Prettier

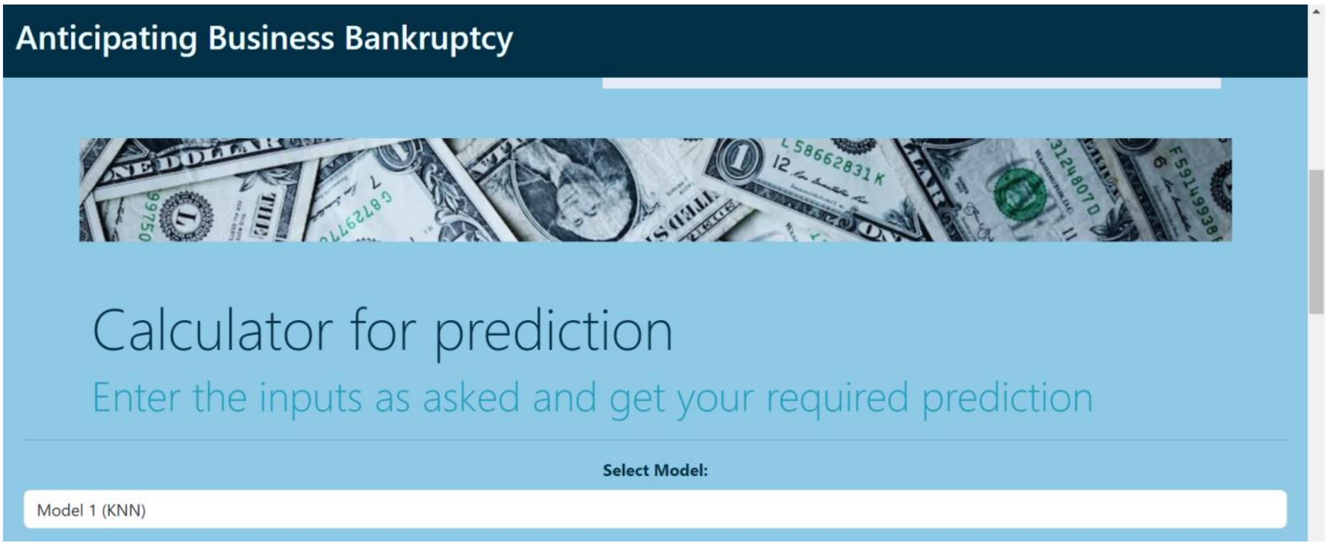
Activity 4: GUI:

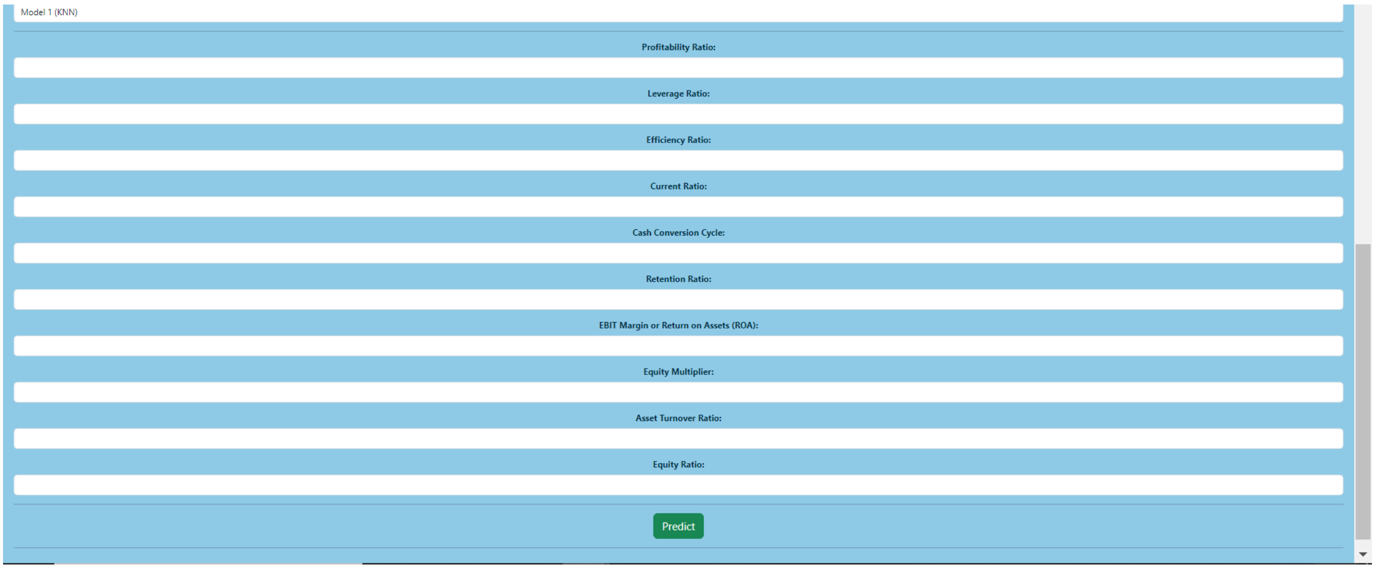
The GUI (Graphical User Interface) created in this Flask application is designed to predict the bankruptcy of the company bases on below features:

- Attribute 1: Profitability Ratio
- Attribute 2: Leverage Ratio
- Attribute 3: Efficiency Ratio
- Attribute 4: Current Ratio
- Attribute 5: Cash Conversion Cycle
- Attribute 6: Retention Ratio
- Attribute 7: EBIT Margin or Return on Assets (ROA)
- Attribute 8: Equity Multiplier
- Attribute 9: Asset Turnover Ratio
- Attribute 10: Equity Ratio

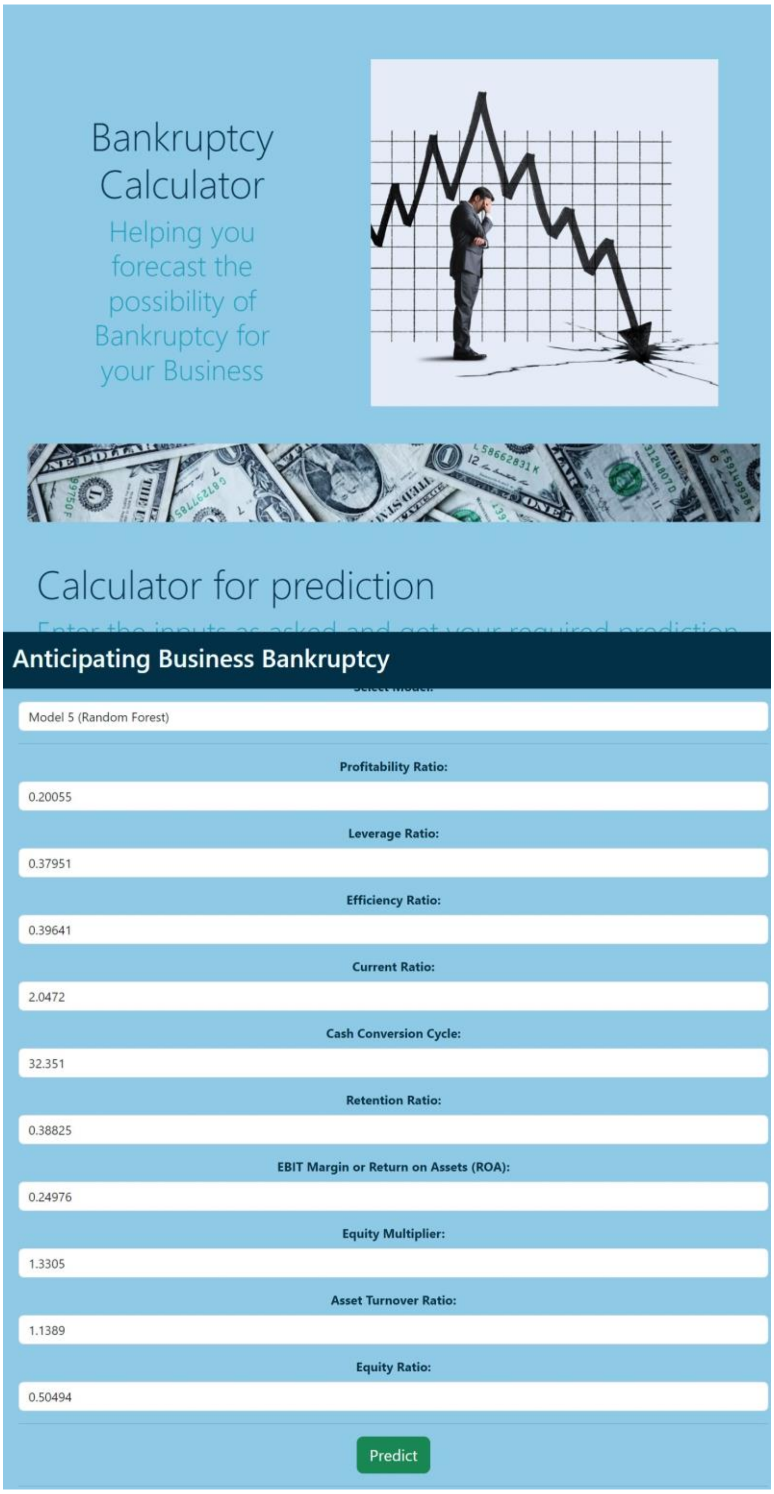
The user can input this feature in a form provided in the home page of the web application. After clicking on the "Predict" button, the application will predict the level of freedom of that country based on the rules and the random forest model defined in the Python script.




Anticipating Business Bankruptcy
Calculator for prediction
Enter the inputs as asked and get your required prediction
Select Model:
Model 1 (KNN)


Model 1 (KNN)
Profitability Ratio:
Leverage Ratio:
Efficiency Ratio:
Current Ratio:
Cash Conversion Cycle:
Retention Ratio:
EBIT Margin or Return on Assets (ROA):
Equity Multiplier:
Asset Turnover Ratio:
Equity Ratio:
Predict

Choosing the model and entering the values for each attribute:



Output in the flask terminal:

```
[2023-11-23 13:51:23,113] INFO in app: Received prediction request for model: model5
[2023-11-23 13:51:23,117] INFO in app: Prediction using Model 5 (Random Forest): [0.]
127.0.0.1 - - [23/Nov/2023 13:51:23] "POST /predict HTTP/1.1" 200 -
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 723-466-661
```

Class values for binary classification:

- 0 - company that **did not bankrupt** in the forecasting period
- 1 - **bankrupted** company

Output in the html page:

Predict

The Model 5 (Random Forest) model predicts that the company will not be bankrupted in the given forecasting period

Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1: Record explanation Video for project end to end solution
https://youtu.be/OMnMjSiHggg
Activity 2: Project Documentation-Step by step project development procedure
Project documentation has been uploaded in the next phase, can be found in the git repository.