# ANTICIPATING BUSINESS BANKRUPTCY

The field of bankruptcy prediction, a prominent topic in economics for almost a century, focuses on creating predictive models that use various econometric indicators to forecast a company's financial health. These models assist in evaluating a company's financial status and its long-term prospects in the market. The research discussed here involves analysing businesses in Poland from 2000 to 2012, with a specific emphasis on those that went bankrupt and those that remained operational in 2007. The dataset includes financial ratios from the initial year, with class labels indicating bankruptcy status. These labels are used to categorize companies based on their vulnerability to bankruptcy.

The bankruptcy prediction field has evolved to keep pace with the changing global economic landscape, employing increasingly sophisticated tools and methodologies. Researchers continuously improve their approaches to provide businesses and stakeholders with critical insights into their financial health and long-term viability in the market. This area of study remains vital for decision-makers seeking to anticipate and manage financial risks in the corporate world.

## Aim:

Develop a predictive model for anticipating business bankruptcy using econometric measures and historic data.

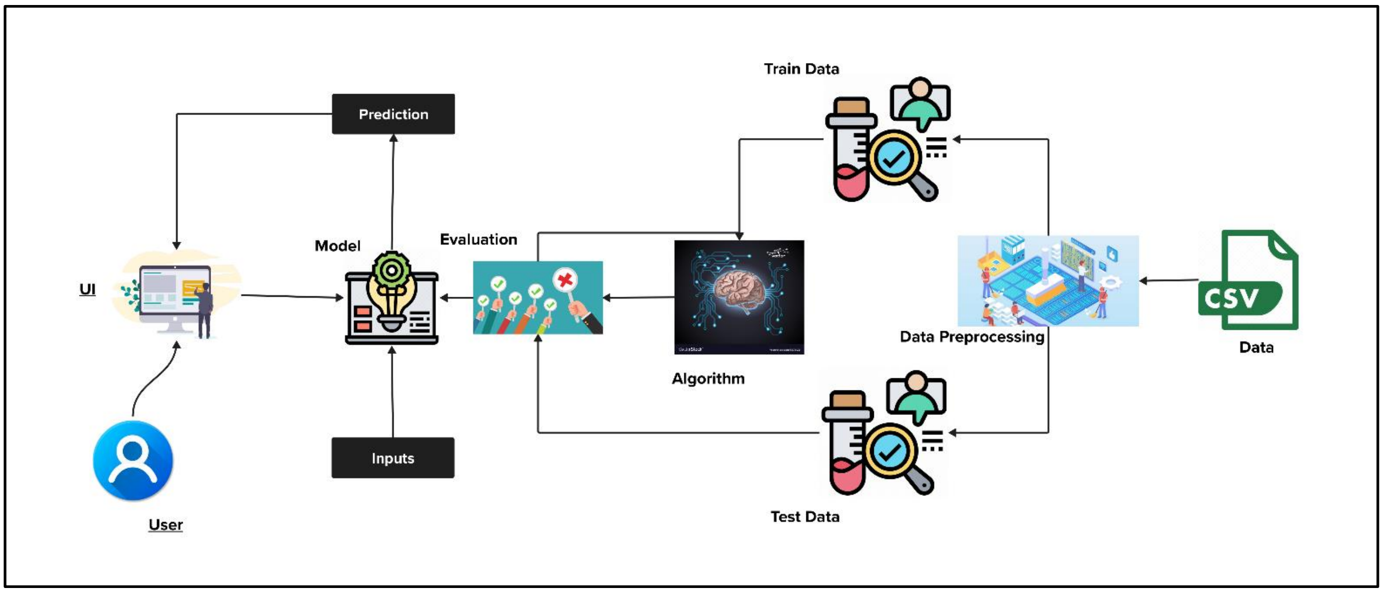
# INDEX

Back to Index

# TECHNICAL ARCHITECTURE

In the process of building a machine learning model, several key steps are followed. It begins with the raw CSV data, which serves as the foundational dataset. Data preprocessing is a critical phase involving data cleaning to address missing values, outliers, and inconsistencies, as well as data transformation, which encompasses feature engineering, encoding categorical variables, scaling, and normalization. Feature selection is performed to choose relevant attributes. The dataset is then split into training data (80% of the dataset) for model training and test data (20%) for model evaluation.

Algorithm selection depends on the task type and data characteristics, with the chosen algorithm used to train the model. In our project, we deal with a classification problem and hence, will be using Classification algorithms in ML such as, Decision Trees, Random Forest, Extreme Gradient Boosting, etc. Model performance is assessed using metrics like accuracy scores and classification reports. A decision point is reached after evaluation: if the model's performance is unsatisfactory, it returns to algorithm selection and data preparation; if it meets criteria, the final model is built using the entire dataset. Successful models can be deployed in the real world, with continuous monitoring and adjustments to ensure ongoing performance as new data becomes available. This comprehensive process ensures the creation of effective machine learning models for various tasks.



Back to Index

# PROJECT FLOW

To ensure the successful execution of our project, it is crucial that we comprehensively outline the major flow of activities, tasks, and processes involved. The entire project can be broken down into a sequence of steps, each contributing to the overall goal of developing an interactive and intelligent user interface driven by predictive and analytical models. Here's an expanded overview of the project's major flow and the associated tasks:

1. Project Initiation:
    - Define the project scope and objectives.
    - Form the project team and allocate responsibilities.
    - Establish a clear timeline and milestones for project execution.
2. Problem Definition:
    - Begin with a thorough understanding of the problem statement.
    - Specify the business problem to be addressed through the UI and integrated model.
    - Identify and document the specific business requirements that the UI should meet.
    - Highlight the potential social and business impact of solving the problem through the project, addressing its broader implications.
3. Data Collection:
    - Identify sources of relevant data for the project.
    - Develop a data collection strategy and gather the necessary datasets.
    - Ensure proper storage and organization of the collected data for later use in the project.
4. Data Preparation:
    - Preprocess the collected data, which includes:
    - Handling null or missing values, employing techniques like imputation or removal.
    - Extracting relevant features from the raw data, preparing it for analysis.
5. Exploratory Data Analysis:
    - Conduct data visualization and exploratory data analysis to gain insights into the dataset.
    - Visualize data distributions, correlations, and patterns to inform subsequent modelling decisions.
6. Data Segmentation (X and Y Splitting):
    - Split the dataset into input (X) and target (Y) variables, enabling the model to learn from the data.
7. Feature Selection:
    - Identify and select the most relevant features to be used in the model, enhancing its efficiency and effectiveness.
8. Train and Test Data Splitting:
    - Split the dataset into training and testing subsets to assess the model's performance accurately.
9. Data Balancing:
    - Address any class imbalance issues in the dataset to ensure that the model doesn't favour one class over others.

Back to Index

10. Model Building:

- Develop the predictive and analytical model using appropriate algorithms and techniques.
- Train the model on the training data to learn patterns and relationships.

11. Performance Testing:

- Assess the model's performance using various evaluation metrics, such as:
- Accuracy scores to measure overall model correctness.
- Classification reports to provide insights into precision, recall, and F1-score.

12. Model Saving and Export:

- Save the trained model for future use and export it for deployment.

13. Deployment of the Model:

- Develop a user-friendly GUI for interaction with the model.
- Integrate the GUI with a web framework, allowing users to access the UI through web browsers.
- Deploy the project, making it accessible to the intended audience, and ensure it runs smoothly in a production environment.

Addressing each of these project activities and tasks comprehensively, will enable us to create a seamless and intelligent user interface that leverages predictive and analytical models to address the specified business problem effectively. This systematic approach will significantly enhance our chances of a successful project implementation.

Back to Index

## PRIOR KNOWLEDGE

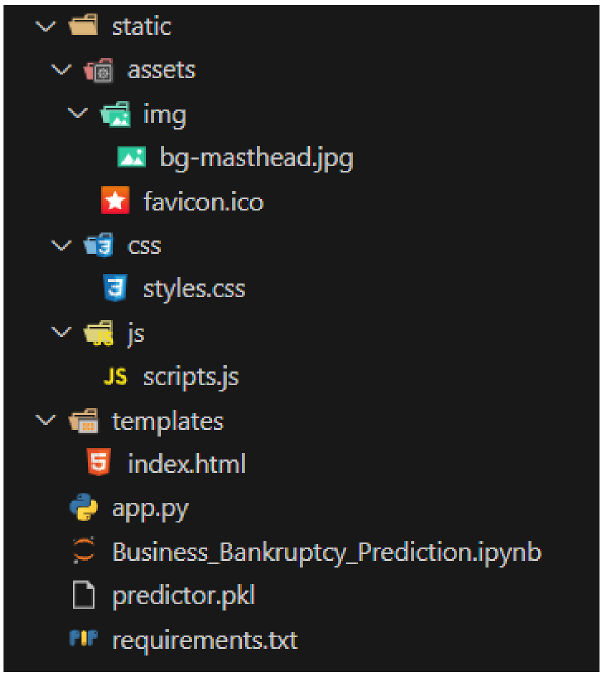
To make sure the bankruptcy prediction project works well, it's really important for the project team to know a lot about five important things. These things are crucial for the project:

1. *Financial Domain Proficiency*:
   - Know a lot about financial numbers, like balance sheets and income statements.
   - This knowledge helps us understand how well a company is doing financially.
   - Be familiar with financial terms like how quickly a company can get cash, how much money it's making, if it can pay its debts, and how efficiently it uses its assets.
2. *Machine Learning Foundations and Classification Algorithms*:
   - It's crucial to know the basics of machine learning tools like decision trees, random forests, and extreme gradient boosting classification algorithms.
   - Knowing the fine details of these tools, like making them work even better and using a bunch of them together, is a big help in making the models work well and give the right answers.
3. *Feature Selection Expertise*:
   - Requires effective selection of relevant features, with a specific focus on Recursive Feature Elimination (RFE).
   - Choosing the right stuff from the data makes it simpler, makes the model work better, and makes training the model faster.
   - Hence, it's very important to be good at finding and keeping the most useful information.
4. *Imbalanced Data Handling Skills*:
   - Dealing with imbalanced data is a will be a crucial step.
   - It's important to be familiar with techniques like Synthetic Minority Over-sampling Technique (SMOTE) to fix this problem.
   - Being unable to handle this issue will result in incorrect predictions.
5. *Data Source Awareness*:
   - Understanding the data source is vital for the project's integrity.
   - In this context, data was collected from Emerging Markets Information Service (EMIS), a comprehensive database of emerging markets worldwide.
   - Being well-informed about the data source provides the foundation for conducting a reliable analysis.

By honing their expertise in these key domains, the project team can effectively pave the way for the development of a robust bankruptcy prediction model. With this multifaceted knowledge, they can navigate the complexities of financial data, machine learning techniques, feature selection, data balancing, and data source intricacies to create a predictive model that offers valuable insights into a company's financial stability.

Back to Index

## PROJECT STRUCTURE



```
static
  assets
    img
      bg-masthead.jpg
    favicon.ico
  css
    styles.css
  js
    scripts.js
templates
  index.html
app.py
Business_Bankruptcy_Prediction.ipynb
predictor.pkl
requirements.txt
```

The project structure is as follows:

```
project
|---- static
|       |---- assets
|       |       |---- img
|       |       |       |---- bg-masthead.jpg
|       |       |---- favicon.ico
|       |---- css
|       |       |---- styles.css
|       |---- js
|       |       |---- scripts.js
|---- templates
|       |---- index.html
|---- app.py
|---- Business_Bankruptcy_Prediction.ipynb
|---- predictor.pkl
|---- requirements.txt
```

At its core, we can find the "static" directory. Nested within this directory is the "assets" subfolder, housing a valuable collection of assets that contribute to the application's visual appeal and functionality. Notably, the "img" subdirectory within "assets" contains "bg-masthead.jpg," a key image resource serving as the background for the web interface. Furthermore, the "favicon.ico" file, located directly within "assets," functions as the browser tab's iconic representation. The "css" directory stores "styles.css," a CSS file crucial for defining the visual aesthetics and styling of the web application. In parallel, the "js" directory houses "scripts.js," a JavaScript file responsible for incorporating client-side scripting features into the web application.

The "templates" directory, as the name suggests, serves as the repository for HTML templates utilized by the Flask web application. Notably, "index.html" is the primary and only HTML template, defining the structure and content of the web application's main page.

At the heart of the project is "app.py," a Python script that leverages the Flask web framework to create, run, and manage the web application. This script connects all the elements, ensuring the smooth execution of the application and serving the model's predictions to users. The "predictor.pkl" file is central to the predictive capabilities of the application, containing a serialized machine learning model ready for use in making predictions related to business bankruptcy.

"Business_Bankruptcy_Prediction.ipynb" is the Jupyter Notebook, that includes code, data analysis, and project documentation.

Finally, "requirements.txt" plays a critical role in specifying the project's dependencies and their version compatibility, guaranteeing that the necessary libraries and packages are properly installed for seamless application functionality.

Back to Index

# DEFINING THE PROBLEM

## Objective

The objective of bankruptcy prediction is to assess both a company's current financial condition and its outlook within the framework of its continued presence and operations in the market over the long term.

## Business Requirements

- *Risk Assessment*: Through the anticipation of bankruptcy likelihood, businesses gain the ability to assess the financial hazards linked to engaging with a specific company. This equips them to make knowledgeable choices regarding extending credit, entering contractual agreements, or venturing into collaborative endeavors.
- *Financial Strategy*: Precise bankruptcy prediction plays a pivotal role in a company's financial strategy. It empowers organizations to foresee possible challenges in cash flow, identify areas in need of enhancement, and proactively institute measures to diminish vulnerabilities.
- *Early Detection Mechanism*: Models for forecasting bankruptcy can act as an early detection mechanism, providing companies with timely alerts about potential financial troubles before they escalate. This enables companies to promptly implement corrective actions, including debt restructuring, contract renegotiation, or the implementation of cost-cutting initiatives.

## Business Impact

- *Credit Risk Management*: The ability to accurately predict bankruptcy helps banks and financial institutions assess creditworthiness, make informed lending decisions, and manage credit risk, reducing the likelihood of loan defaults and financial losses.
- *Investment Decisions*: Investors can use bankruptcy predictions to evaluate the financial health of companies before making investments, which can include stock purchases, bond investments, and decisions related to initial public offerings (IPOs).
- *Supply Chain Management*: Proactive identification of financially distressed suppliers allows companies to secure alternative sources and maintain a smooth and resilient supply chain.
- *Mergers and Acquisitions*: Companies can assess the financial risk associated with potential merger or acquisition targets, aiding in deal negotiations and structuring transactions.
- *Regulatory Compliance*: Businesses in regulated industries can use bankruptcy prediction to demonstrate compliance with financial stability requirements imposed by regulators, ensuring adherence to industry standards and avoiding regulatory penalties.

Back to Index

## Social Impact

- *Job Security*: When businesses can predict financial problems early, they're more likely to take steps to prevent bankruptcy. This helps protect the jobs of employees and ensures a more stable work environment.
- *Customer Trust*: Maintaining financial stability and a low risk of bankruptcy can positively impact a company's reputation and customer trust. Customers feel more confident doing business with companies that are financially secure.
- *Economic Stability*: By reducing the likelihood of businesses going bankrupt, the project contributes to overall economic stability. Fewer bankruptcies mean fewer disruptions in the job market and a healthier economy.
- *Supplier Relationships*: The project helps companies maintain good relationships with suppliers by ensuring they don't run into financial trouble. This, in turn, fosters stronger social and business connections.
- *Community Well-being*: Strong, stable businesses contribute to the well-being of the communities in which they operate. They can support local causes, provide employment, and stimulate economic growth, benefiting the overall quality of life in the community.

Back to Index

# DATA COLLECTION AND PREPARATION

## Collecting the Dataset

The dataset utilized in the project was sourced from Kaggle, a prominent platform for data sharing and analysis. This dataset consisted of five distinct CSV files, each corresponding to a specific year within the time frame from 2007 to 2012. These files collectively provided the foundational data required for the bankruptcy prediction project mentioned earlier.

Dataset files: Link

Each of the five dataset files, spanning from 2007 to 2012, contains 65 columns. The critical column, "class," serves as the target variable for predicting a company's financial status. The remaining 64 columns, labeled "Attr1" through "Attr64," encompass diverse financial metrics and ratios, providing a comprehensive financial overview. This dataset forms the foundation for our bankruptcy prediction project, enabling in-depth analysis and predictive modeling.

## About the Data

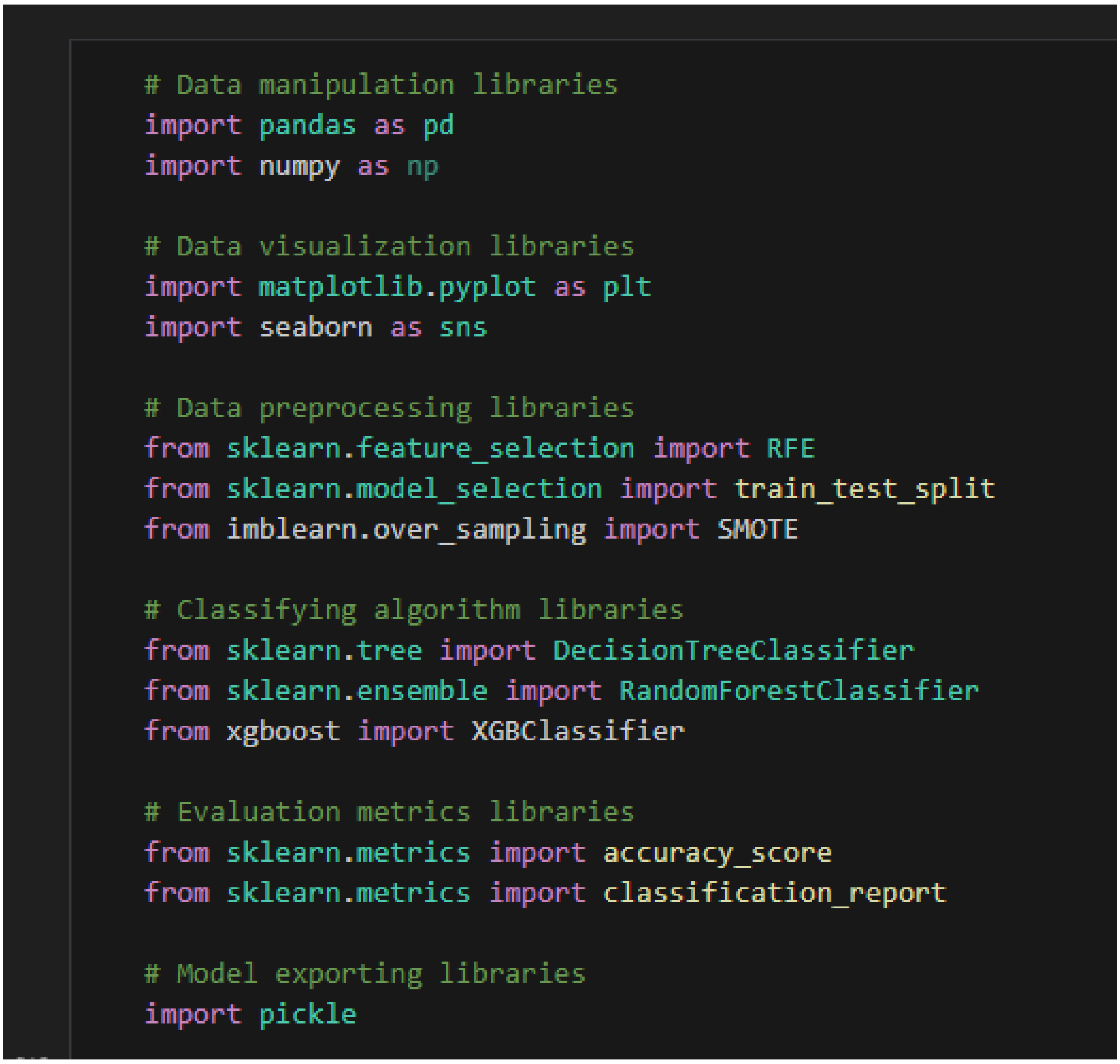
| Features | Description |
|---|---|
| Attr1 | net profit / total assets |
| Attr2 | total liabilities / total assets |
| Attr3 | working capital / total assets |
| Attr4 | current assets / short-term liabilities |
| Attr5 | [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365 |
| Attr6 | retained earnings / total assets |
| Attr7 | EBIT / total assets |
| Attr8 | book value of equity / total liabilities |
| Attr9 | sales / total assets |
| Attr10 | equity / total assets |
| Attr11 | (gross profit + extraordinary items + financial expenses) / total assets |
| Attr12 | gross profit / short-term liabilities |
| Attr13 | (gross profit + depreciation) / sales |
| Attr14 | (gross profit + interest) / total assets |
| Attr15 | (total liabilities * 365) / (gross profit + depreciation) |
| Attr16 | (gross profit + depreciation) / total liabilities |
| Attr17 | total assets / total liabilities |
| Attr18 | gross profit / total assets |
| Attr19 | gross profit / sales |
| Attr20 | (inventory * 365) / sales |
| Attr21 | sales (n) / sales (n-1) |
| Attr22 | profit on operating activities / total assets |
| Attr23 | net profit / sales |
| Attr24 | gross profit (in 3 years) / total assets |
| Attr25 | (equity share capital) / total assets |

| | |
|---|---|
| Attr26 | (net profit + depreciation) / total liabilities |
| Attr27 | profit on operating activities / financial expenses |
| Attr28 | working capital / fixed assets |
| Attr29 | logarithm of total assets |
| Attr30 | (total liabilities cash) / sales |
| Attr31 | (gross profit + interest) / sales |
| Attr32 | (current liabilities * 365) / cost of products sold |
| Attr33 | operating expenses / short-term liabilities |
| Attr34 | operating expenses / total liabilities |
| Attr35 | profit on sales / total assets |
| Attr36 | total sales / total assets |
| Attr37 | (current assets inventories) / long-term liabilities |
| Attr38 | constant capital / total assets |
| Attr39 | profit on sales / sales |
| Attr40 | (current assets inventory receivables) / short-term liabilities |
| Attr41 | total liabilities / ((profit on operating activities + depreciation) * (12/365)) |
| Attr42 | profit on operating activities / sales |
| Attr43 | rotation receivables + inventory turnover in days |
| Attr44 | (receivables * 365) / sales |
| Attr45 | net profit / inventory |
| Attr46 | (current assets inventory) / short-term liabilities |
| Attr47 | (inventory * 365) / cost of products sold |
| Attr48 | EBITDA (profit on operating activities depreciation) / total assets |
| Attr49 | EBITDA (profit on operating activities depreciation) / sales |
| Attr50 | current assets / total liabilities |
| Attr51 | : short-term liabilities / total assets |
| Attr52 | (short-term liabilities * 365) / cost of products sold) |
| Attr53 | equity / fixed assets |
| Attr54 | constant capital / fixed assets |
| Attr55 | working capital |
| Attr56 | (sales cost of products sold) / sales |
| Attr57 | (current assets inventory short-term liabilities) / (sales gross profit depreciation) |
| Attr58 | total costs /total sales |
| Attr59 | long-term liabilities / equity |
| Attr60 | sales / inventory |
| Attr61 | sales / receivables |
| Attr62 | (short-term liabilities *365) / sales |
| Attr63 | sales / short-term liabilities |
| Attr64 | sales / fixed assets |
| Class | did not get bankrupt (0) or got bankrupt (1) |

Back to Index

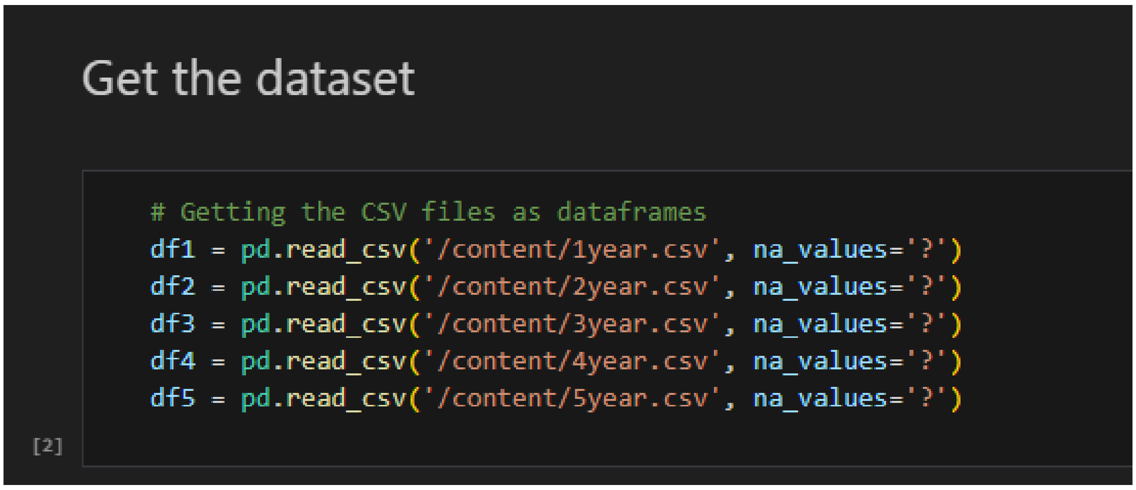
## Data Preparation

*Importing the Libraries*: Various libraries played a fundamental role in the development of our machine learning model for bankruptcy prediction. These libraries provide essential functionalities for data manipulation, visualization, preprocessing, classification algorithms, evaluation metrics, and model exporting.



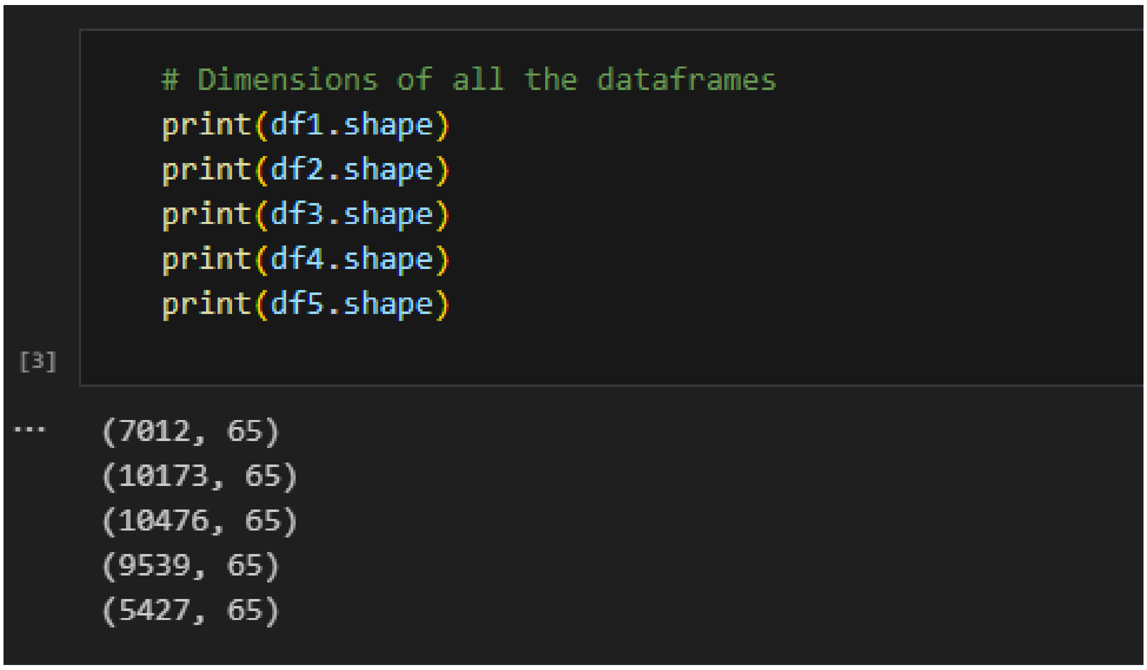
```python
# Data manipulation libraries
import pandas as pd
import numpy as np

# Data visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Data preprocessing libraries
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Classifying algorithm libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# Evaluation metrics libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Model exporting libraries
import pickle
```

*Read the dataset*: To read the dataset, we use pandas' “read_csv()” function. As a parameter, we must specify the directory of the csv file.



Get the dataset

```
# Getting the CSV files as dataframes
df1 = pd.read_csv('/content/1year.csv', na_values='?')
df2 = pd.read_csv('/content/2year.csv', na_values='?')
df3 = pd.read_csv('/content/3year.csv', na_values='?')
df4 = pd.read_csv('/content/4year.csv', na_values='?')
df5 = pd.read_csv('/content/5year.csv', na_values='?')
```
[2]

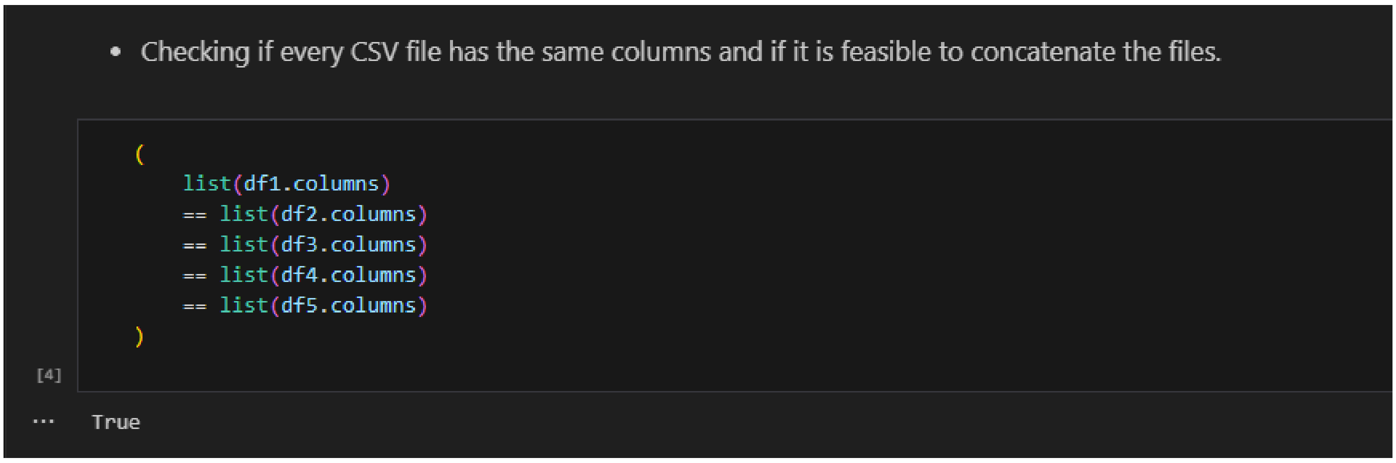


```
# Dimensions of all the dataframes
print(df1.shape)
print(df2.shape)
print(df3.shape)
print(df4.shape)
print(df5.shape)
```
[3]

```
(7012, 65)
(10173, 65)
(10476, 65)
(9539, 65)
(5427, 65)
```

Back to Index

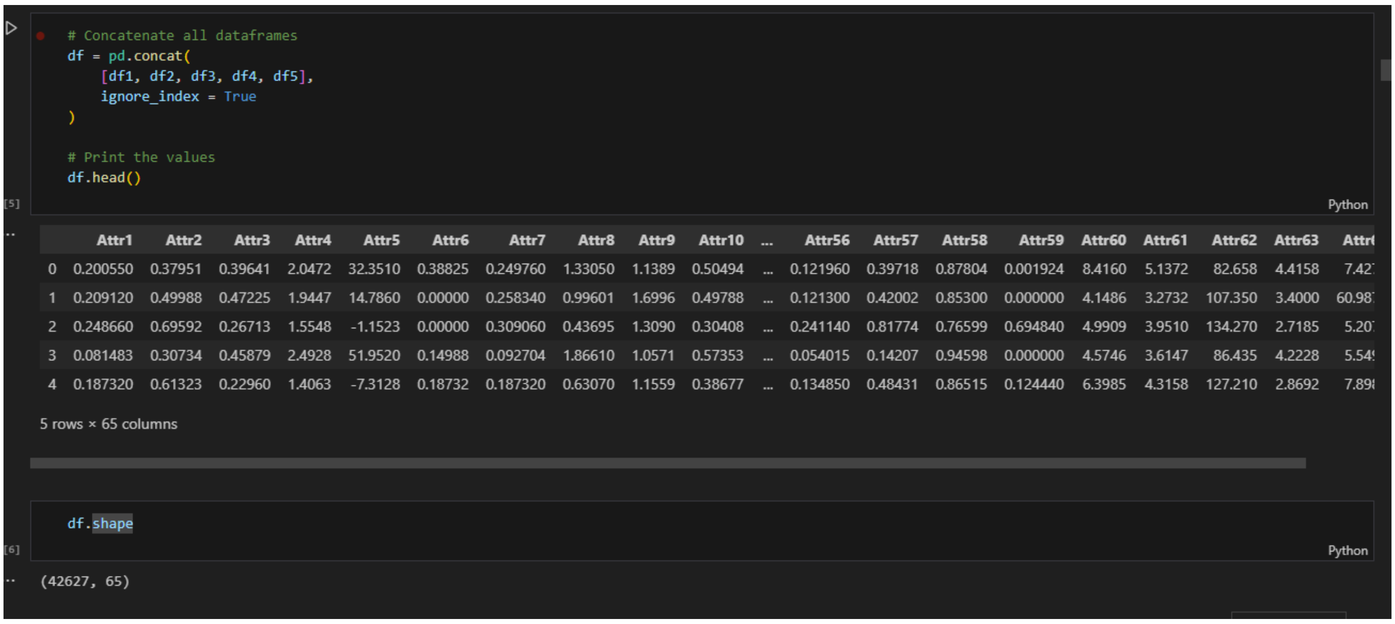
*Combining the files into one data frame*: We group the five datasets into one as they share the same properties and target. This allows us to conduct our analysis on a larger dataset with more observations.



- Checking if every CSV file has the same columns and if it is feasible to concatenate the files.

```
[4]
(
    list(df1.columns)
    == list(df2.columns)
    == list(df3.columns)
    == list(df4.columns)
    == list(df5.columns)
)
```

```
True
```



```
[5]
# Concatenate all dataframes
df = pd.concat(
    [df1, df2, df3, df4, df5],
    ignore_index = True
)

# Print the values
df.head()
```

Python

| | Attr1 | Attr2 | Attr3 | Attr4 | Attr5 | Attr6 | Attr7 | Attr8 | Attr9 | Attr10 | ... | Attr56 | Attr57 | Attr58 | Attr59 | Attr60 | Attr61 | Attr62 | Attr63 | Attr6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.200550 | 0.37951 | 0.39641 | 2.0472 | 32.3510 | 0.38825 | 0.249760 | 1.33050 | 1.1389 | 0.50494 | ... | 0.121960 | 0.39718 | 0.87804 | 0.001924 | 8.4160 | 5.1372 | 82.658 | 4.4158 | 7.42 |
| 1 | 0.209120 | 0.49988 | 0.47225 | 1.9447 | 14.7860 | 0.00000 | 0.258340 | 0.99601 | 1.6996 | 0.49788 | ... | 0.121300 | 0.42002 | 0.85300 | 0.000000 | 4.1486 | 3.2732 | 107.350 | 3.4000 | 60.98 |
| 2 | 0.248660 | 0.69592 | 0.26713 | 1.5548 | -1.1523 | 0.00000 | 0.309060 | 0.43695 | 1.3090 | 0.30408 | ... | 0.241140 | 0.81774 | 0.76599 | 0.694840 | 4.9909 | 3.9510 | 134.270 | 2.7185 | 5.20 |
| 3 | 0.081483 | 0.30734 | 0.45879 | 2.4928 | 51.9520 | 0.14988 | 0.092704 | 1.86610 | 1.0571 | 0.57353 | ... | 0.054015 | 0.14207 | 0.94598 | 0.000000 | 4.5746 | 3.6147 | 86.435 | 4.2228 | 5.54 |
| 4 | 0.187320 | 0.61323 | 0.22960 | 1.4063 | -7.3128 | 0.18732 | 0.187320 | 0.63070 | 1.1559 | 0.38677 | ... | 0.134850 | 0.48431 | 0.86515 | 0.124440 | 6.3985 | 4.3158 | 127.210 | 2.8692 | 7.89 |

5 rows × 65 columns

```
[6]
df.shape
```

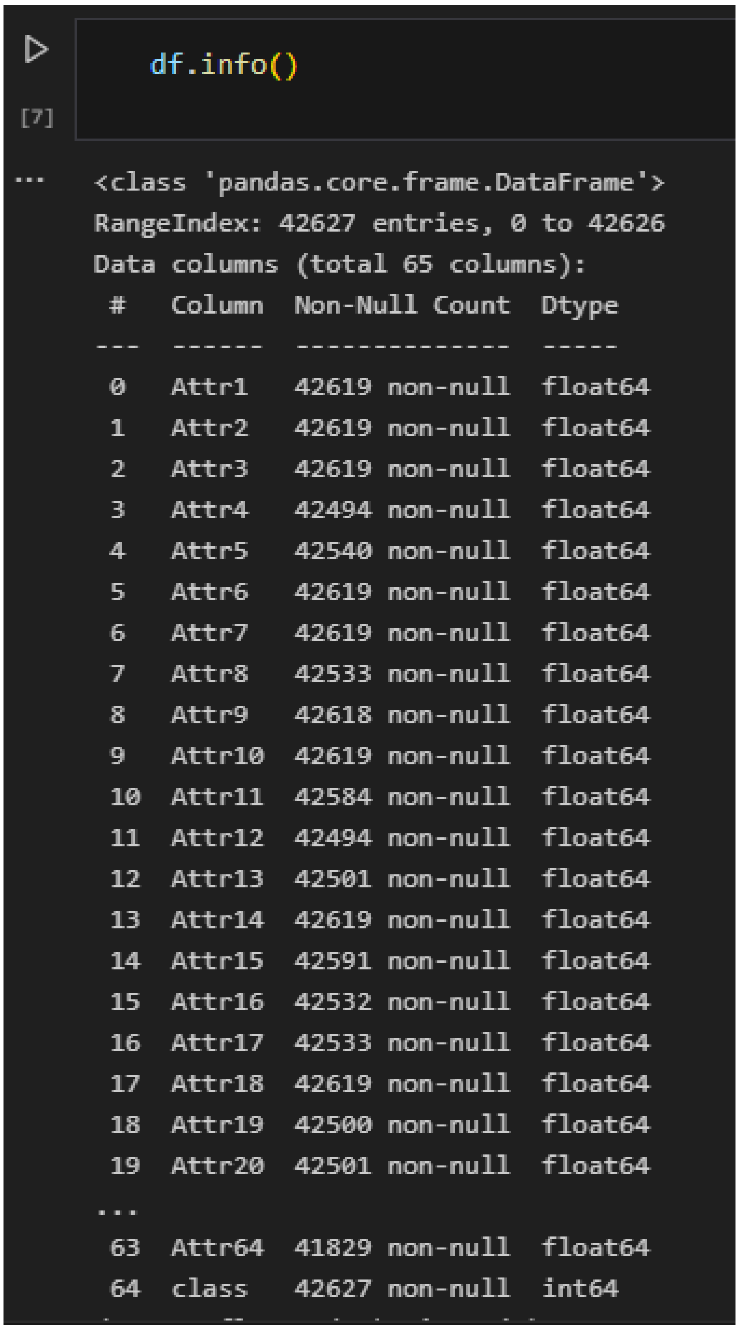
Python

```
(42627, 65)
```

Back to Index

## Data Preprocessing

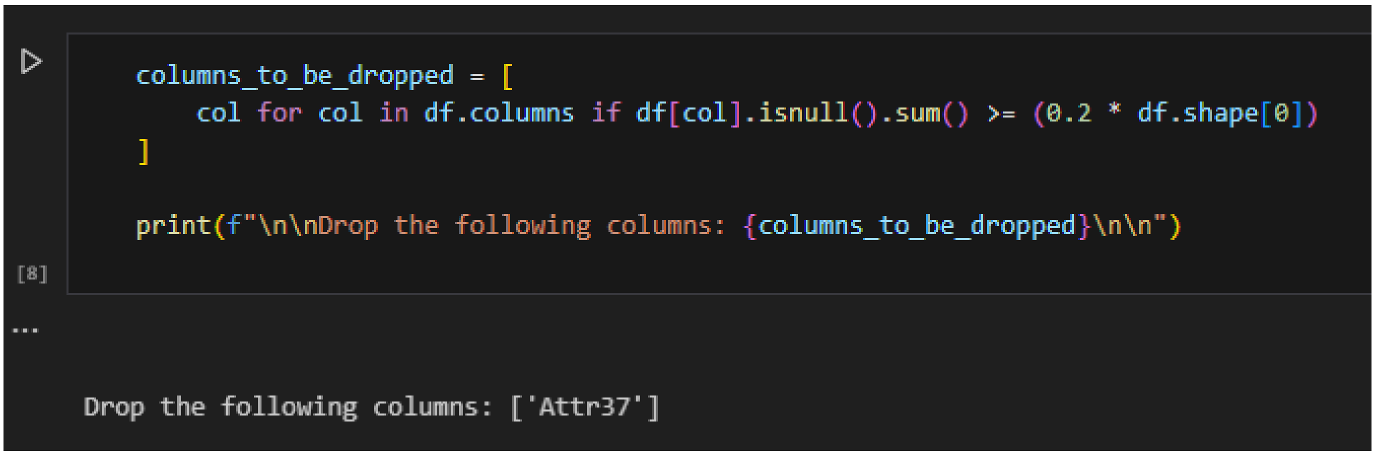
*Checking column information*: We use “df.info()” to get the preliminary information about our dataset.



```
df.info()
```

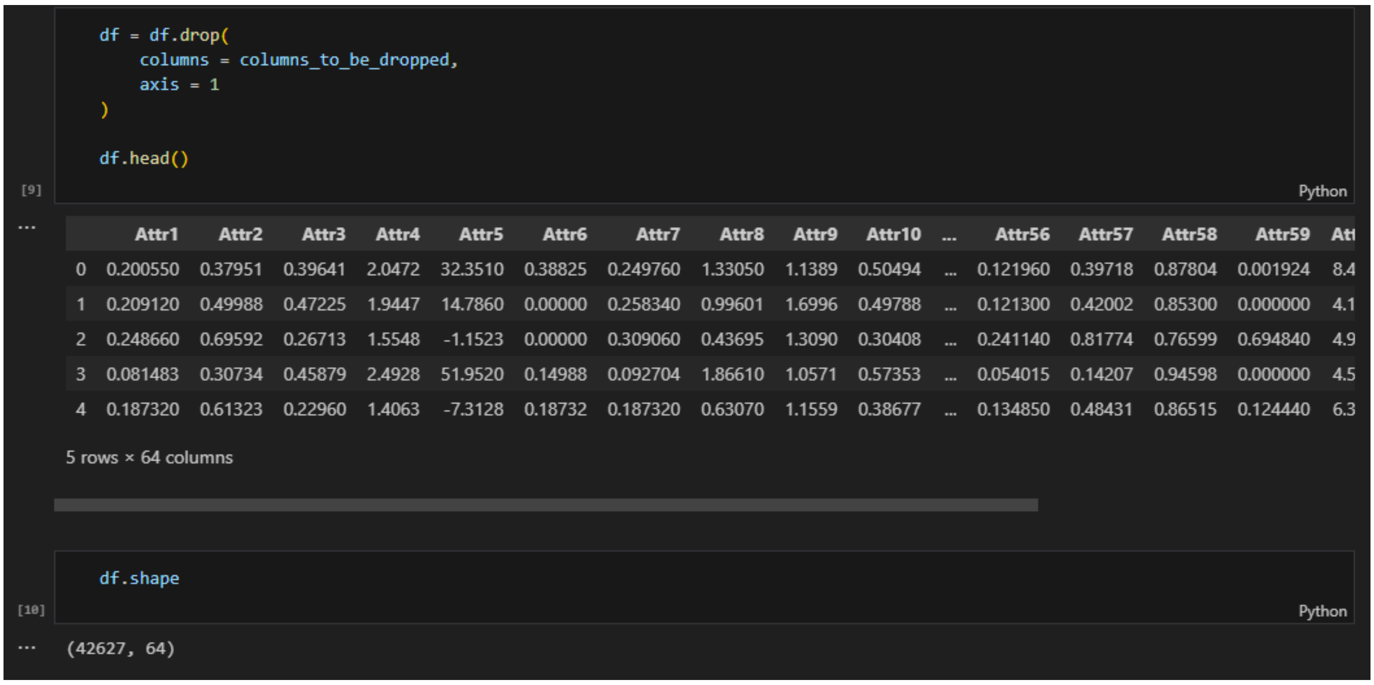
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42627 entries, 0 to 42626
Data columns (total 65 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Attr1   42619 non-null  float64
 1   Attr2   42619 non-null  float64
 2   Attr3   42619 non-null  float64
 3   Attr4   42494 non-null  float64
 4   Attr5   42540 non-null  float64
 5   Attr6   42619 non-null  float64
 6   Attr7   42619 non-null  float64
 7   Attr8   42533 non-null  float64
 8   Attr9   42618 non-null  float64
 9   Attr10  42619 non-null  float64
 10  Attr11  42584 non-null  float64
 11  Attr12  42494 non-null  float64
 12  Attr13  42501 non-null  float64
 13  Attr14  42619 non-null  float64
 14  Attr15  42591 non-null  float64
 15  Attr16  42532 non-null  float64
 16  Attr17  42533 non-null  float64
 17  Attr18  42619 non-null  float64
 18  Attr19  42500 non-null  float64
 19  Attr20  42501 non-null  float64
...
 63  Attr64  41829 non-null  float64
 64  class   42627 non-null  int64
```

*Dropping columns with large numbers of missing values*: Dropping columns with 20% missing data is a common practice in data preprocessing since keeping such columns introduces noise and reduces model performance. Reducing them enables cleaner and more relevant data, lowering the chance of incorrect predictions and strengthening the model. Furthermore, it simplifies the data processing pipeline, increasing performance and decreasing computational complexity.



```
[8]
columns_to_be_dropped = [
    col for col in df.columns if df[col].isnull().sum() >= (0.2 * df.shape[0])
]

print(f"\n\nDrop the following columns: {columns_to_be_dropped}\n\n")
```

```
...

Drop the following columns: ['Attr37']
```



```
[9]
df = df.drop(
    columns = columns_to_be_dropped,
    axis = 1
)

df.head()
```

Python

|  | Attr1 | Attr2 | Attr3 | Attr4 | Attr5 | Attr6 | Attr7 | Attr8 | Attr9 | Attr10 | ... | Attr56 | Attr57 | Attr58 | Attr59 | Att |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.200550 | 0.37951 | 0.39641 | 2.0472 | 32.3510 | 0.38825 | 0.249760 | 1.33050 | 1.1389 | 0.50494 | ... | 0.121960 | 0.39718 | 0.87804 | 0.001924 | 8.4 |
| 1 | 0.209120 | 0.49988 | 0.47225 | 1.9447 | 14.7860 | 0.00000 | 0.258340 | 0.99601 | 1.6996 | 0.49788 | ... | 0.121300 | 0.42002 | 0.85300 | 0.000000 | 4.1 |
| 2 | 0.248660 | 0.69592 | 0.26713 | 1.5548 | -1.1523 | 0.00000 | 0.309060 | 0.43695 | 1.3090 | 0.30408 | ... | 0.241140 | 0.81774 | 0.76599 | 0.694840 | 4.9 |
| 3 | 0.081483 | 0.30734 | 0.45879 | 2.4928 | 51.9520 | 0.14988 | 0.092704 | 1.86610 | 1.0571 | 0.57353 | ... | 0.054015 | 0.14207 | 0.94598 | 0.000000 | 4.5 |
| 4 | 0.187320 | 0.61323 | 0.22960 | 1.4063 | -7.3128 | 0.18732 | 0.187320 | 0.63070 | 1.1559 | 0.38677 | ... | 0.134850 | 0.48431 | 0.86515 | 0.124440 | 6.3 |

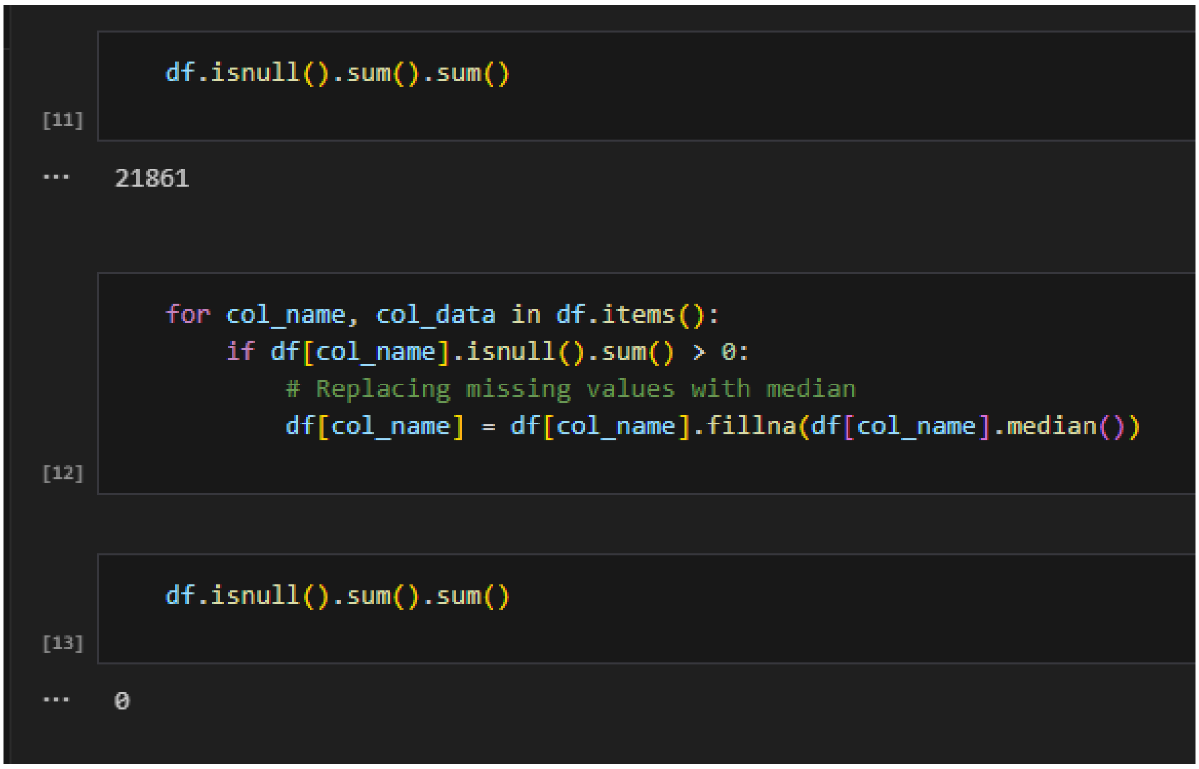
5 rows × 64 columns

```
[10]
df.shape
```

Python

```
... (42627, 64)
```

*Replacing the missing/null values*: It is essential that you replace missing values to maintain data integrity and ensure proper analysis. It prevents biassed results from occurring because missing data might skew statistics and lead to incorrect conclusions. Imputing missing values also enables us to make full use of the available data, allowing for robust modeling and decision-making while minimizing data loss. The "df.isnull()" function is used to check for null values. To find the number of null values, we use "df.isnull().sum()" function and for the sum of all null values, we use "df.isnull().sum().sum()" function. The missing values are replaced with the column's median since it is less sensitive to outliers than the mean. When data is sorted, the median indicates the middle value, making it a reliable indicator of central tendency.



```
[11]
df.isnull().sum().sum()
```

```
21861
```

```
[12]
for col_name, col_data in df.items():
    if df[col_name].isnull().sum() > 0:
        # Replacing missing values with median
        df[col_name] = df[col_name].fillna(df[col_name].median())
```

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

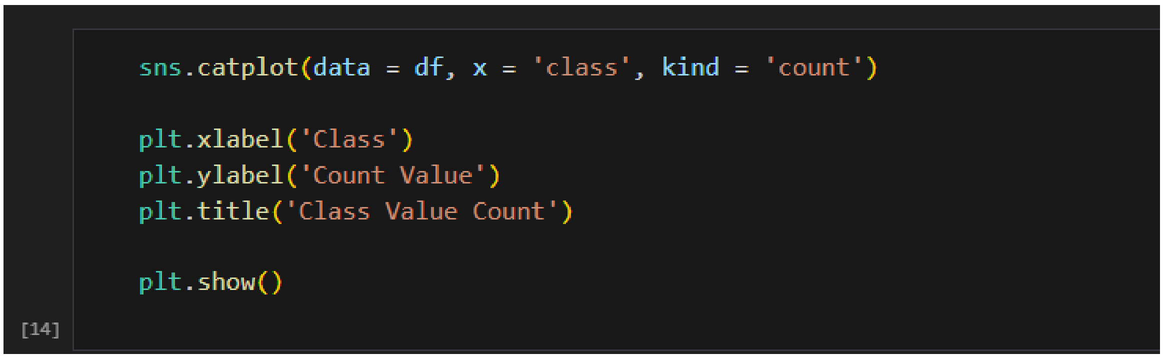
```
0
```

Upon replacing the missing values, we check for null values to ensure we have consistent data.

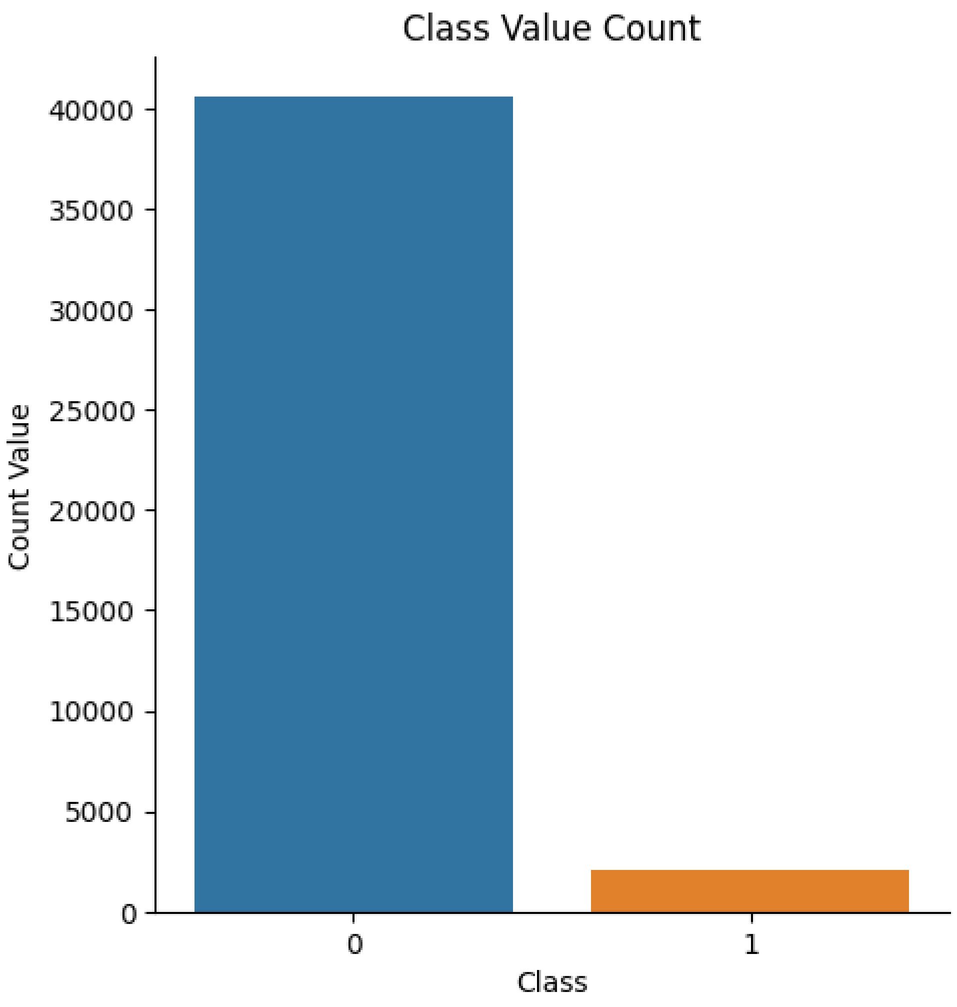
Back to Index

## Exploratory Data Analysis

*Class Distribution Visualization*: This code employs Seaborn to visualize the class distribution in the 'class' column. The x-axis represents the 'class' variable, and the y-axis shows count values. The graph reveals a notable class imbalance, underscoring the need for data balancing strategies to enhance model performance.

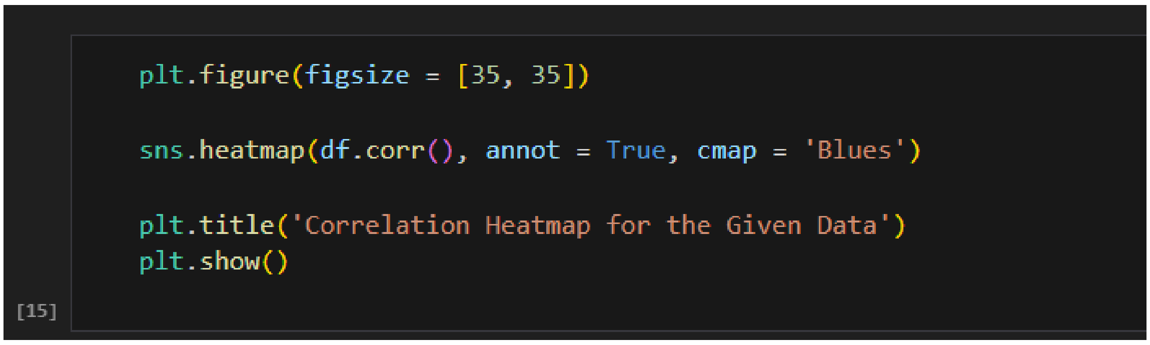


```
sns.catplot(data = df, x = 'class', kind = 'count')

plt.xlabel('Class')
plt.ylabel('Count Value')
plt.title('Class Value Count')

plt.show()
```



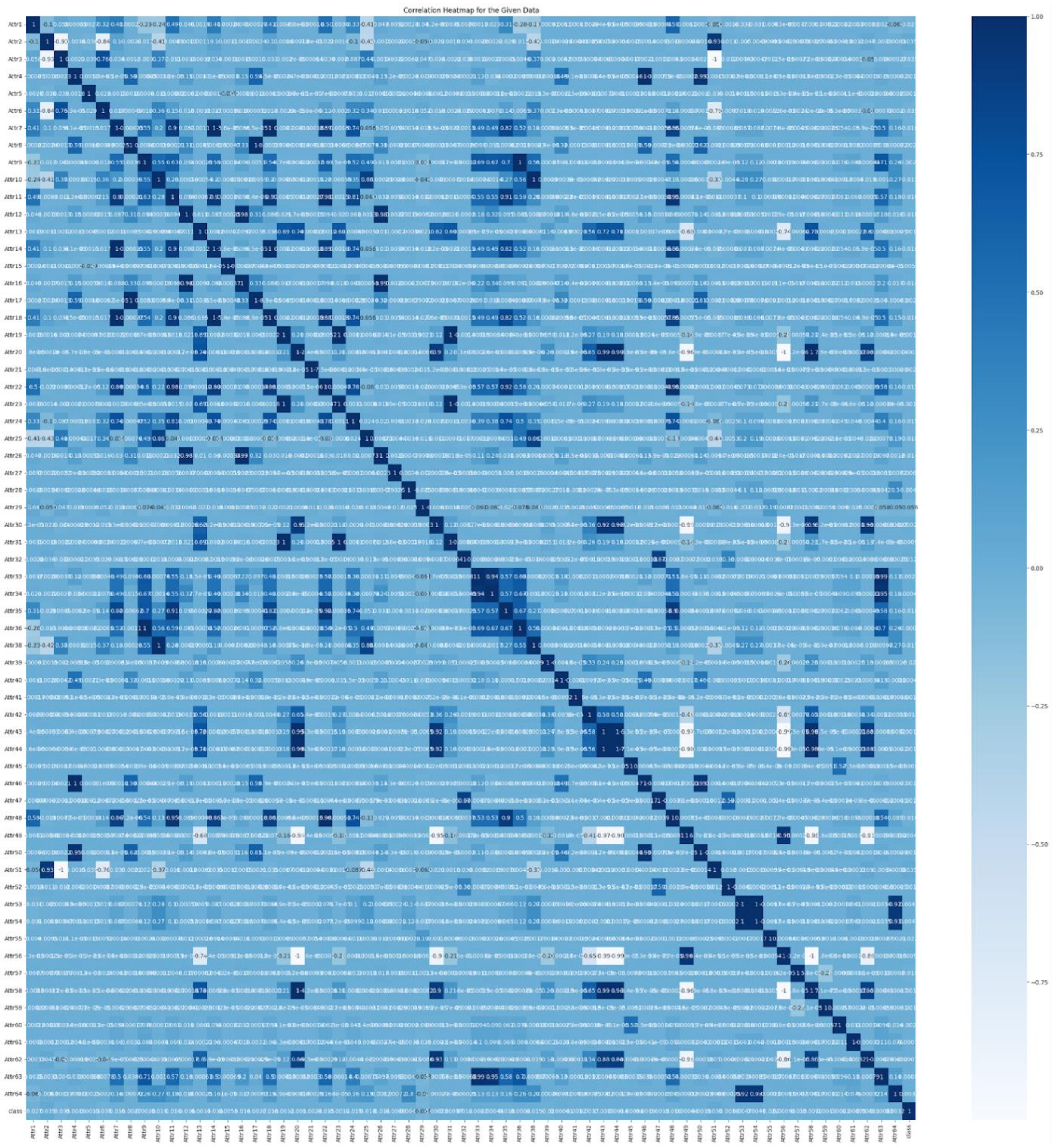
Back to Index

*Visualizing Variable Correlation with a Heatmap*: To gain insights into variable relationships within the dataset, we create a heatmap using Matplotlib and Seaborn. The 'figsize' parameter controls the plot size, while annotations display correlation values. We employ the 'Blues' color map, with darker shades indicating stronger correlations. This heatmap aids in exploring variable interactions, guiding feature selection and enhancing data pattern comprehension.

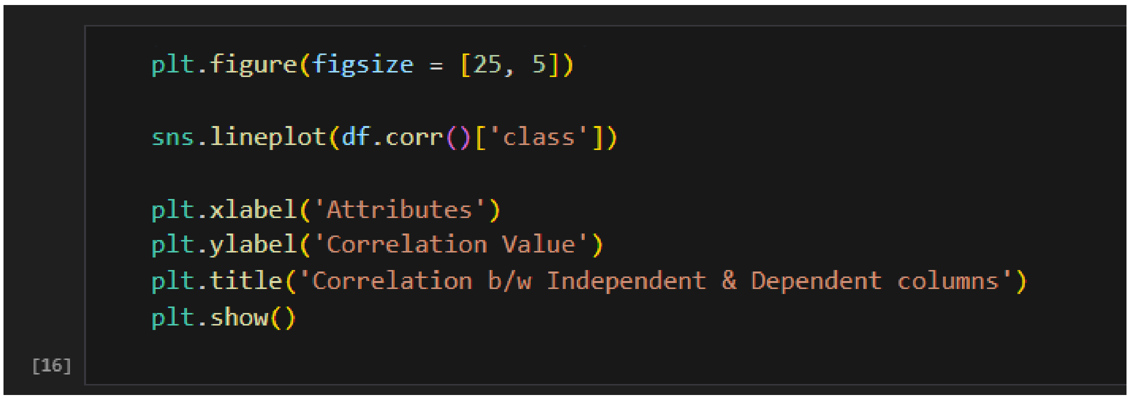


```
plt.figure(figsize = [35, 35])

sns.heatmap(df.corr(), annot = True, cmap = 'Blues')

plt.title('Correlation Heatmap for the Given Data')
plt.show()
```

[15]

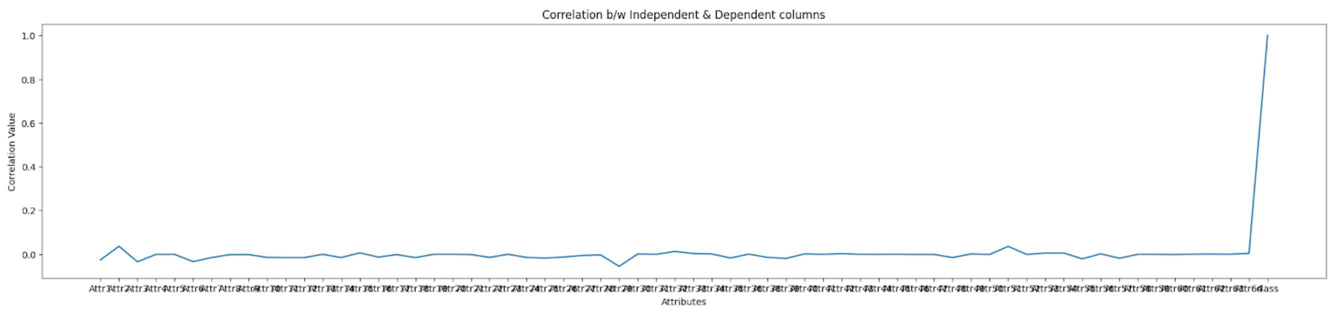


*Correlation Line Plot for Feature-Target Relationship*: In this visualization, Matplotlib and Seaborn collaborate to present a line plot illustrating the correlation between independent attributes and the dependent 'class' variable. The plot's dimensions are determined by the 'figsize' parameter, while 'sns.lineplot' displays the correlations. On the x-axis, attributes are represented, and on the y-axis, you'll find the correlation values. This informative plot sheds light on the connections between individual attributes and the target variable. It serves as a valuable tool for feature selection and understanding how attributes influence the target variable.



```
plt.figure(figsize = [25, 5])

sns.lineplot(df.corr()['class'])

plt.xlabel('Attributes')
plt.ylabel('Correlation Value')
plt.title('Correlation b/w Independent & Dependent columns')
plt.show()
```

[16]

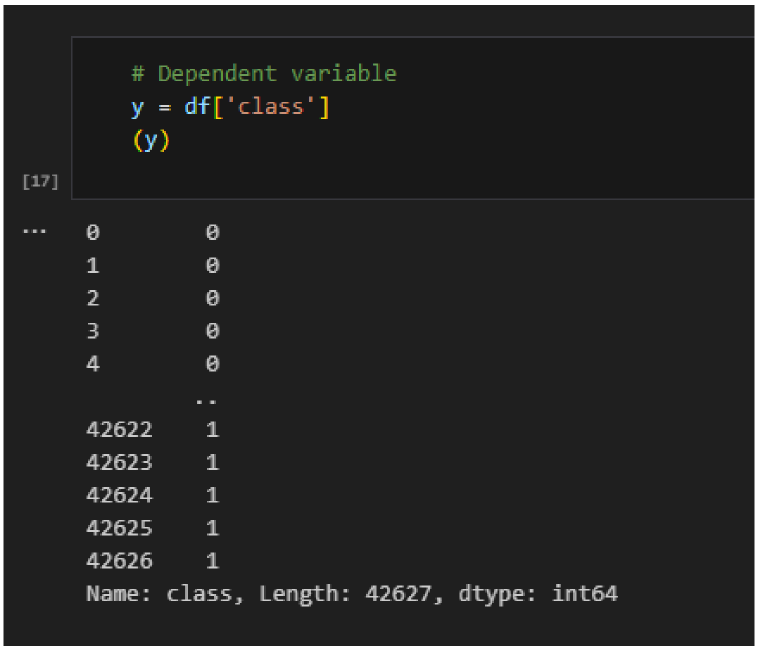


Back to Index

## X and Y Splitting

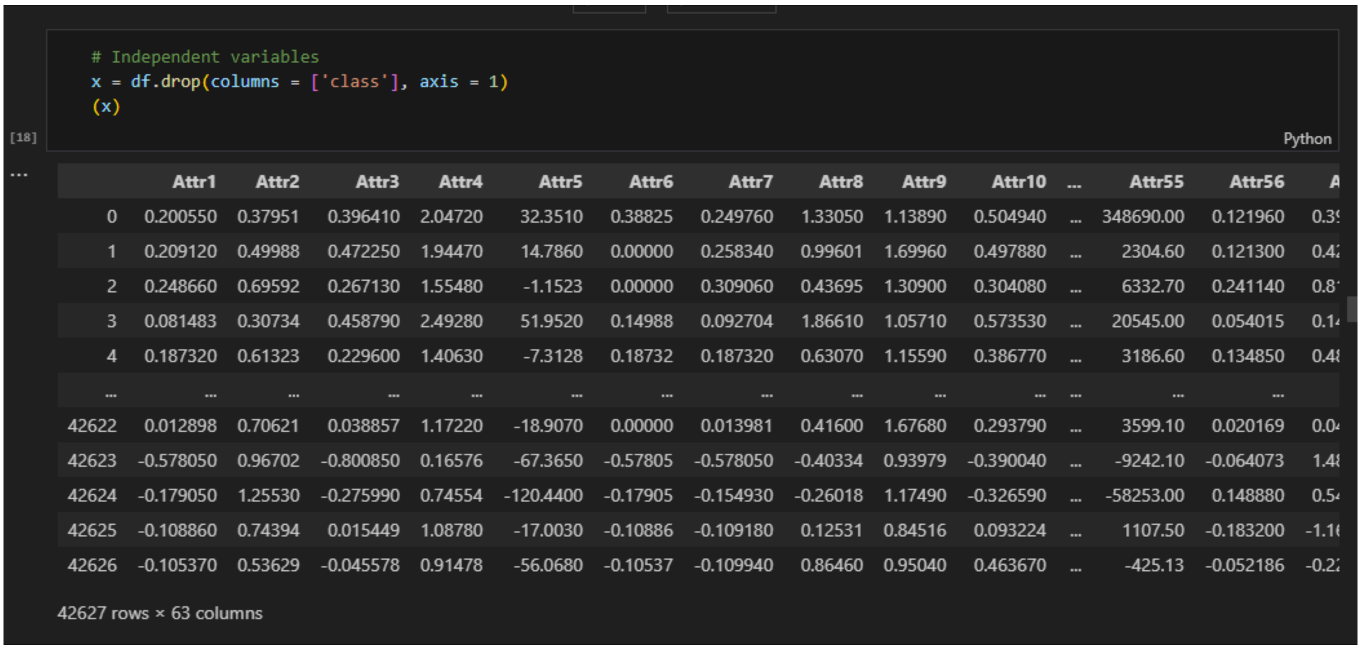
In machine learning, an "x-y split" is a crucial step that separates a dataset into two distinct components: the independent variables (often denoted as 'X') and the dependent variable or target (typically denoted as 'Y'). The 'X' component comprises the input features or attributes that the model uses to make predictions, while 'Y' represents the variable the model aims to predict. This separation is essential for model training, as it allows the algorithm to learn from the relationships between 'X' and 'Y' and later make accurate predictions based on unseen 'X' values. It also facilitates rigorous model evaluation by testing its ability to generalize to new data points. The dependent variable, 'class,' is assigned to 'y,' while the independent variables are assigned to 'x' after excluding the 'class' column.

This division simplifies the process of training and validating models, preventing data leakage and enabling rigorous testing of a model's generalization to new, unseen data. It ensures precise predictions and reliable performance evaluation, a fundamental practice in machine learning and data analysis.



```
[17]
# Dependent variable
y = df['class']
(y)
```

```
0        0
1        0
2        0
3        0
4        0
        ..
42622    1
42623    1
42624    1
42625    1
42626    1
Name: class, Length: 42627, dtype: int64
```



```
[18]
# Independent variables
x = df.drop(columns = ['class'], axis = 1)
(x)
```

| | Attr1 | Attr2 | Attr3 | Attr4 | Attr5 | Attr6 | Attr7 | Attr8 | Attr9 | Attr10 | ... | Attr55 | Attr56 | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.200550 | 0.37951 | 0.396410 | 2.04720 | 32.3510 | 0.38825 | 0.249760 | 1.33050 | 1.13890 | 0.504940 | ... | 348690.00 | 0.121960 | 0.3 |
| 1 | 0.209120 | 0.49988 | 0.472250 | 1.94470 | 14.7860 | 0.00000 | 0.258340 | 0.99601 | 1.69960 | 0.497880 | ... | 2304.60 | 0.121300 | 0.4 |
| 2 | 0.248660 | 0.69592 | 0.267130 | 1.55480 | -1.1523 | 0.00000 | 0.309060 | 0.43695 | 1.30900 | 0.304080 | ... | 6332.70 | 0.241140 | 0.8 |
| 3 | 0.081483 | 0.30734 | 0.458790 | 2.49280 | 51.9520 | 0.14988 | 0.092704 | 1.86610 | 1.05710 | 0.573530 | ... | 20545.00 | 0.054015 | 0.1 |
| 4 | 0.187320 | 0.61323 | 0.229600 | 1.40630 | -7.3128 | 0.18732 | 0.187320 | 0.63070 | 1.15590 | 0.386770 | ... | 3186.60 | 0.134850 | 0.4 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 42622 | 0.012898 | 0.70621 | 0.038857 | 1.17220 | -18.9070 | 0.00000 | 0.013981 | 0.41600 | 1.67680 | 0.293790 | ... | 3599.10 | 0.020169 | 0.0 |
| 42623 | -0.578050 | 0.96702 | -0.800850 | 0.16576 | -67.3650 | -0.57805 | -0.578050 | -0.40334 | 0.93979 | -0.390040 | ... | -9242.10 | -0.064073 | 1.4 |
| 42624 | -0.179050 | 1.25530 | -0.275990 | 0.74554 | -120.4400 | -0.17905 | -0.154930 | -0.26018 | 1.17490 | -0.326590 | ... | -58253.00 | 0.148880 | 0.5 |
| 42625 | -0.108860 | 0.74394 | 0.015449 | 1.08780 | -17.0030 | -0.10886 | -0.109180 | 0.12531 | 0.84516 | 0.093224 | ... | 1107.50 | -0.183200 | -1.1 |
| 42626 | -0.105370 | 0.53629 | -0.045578 | 0.91478 | -56.0680 | -0.10537 | -0.109940 | 0.86460 | 0.95040 | 0.463670 | ... | -425.13 | -0.052186 | -0.2 |

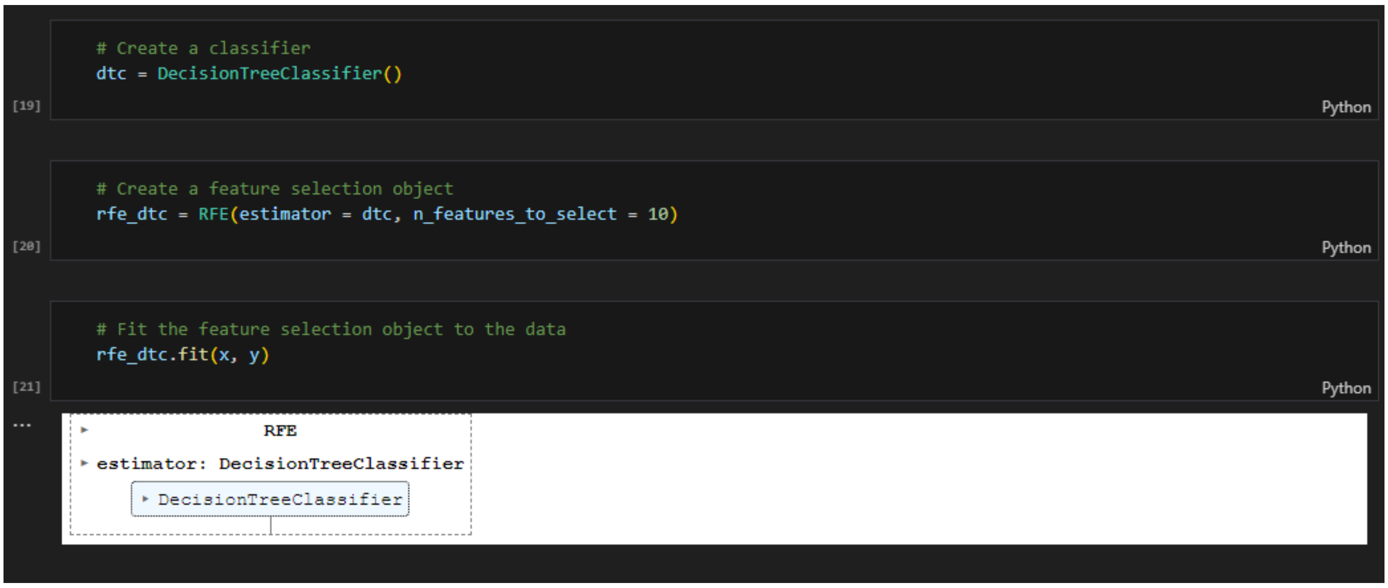
42627 rows × 63 columns

## Feature Selection

We harness the potential of three machine learning classifiers – Decision Tree, Random Forest, and XGBoost – coupled with feature selection techniques to optimize model performance. Each classifier is paired with an RFE-based feature selection object, zeroing in on the ten most influential attributes. By fitting these objects to input features (X) and the target variable (Y), we systematically identify the most significant attributes, streamlining the dataset and boosting interpretability. This often leads to higher model accuracy, ensuring data-driven decision-making and efficient results.

To make this process transparent and accessible, we create lists for each classifier, storing the selected features. We iteratively examine input feature columns, using RFE support arrays to identify the top ten attributes. The resulting feature lists provide a clear overview of critical attributes that profoundly affect model performance, enhancing transparency and feature selection for more effective outcomes.

*Decision Tree Classifier*

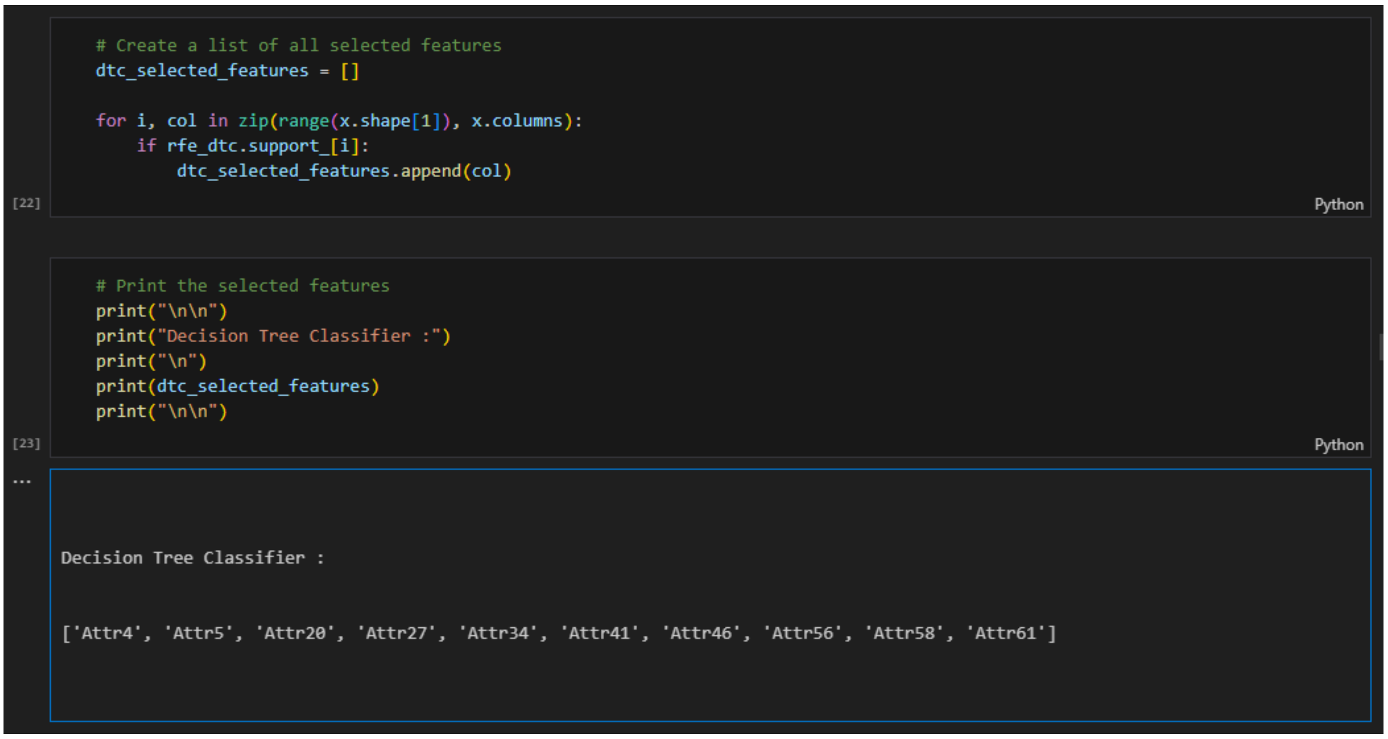


```
# Create a classifier
dtc = DecisionTreeClassifier()
```
[19] Python

```
# Create a feature selection object
rfe_dtc = RFE(estimator = dtc, n_features_to_select = 10)
```
[20] Python

```
# Fit the feature selection object to the data
rfe_dtc.fit(x, y)
```
[21] Python

```
RFE
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

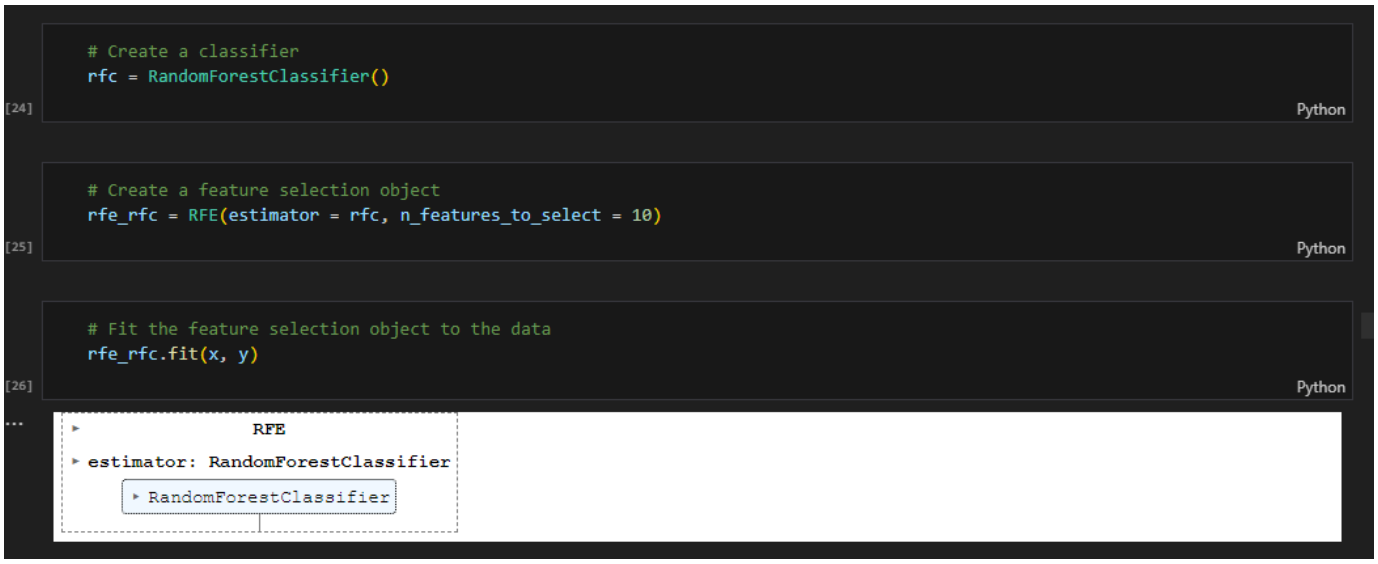


```
# Create a list of all selected features
dtc_selected_features = []

for i, col in zip(range(x.shape[1]), x.columns):
    if rfe_dtc.support_[i]:
        dtc_selected_features.append(col)
```
[22] Python

```
# Print the selected features
print("\n\n")
print("Decision Tree Classifier :")
print("\n")
print(dtc_selected_features)
print("\n\n")
```
[23] Python

```


Decision Tree Classifier :


['Attr4', 'Attr5', 'Attr20', 'Attr27', 'Attr34', 'Attr41', 'Attr46', 'Attr56', 'Attr58', 'Attr61']


```

*Random Forest Classifier*



```
[24]
# Create a classifier
rfc = RandomForestClassifier()
```

Python

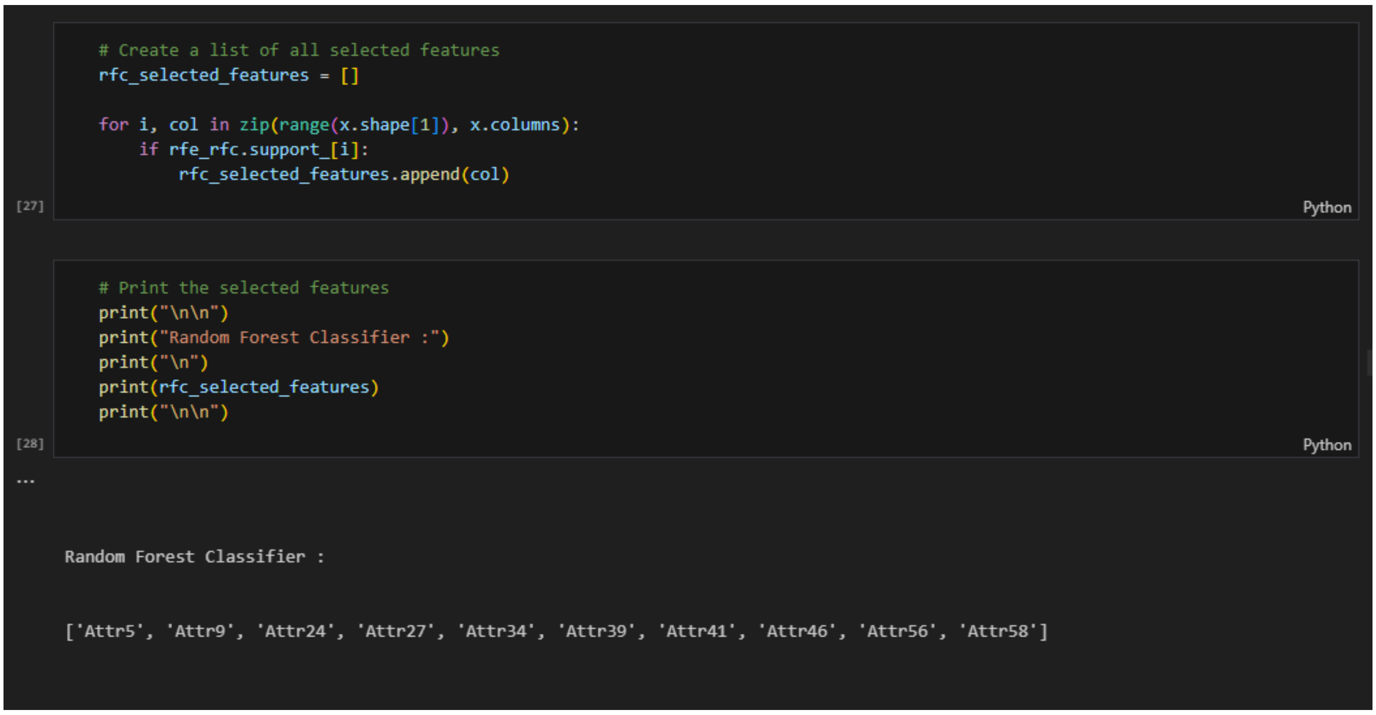
```
[25]
# Create a feature selection object
rfe_rfc = RFE(estimator = rfc, n_features_to_select = 10)
```

Python

```
[26]
# Fit the feature selection object to the data
rfe_rfc.fit(x, y)
```

Python

```
...
▸ RFE
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```



```
[27]
# Create a list of all selected features
rfc_selected_features = []

for i, col in zip(range(x.shape[1]), x.columns):
    if rfe_rfc.support_[i]:
        rfc_selected_features.append(col)
```

Python

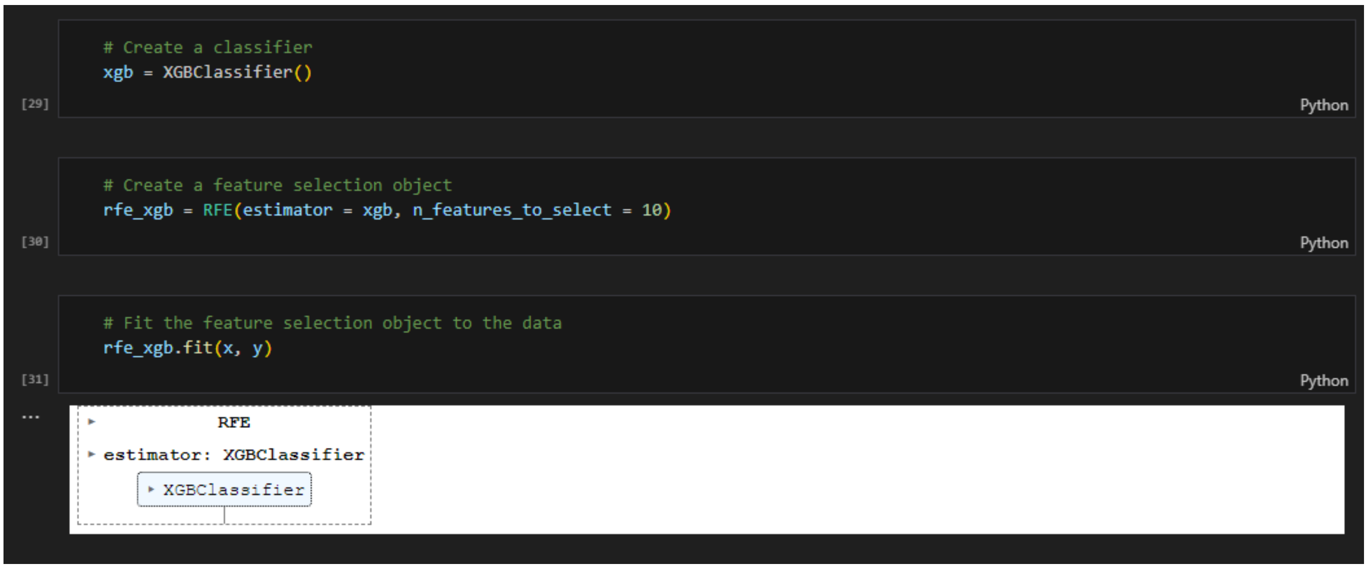
```
[28]
# Print the selected features
print("\n\n")
print("Random Forest Classifier :")
print("\n")
print(rfc_selected_features)
print("\n\n")
```

Python

```
...


Random Forest Classifier :


['Attr5', 'Attr9', 'Attr24', 'Attr27', 'Attr34', 'Attr39', 'Attr41', 'Attr46', 'Attr56', 'Attr58']
```

Back to Index

## *XGB Classifier*

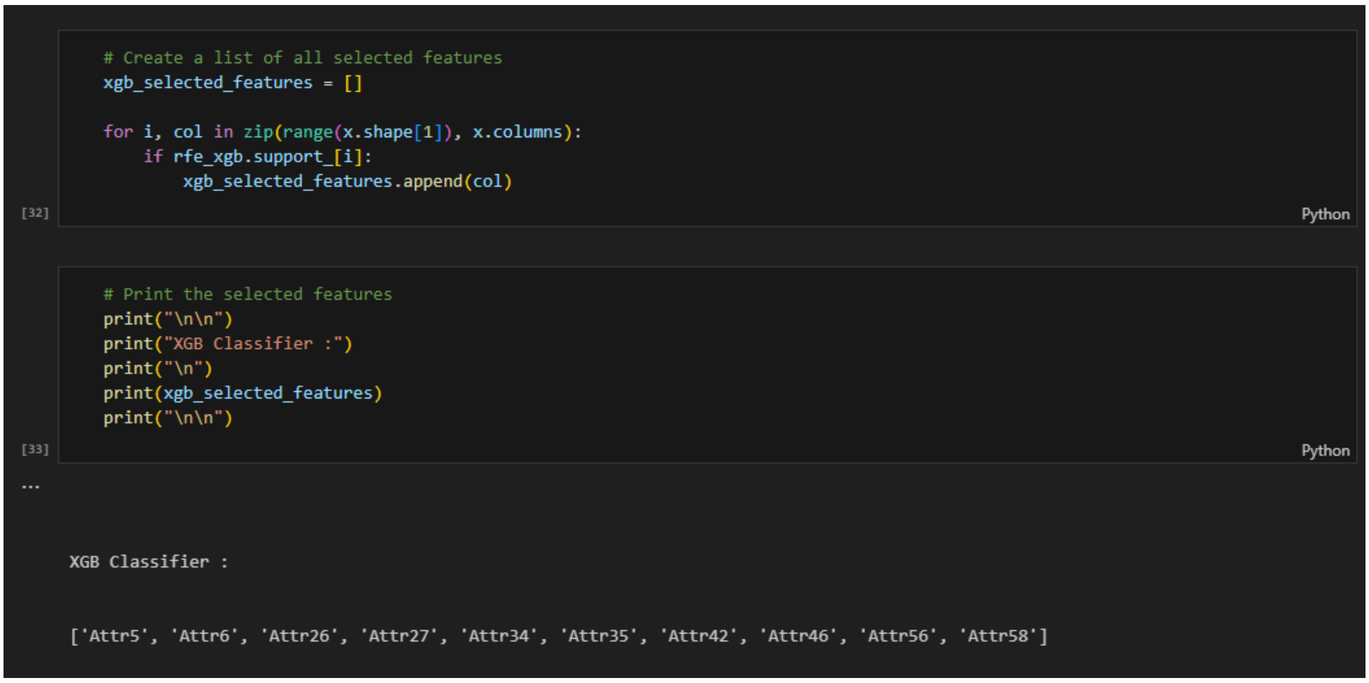


```
# Create a classifier
xgb = XGBClassifier()
```
[29] Python

```
# Create a feature selection object
rfe_xgb = RFE(estimator = xgb, n_features_to_select = 10)
```
[30] Python

```
# Fit the feature selection object to the data
rfe_xgb.fit(x, y)
```
[31] Python

```
...
▸ RFE
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```



```
# Create a list of all selected features
xgb_selected_features = []

for i, col in zip(range(x.shape[1]), x.columns):
    if rfe_xgb.support_[i]:
        xgb_selected_features.append(col)
```
[32] Python

```
# Print the selected features
print("\n\n")
print("XGB Classifier :")
print("\n")
print(xgb_selected_features)
print("\n\n")
```
[33] Python

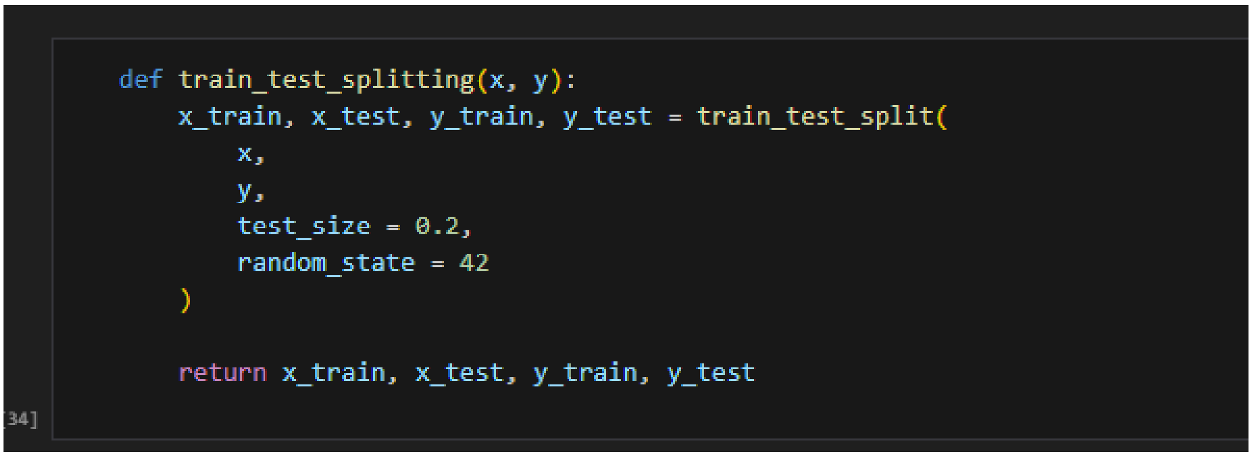
```
...


XGB Classifier :


['Attr5', 'Attr6', 'Attr26', 'Attr27', 'Attr34', 'Attr35', 'Attr42', 'Attr46', 'Attr56', 'Attr58']
```

Back to Index

## MODEL BUILDING

### Train-test Splitting

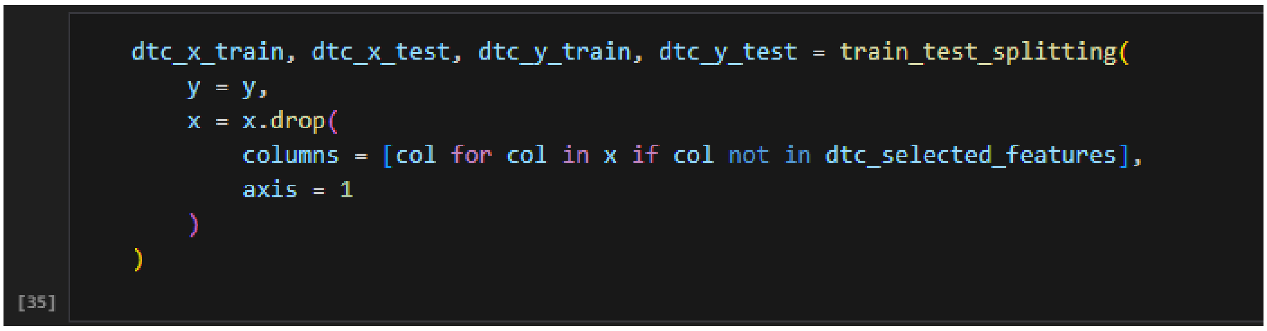
We've introduced a custom function called 'train_test_splitting' designed to streamline the data division into training and testing sets. This function accepts input features ('x') and target variables ('y'). By utilizing Scikit-Learn's 'train_test_split' function, it efficiently partitions the data, dedicating 20% to testing and employing a specific random seed (in this instance, set to 42 for replicability). The function conveniently returns the resulting training and testing datasets for both input features and target variables. This simplifies the vital process of data splitting, a crucial stage in the development of machine learning models.



```
def train_test_splitting(x, y):
    x_train, x_test, y_train, y_test = train_test_split(
        x,
        y,
        test_size = 0.2,
        random_state = 42
    )

    return x_train, x_test, y_train, y_test
```

[34]

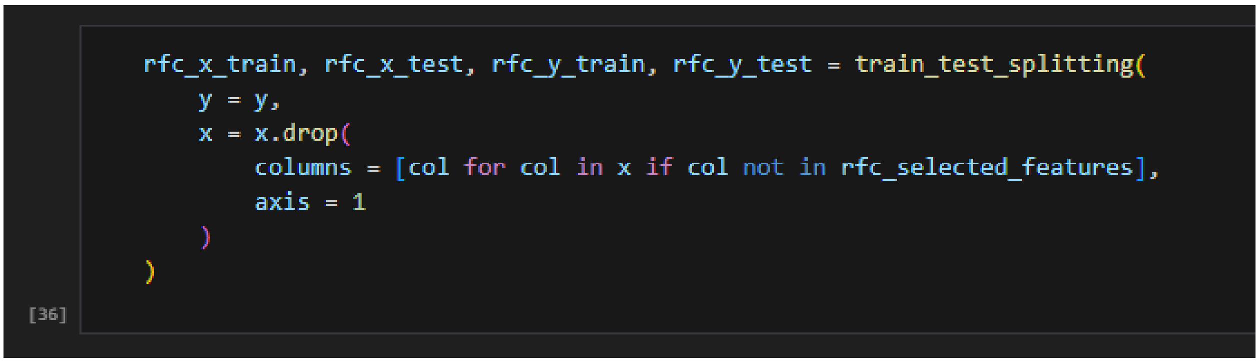
*Decision Tree Classifier*



```
dtc_x_train, dtc_x_test, dtc_y_train, dtc_y_test = train_test_splitting(
    y = y,
    x = x.drop(
        columns = [col for col in x if col not in dtc_selected_features],
        axis = 1
    )
)
```

[35]

*Random Forest Classifier*

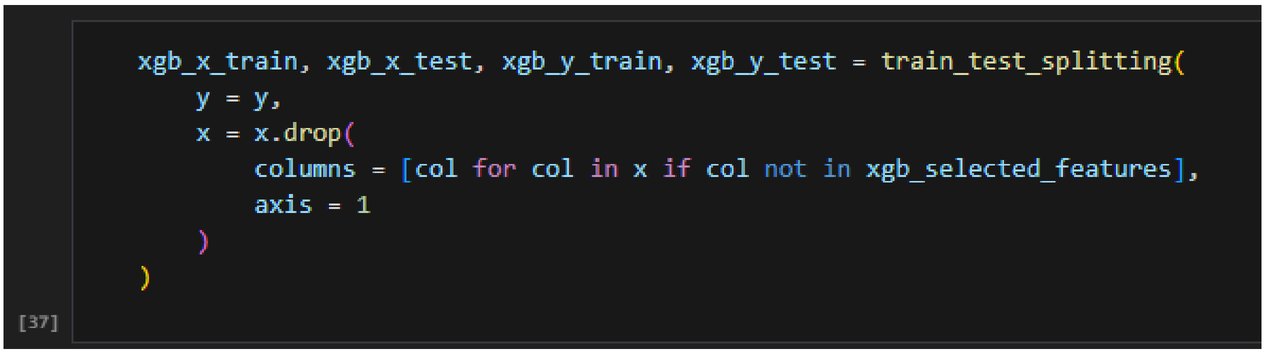


```
rfc_x_train, rfc_x_test, rfc_y_train, rfc_y_test = train_test_splitting(
    y = y,
    x = x.drop(
        columns = [col for col in x if col not in rfc_selected_features],
        axis = 1
    )
)
```

[36]

Back to Index

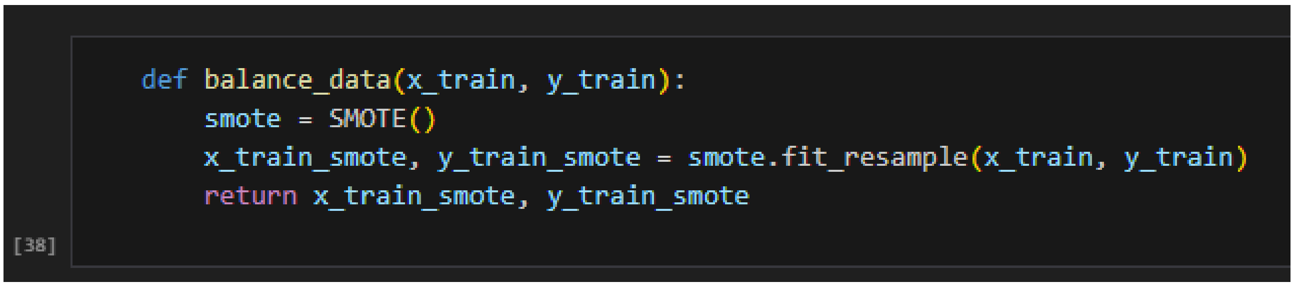
*XGB Classifier*



```
[37]
xgb_x_train, xgb_x_test, xgb_y_train, xgb_y_test = train_test_splitting(
    y = y,
    x = x.drop(
        columns = [col for col in x if col not in xgb_selected_features],
        axis = 1
    )
)
```

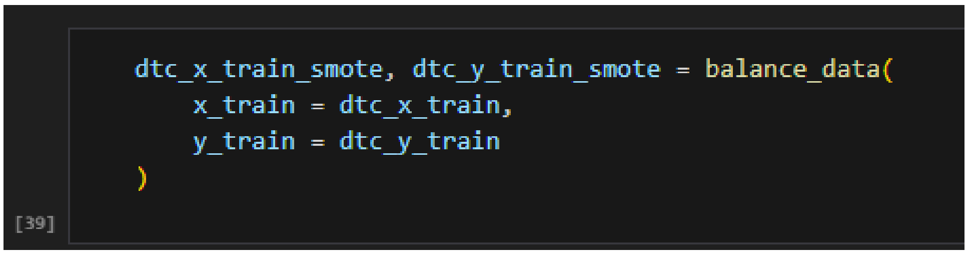
## Handling Imbalanced Data

Handling imbalanced data is crucial because it prevents machine learning models from favoring the majority class, resulting in more balanced predictions. Neglecting imbalances can result in misleadingly high accuracy and untrustworthy findings, particularly in high-stakes settings. Balancing strategies improve generalization and fairness in models, making them more applicable and dependable in real-world circumstances.



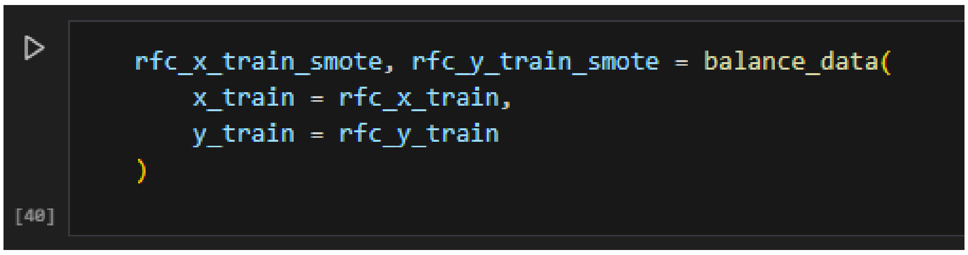
```
[38]
def balance_data(x_train, y_train):
    smote = SMOTE()
    x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)
    return x_train_smote, y_train_smote
```

*Decision Tree Classifier*



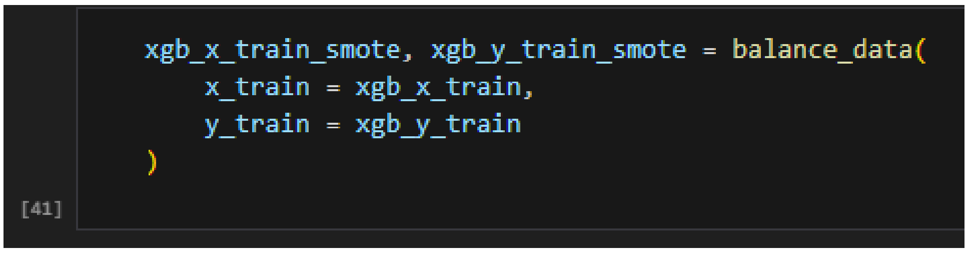
```
[39]
dtc_x_train_smote, dtc_y_train_smote = balance_data(
    x_train = dtc_x_train,
    y_train = dtc_y_train
)
```

*Random Forest Classifier*



```
[40]
rfc_x_train_smote, rfc_y_train_smote = balance_data(
    x_train = rfc_x_train,
    y_train = rfc_y_train
)
```

*XGB Classifier*



```
[41]
xgb_x_train_smote, xgb_y_train_smote = balance_data(
    x_train = xgb_x_train,
    y_train = xgb_y_train
)
```

Back to Index

## Building the Models

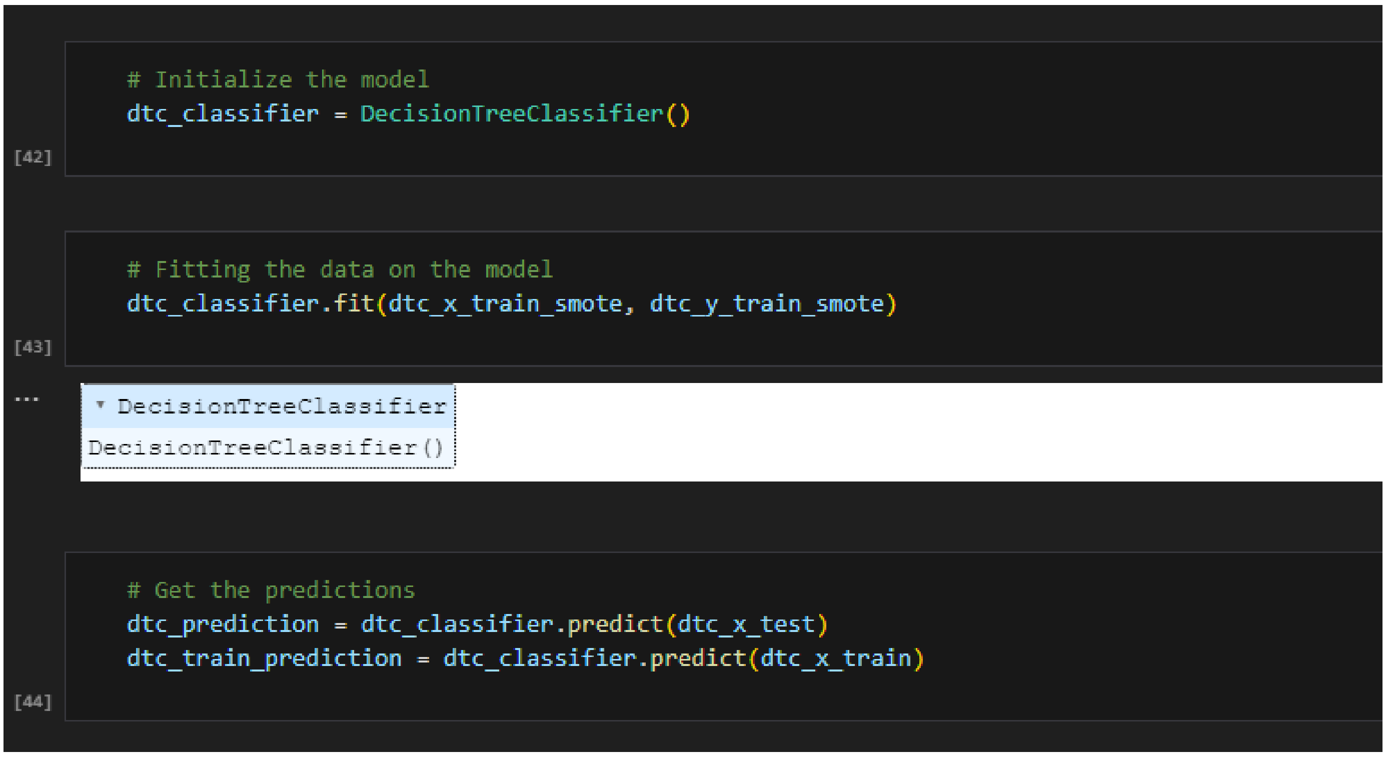
Within the realm of machine learning, the concept of model construction revolves around the art of training a predictive algorithm or model using a labeled dataset. This process involves feeding the model a combination of features and target variables, enabling it to unearth intricate patterns and correlations within the data. Crafting a model often involves the delicate task of refining hyperparameters, choosing the most fitting algorithm, and enhancing its overall efficacy. The ultimate objective is to engineer a sturdy and dependable predictive model, equipped to make insightful predictions on entirely new, uncharted data. In doing so, it streamlines data-driven decision-making across a diverse spectrum of applications.

### *Decision Tree Classifier*

The model is initiated using the `DecisionTreeClassifier`, creating the `dtc_classifier` instance. It's then trained with the balanced training data, denoted as `dtc_x_train_smote` and `dtc_y_train_smote`. This training process enables the model to identify patterns within the data.

After training, the model is used to make predictions on two datasets. The first is the test dataset, `dtc_x_test`, which assesses the model's performance. Predictions are stored in `dtc_prediction`. The second is the training dataset, `dtc_x_train`, used for reference, with predictions saved in `dtc_train_prediction`.

These predictions serve to evaluate the model's accuracy, performance, and generalization capabilities on both the test and training data, ensuring its suitability for the classification task.



```
# Initialize the model
dtc_classifier = DecisionTreeClassifier()
[42]
```

```
# Fitting the data on the model
dtc_classifier.fit(dtc_x_train_smote, dtc_y_train_smote)
[43]
```

```
...
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

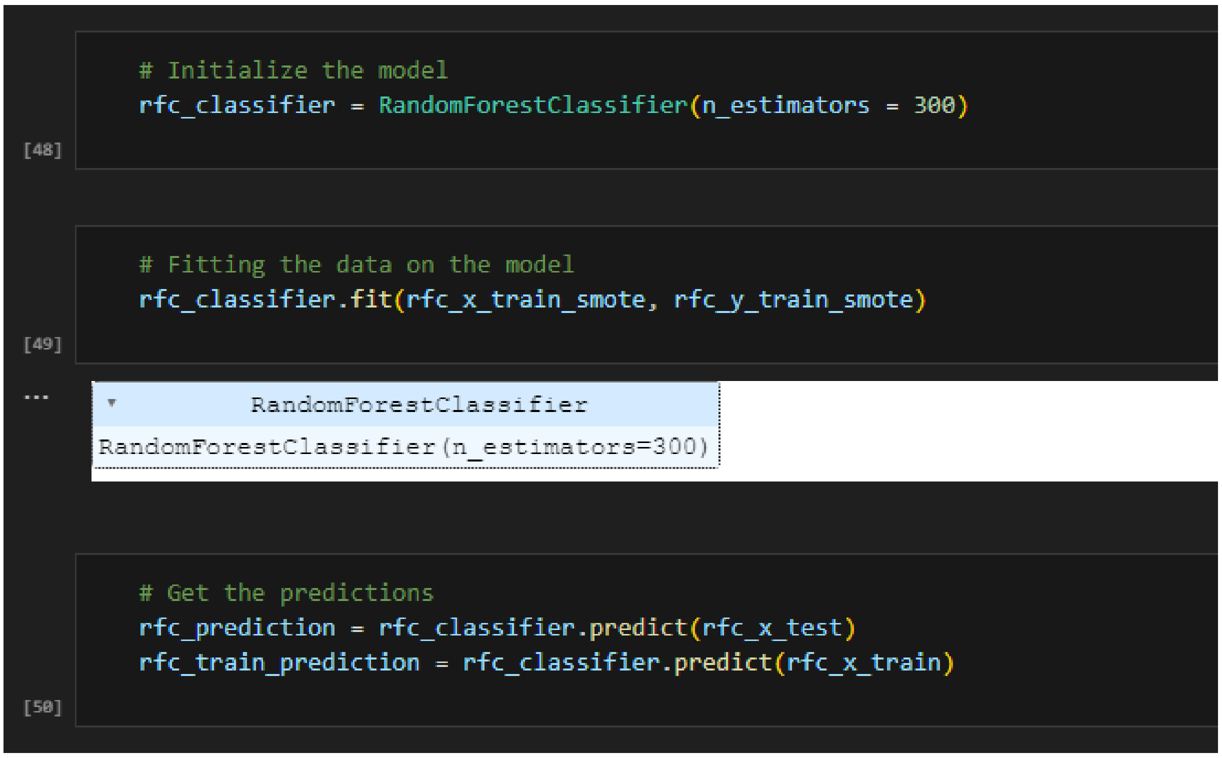
```
# Get the predictions
dtc_prediction = dtc_classifier.predict(dtc_x_test)
dtc_train_prediction = dtc_classifier.predict(dtc_x_train)
[44]
```

*Random Forest Classifier*

An instance of the `RandomForestClassifier` class is created to initialize the model, named `rfc_classifier`, with 300 decision trees. This model is then trained on the balanced training data, which underwent SMOTE resampling. The training data includes `rfc_x_train_smote` for input features and `rfc_y_train_smote` for the target variable, enabling the model to capture data patterns and relationships.

The trained Random Forest model is used to predict two datasets: `rfc_x_test` for evaluating model performance, with predictions stored in `rfc_prediction`, and `rfc_x_train` for reference, with predictions saved in `rfc_train_prediction`.

These predictions are crucial for assessing model accuracy, overall performance, and its ability to generalize to new data. This process is vital for evaluating the model's effectiveness and suitability for the binary classification task.



```
[48]
# Initialize the model
rfc_classifier = RandomForestClassifier(n_estimators = 300)
```

```
[49]
# Fitting the data on the model
rfc_classifier.fit(rfc_x_train_smote, rfc_y_train_smote)
```

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=300)
```

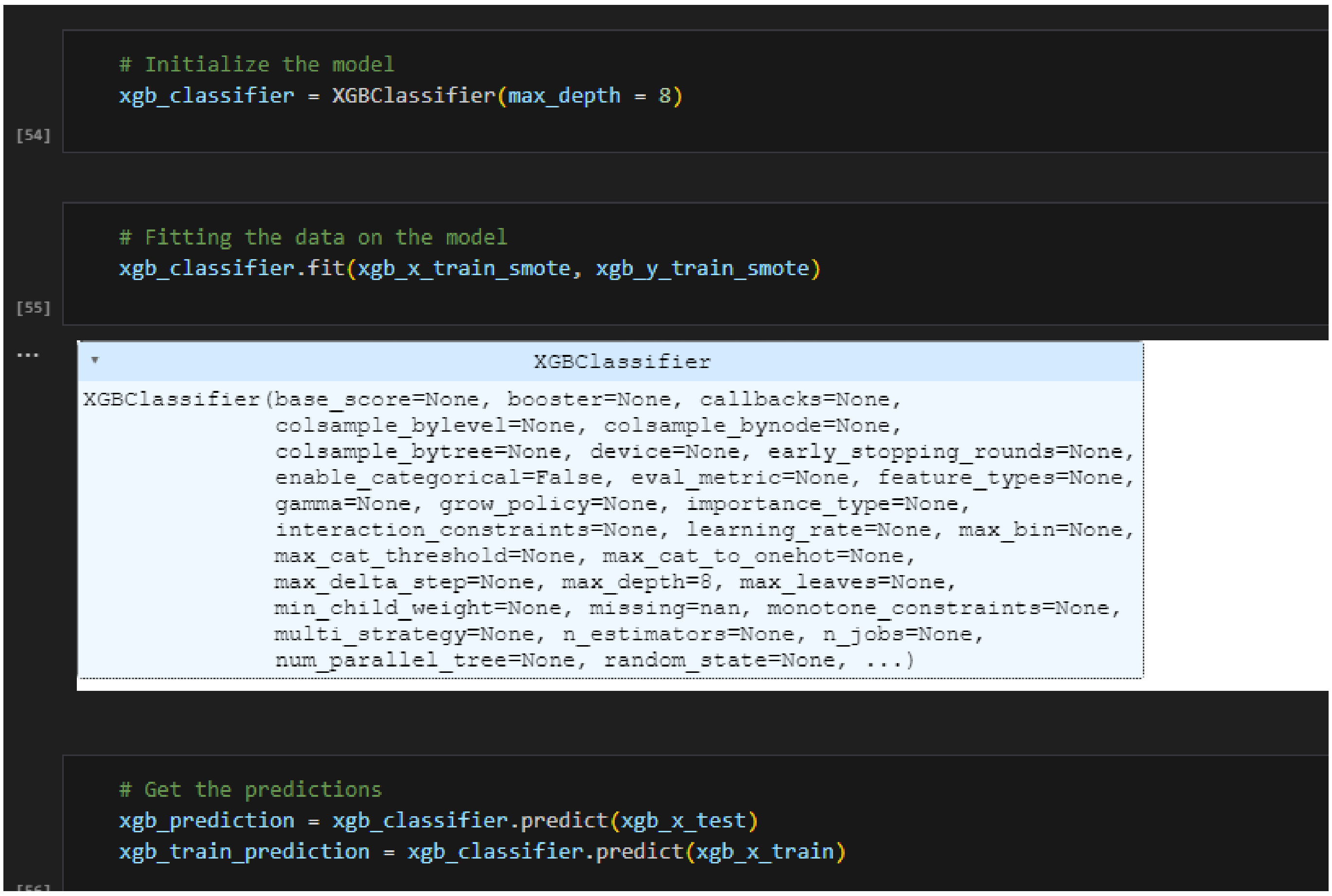
```
[50]
# Get the predictions
rfc_prediction = rfc_classifier.predict(rfc_x_test)
rfc_train_prediction = rfc_classifier.predict(rfc_x_train)
```

*XGB Classifier*

The XGBoost Classifier model is initialized using the `XGBClassifier` class, configured with a maximum tree depth of eight (parameter: `max_depth`). This model instance is named `xgb_classifier`. Following initialization, the model is trained with the SMOTE-balanced training data, comprising `xgb_x_train_smote` for input features and `xgb_y_train_smote` for the target variable, allowing the model to learn data patterns and relationships.

The trained XGBoost model is used to predict two datasets: `xgb_x_test` for evaluating performance, with predictions stored in `xgb_prediction`, and `xgb_x_train` for reference, with predictions saved in `xgb_train_prediction`.

These predictions are vital for assessing model accuracy, overall performance, and its ability to generalize to new, unseen data. This evaluation process is essential for determining the model's effectiveness and suitability for the binary classification task.



```
# Initialize the model
xgb_classifier = XGBClassifier(max_depth = 8)
```
[54]

```
# Fitting the data on the model
xgb_classifier.fit(xgb_x_train_smote, xgb_y_train_smote)
```
[55]

...

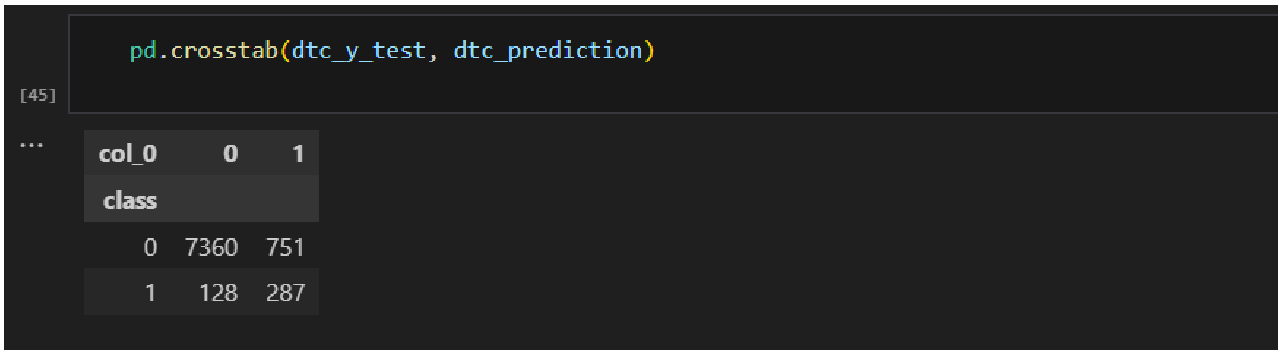
XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
# Get the predictions
xgb_prediction = xgb_classifier.predict(xgb_x_test)
xgb_train_prediction = xgb_classifier.predict(xgb_x_train)
```

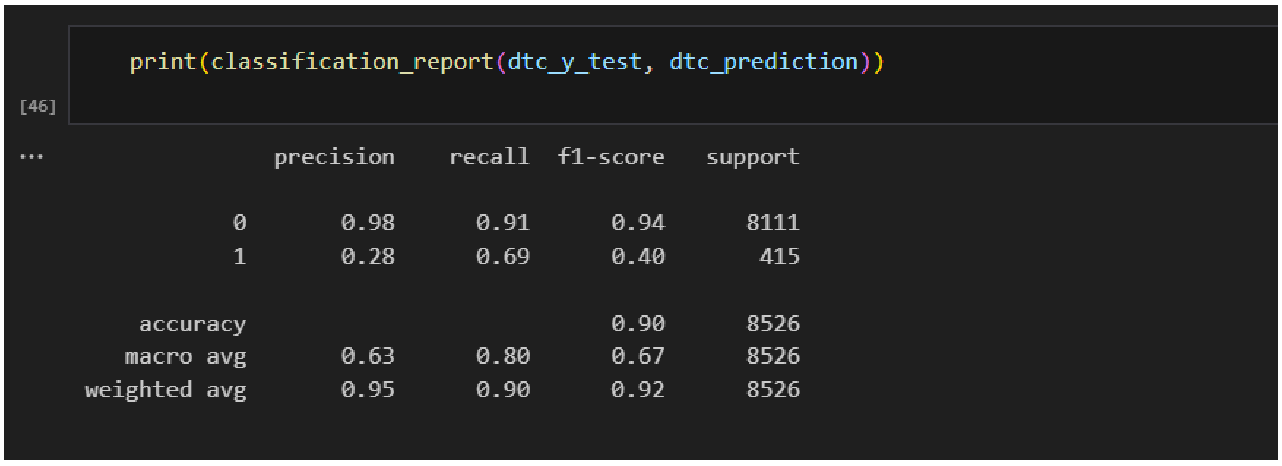
# PERFORMANCE TESTING

*Decision Tree Classifier*



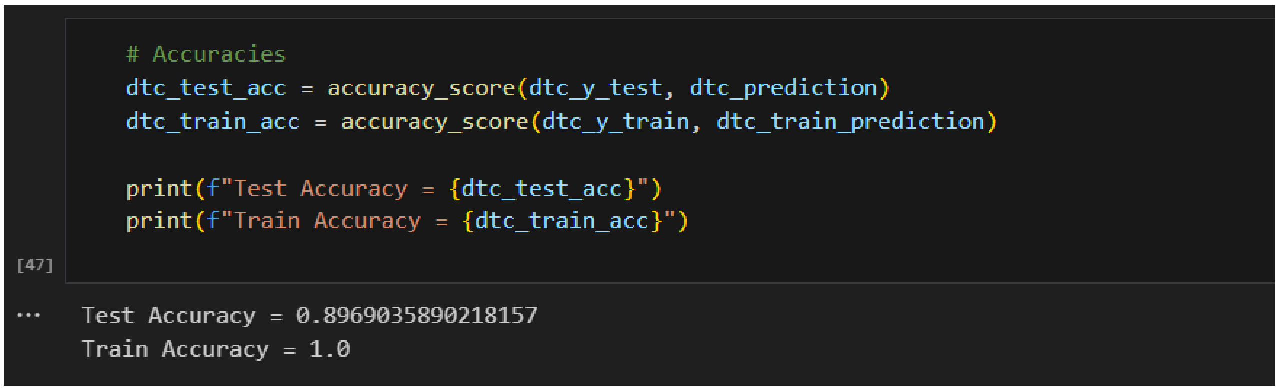
```
[45]
pd.crosstab(dtc_y_test, dtc_prediction)
```

| col_0 | 0 | 1 |
|---|---|---|
| class | | |
| 0 | 7360 | 751 |
| 1 | 128 | 287 |



```
[46]
print(classification_report(dtc_y_test, dtc_prediction))
```

```
              precision    recall  f1-score   support

           0       0.98      0.91      0.94      8111
           1       0.28      0.69      0.40       415

    accuracy                           0.90      8526
   macro avg       0.63      0.80      0.67      8526
weighted avg       0.95      0.90      0.92      8526
```

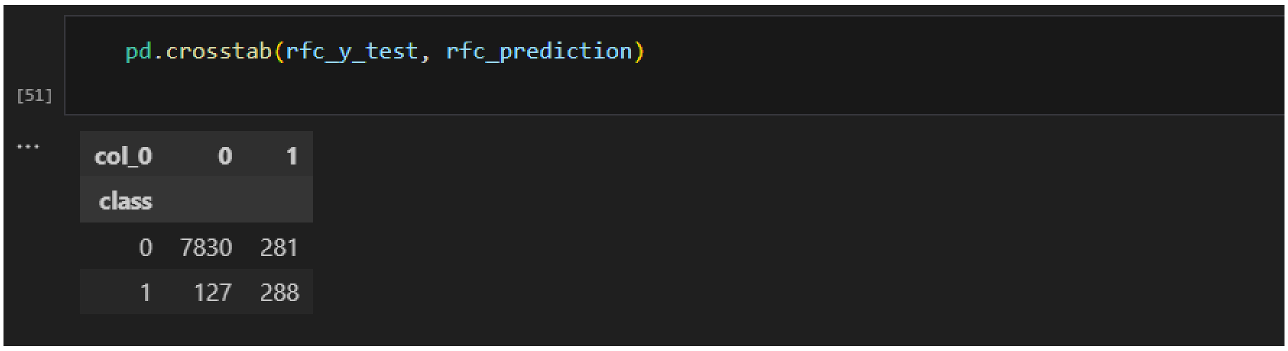


```
[47]
# Accuracies
dtc_test_acc = accuracy_score(dtc_y_test, dtc_prediction)
dtc_train_acc = accuracy_score(dtc_y_train, dtc_train_prediction)

print(f"Test Accuracy = {dtc_test_acc}")
print(f"Train Accuracy = {dtc_train_acc}")
```

```
Test Accuracy = 0.8969035890218157
Train Accuracy = 1.0
```

Back to Index

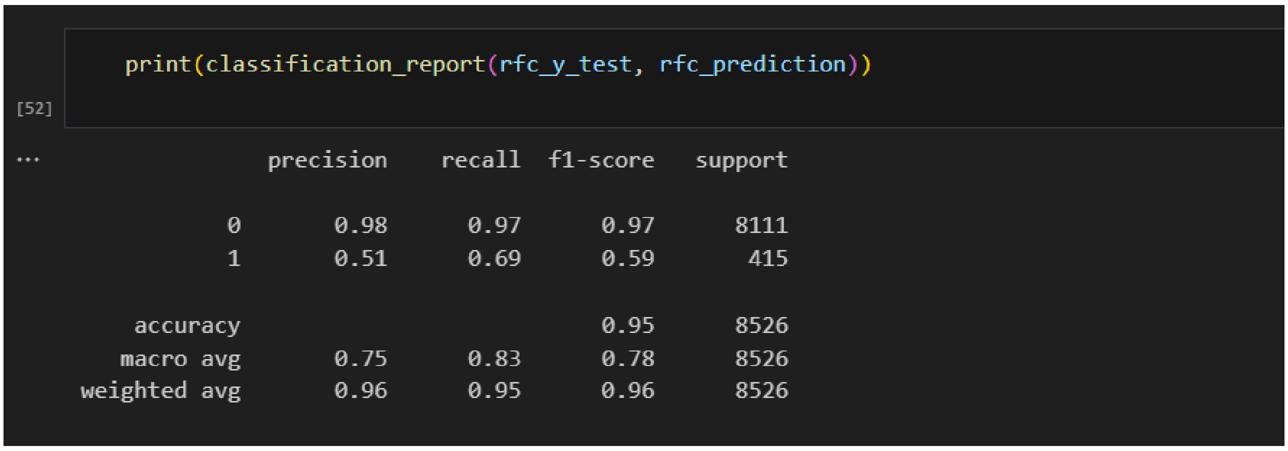
*Random Forest Classifier*



```
pd.crosstab(rfc_y_test, rfc_prediction)
```

[51]

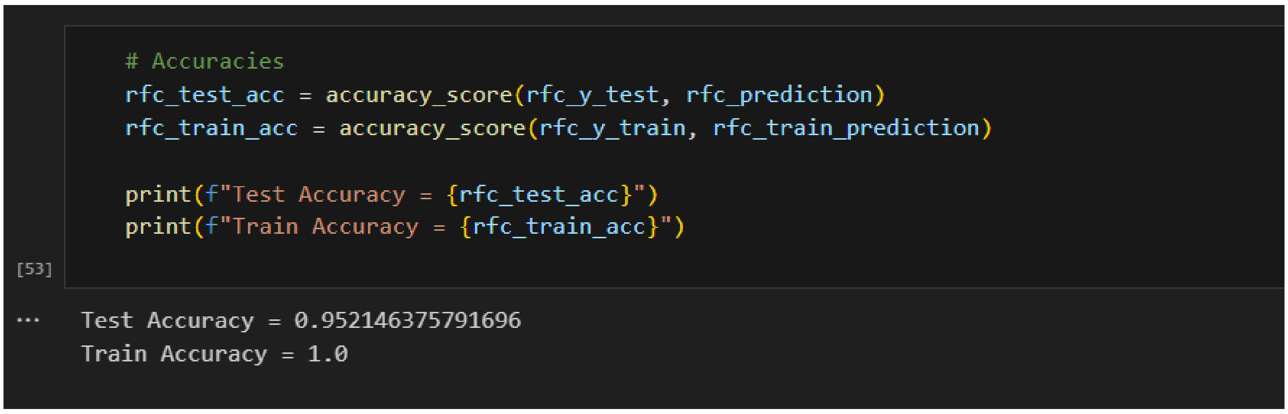
| col_0 | 0 | 1 |
|---|---|---|
| class | | |
| 0 | 7830 | 281 |
| 1 | 127 | 288 |



```
print(classification_report(rfc_y_test, rfc_prediction))
```

[52]

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97      8111
           1       0.51      0.69      0.59       415

    accuracy                           0.95      8526
   macro avg       0.75      0.83      0.78      8526
weighted avg       0.96      0.95      0.96      8526
```



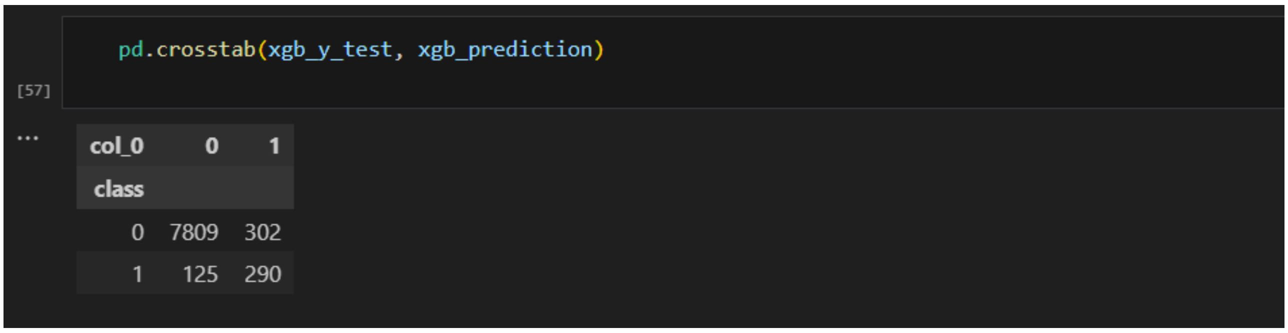
```
# Accuracies
rfc_test_acc = accuracy_score(rfc_y_test, rfc_prediction)
rfc_train_acc = accuracy_score(rfc_y_train, rfc_train_prediction)

print(f"Test Accuracy = {rfc_test_acc}")
print(f"Train Accuracy = {rfc_train_acc}")
```

[53]

```
Test Accuracy = 0.952146375791696
Train Accuracy = 1.0
```

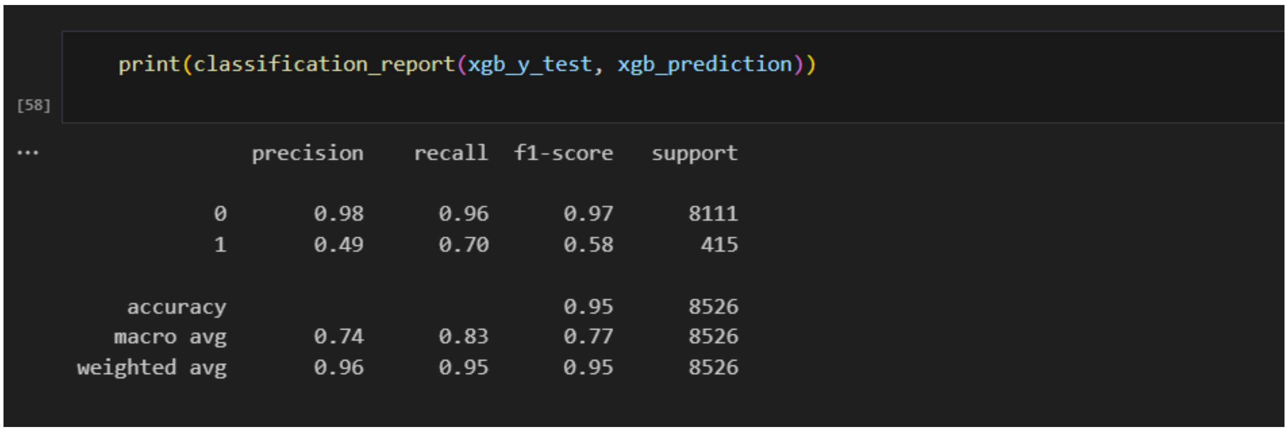
Back to Index

*XGB Classifier*



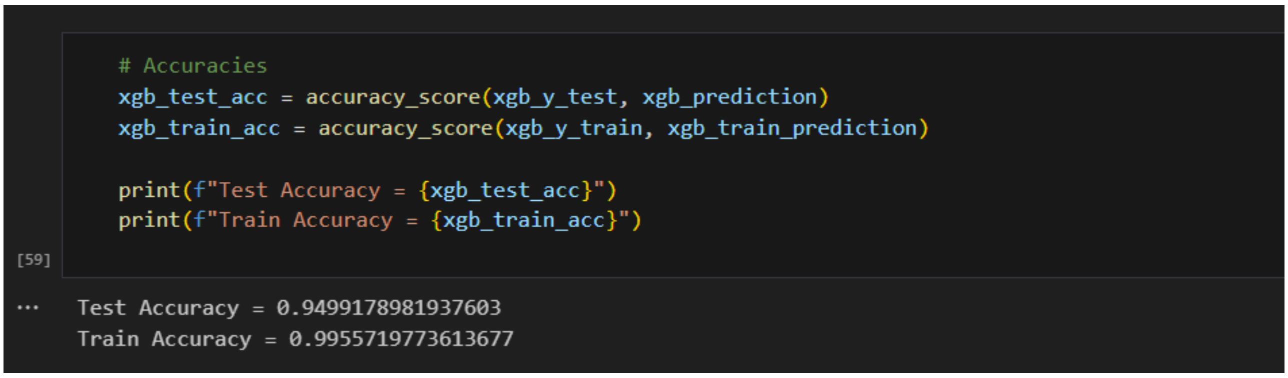
```
[57]
pd.crosstab(xgb_y_test, xgb_prediction)
```

| col_0 | 0 | 1 |
|---|---|---|
| class | | |
| 0 | 7809 | 302 |
| 1 | 125 | 290 |



```
[58]
print(classification_report(xgb_y_test, xgb_prediction))
```

```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      8111
           1       0.49      0.70      0.58       415

    accuracy                           0.95      8526
   macro avg       0.74      0.83      0.77      8526
weighted avg       0.96      0.95      0.95      8526
```

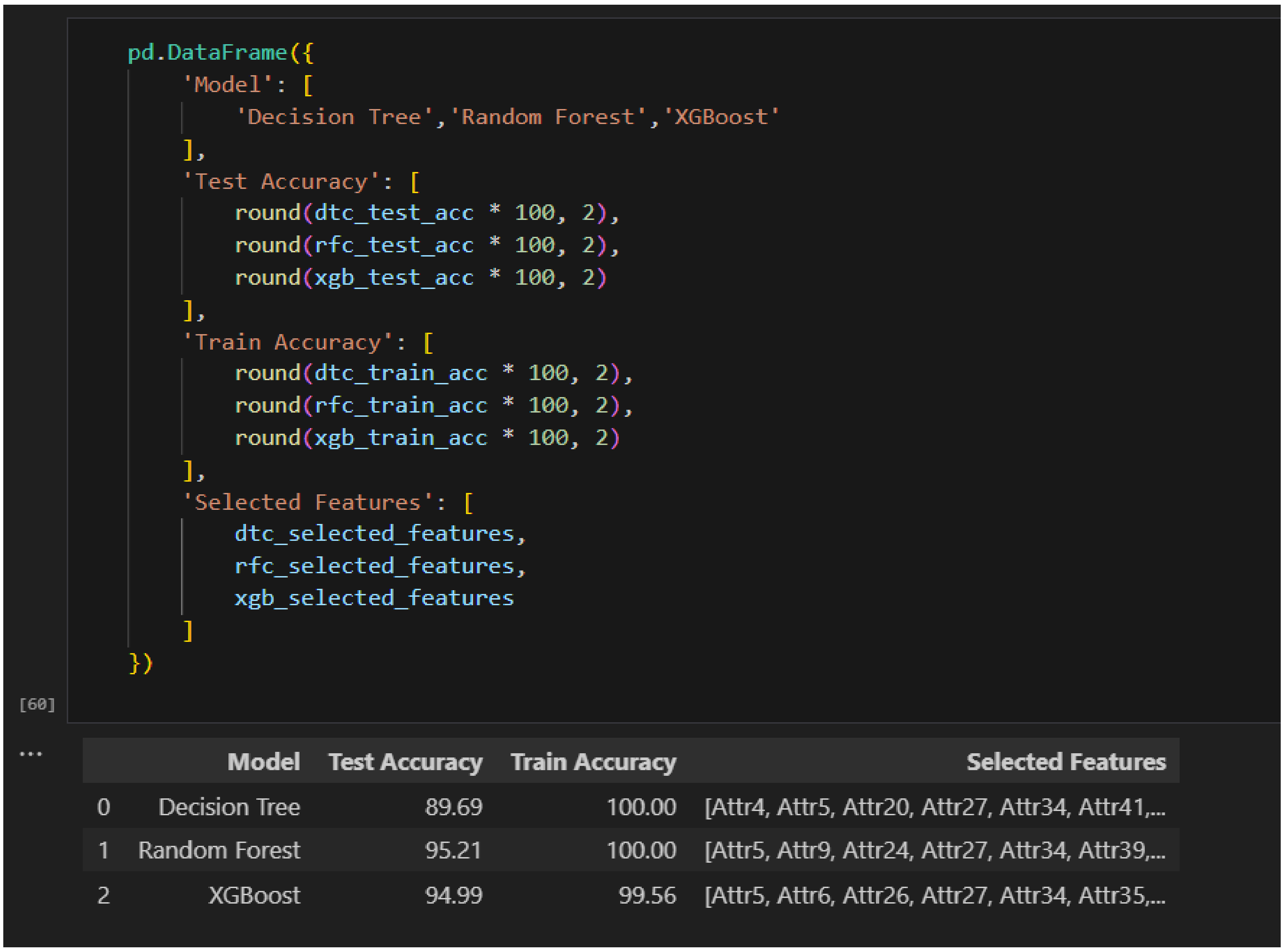


```
# Accuracies
xgb_test_acc = accuracy_score(xgb_y_test, xgb_prediction)
xgb_train_acc = accuracy_score(xgb_y_train, xgb_train_prediction)

print(f"Test Accuracy = {xgb_test_acc}")
print(f"Train Accuracy = {xgb_train_acc}")
[59]
```

```
Test Accuracy = 0.9499178981937603
Train Accuracy = 0.9955719773613677
```

Back to Index

## Comparison



```
pd.DataFrame({
    'Model': [
        'Decision Tree','Random Forest','XGBoost'
    ],
    'Test Accuracy': [
        round(dtc_test_acc * 100, 2),
        round(rfc_test_acc * 100, 2),
        round(xgb_test_acc * 100, 2)
    ],
    'Train Accuracy': [
        round(dtc_train_acc * 100, 2),
        round(rfc_train_acc * 100, 2),
        round(xgb_train_acc * 100, 2)
    ],
    'Selected Features': [
        dtc_selected_features,
        rfc_selected_features,
        xgb_selected_features
    ]
})
```

[60]

```
...
           Model  Test Accuracy  Train Accuracy                                  Selected Features
0  Decision Tree          89.69          100.00  [Attr4, Attr5, Attr20, Attr27, Attr34, Attr41,...
1  Random Forest          95.21          100.00  [Attr5, Attr9, Attr24, Attr27, Attr34, Attr39,...
2        XGBoost          94.99           99.56  [Attr5, Attr6, Attr26, Attr27, Attr34, Attr35,...
```

Following an extensive evaluation of three classifiers – Decision Tree Classifier (DTC), Random Forest Classifier (RFC), and Extreme Gradient Boosting Classifier (XGBoost) – it's evident that these models exhibit varying performance levels on both the test and training datasets. DTC showcases solid performance with an accuracy of 90% on the test data and perfect accuracy on the training data. RFC further excels with an impressive test accuracy of 95% and complete training data accuracy. Particularly noteworthy, the XGBoost Classifier demonstrates outstanding accuracy, achieving nearly ideal performance with 94.99% on the test data and a substantial 99.56% on the training data, indicating strong generalization capabilities.

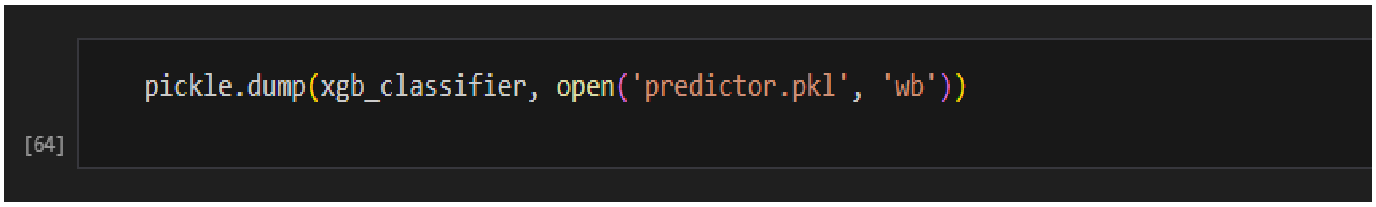
In light of these findings, the Extreme Gradient Boosting Classifier emerges as the most promising choice. Its ability to maintain high test accuracy while avoiding overfitting, as evident in the training data, highlights its robustness and suitability for this binary classification task. Consequently, we conclude that the Extreme Gradient Boosting Classifier is the most effective and reliable model among the evaluated classifiers, providing superior results for this specific problem.

Back to Index

# MODEL DEPLOYMENT

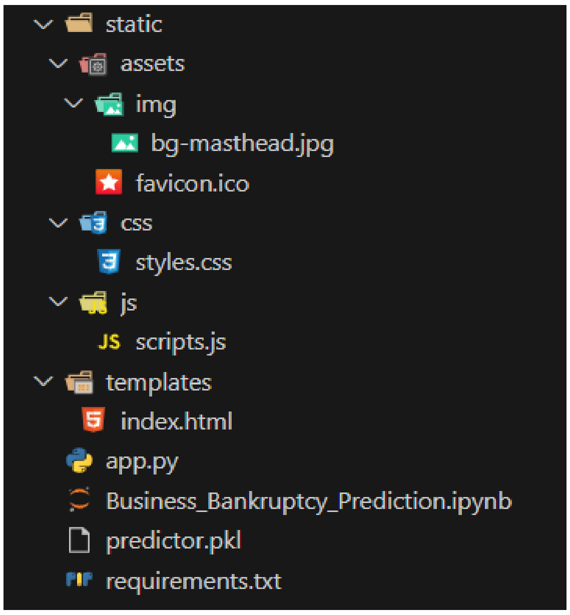
## Exporting the Model

Serializes the XGBoost Classifier model into 'predictor.pkl' for future use, allowing predictions and analysis without retraining. It uses `pickle.dump` in binary write mode.



```
[64]
    pickle.dump(xgb_classifier, open('predictor.pkl', 'wb'))
```

## Web Application



```
static
  assets
    img
      bg-masthead.jpg
    favicon.ico
  css
    styles.css
  js
    scripts.js
templates
  index.html
app.py
Business_Bankruptcy_Prediction.ipynb
predictor.pkl
requirements.txt
```

The project structure comprises essential directories and files that were used to build this.

In the "static" directory, "assets" holds image resources like "bg-masthead.jpg" and the "favicon.ico" for tab representation.

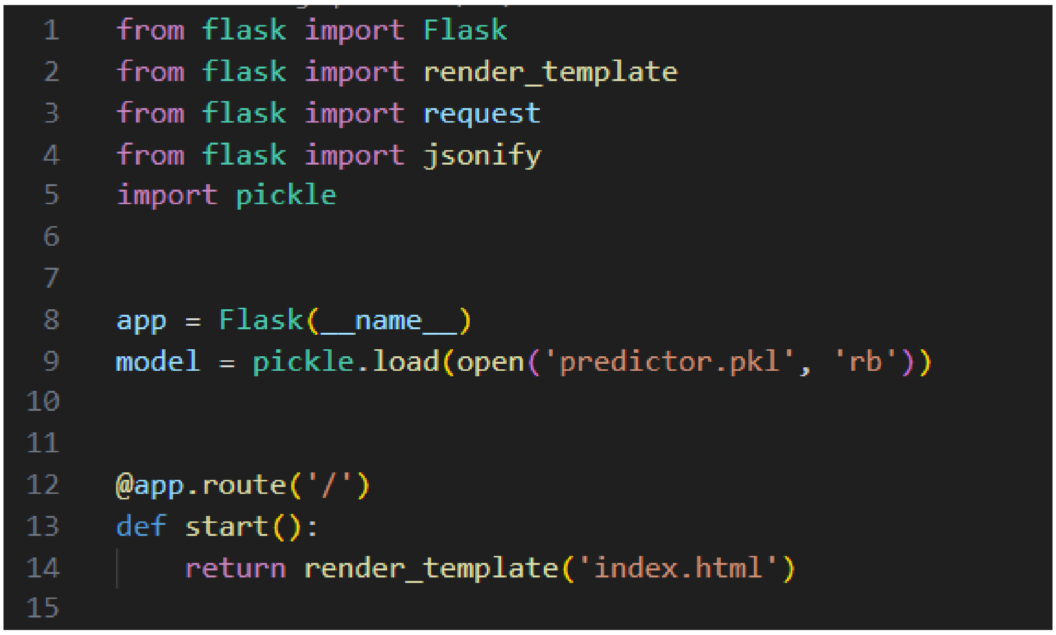
The "css" folder contains "styles.css" for styling, while the "js" folder houses "scripts.js" for client-side scripting. "templates" contains the primary HTML template, "index.html." "app.py" connects all elements and manages the Flask web application.

"predictor.pkl" stores the serialized machine learning model for predictions.

"Business_Bankruptcy_Prediction.ipynb" is the Jupyter Notebook with code and documentation. "requirements.txt" lists project dependencies for smooth functionality.

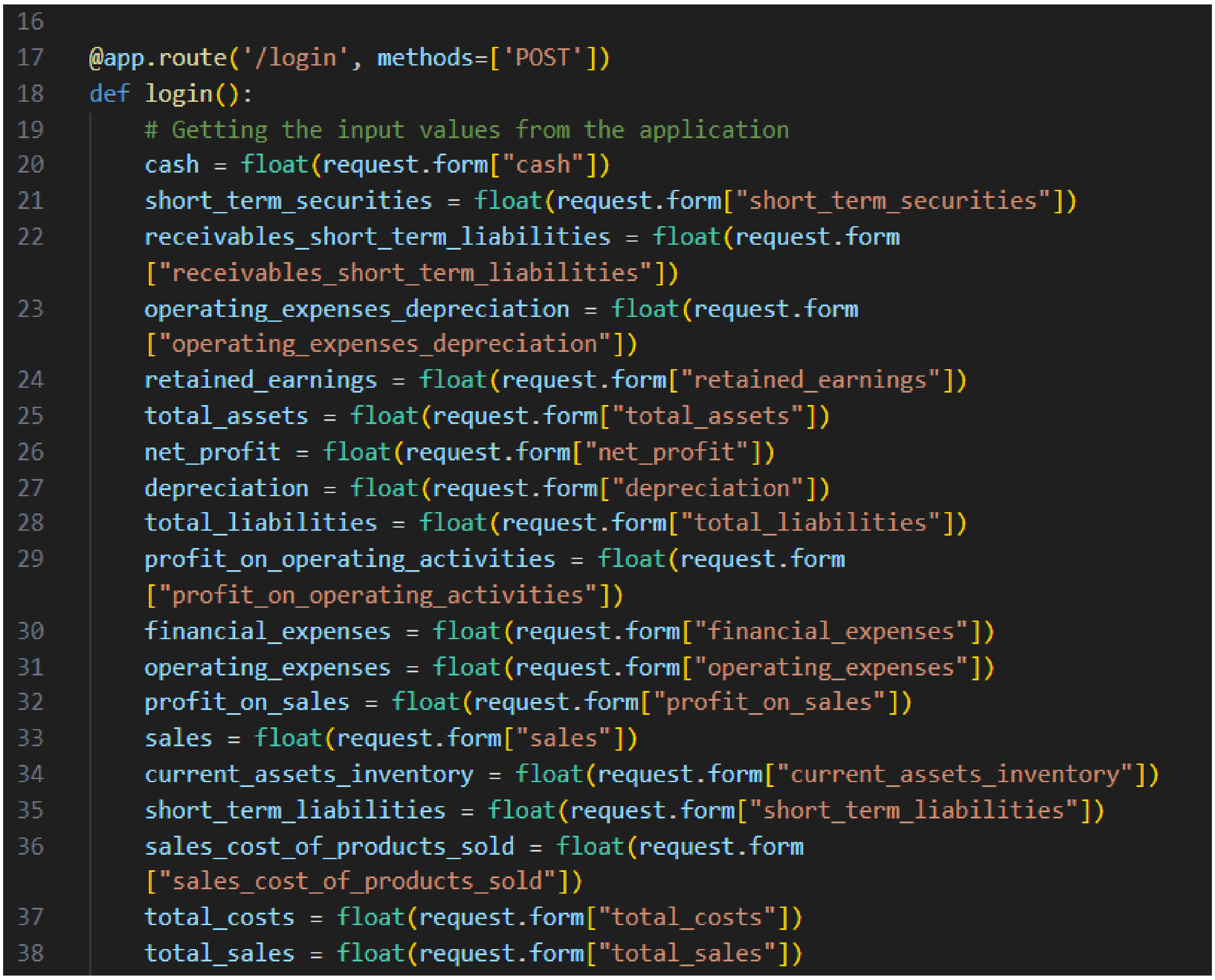
## Flask App

This Flask web app loads a machine learning model from 'predictor.pkl' and provides a single route for user interaction via the 'index.html' template.



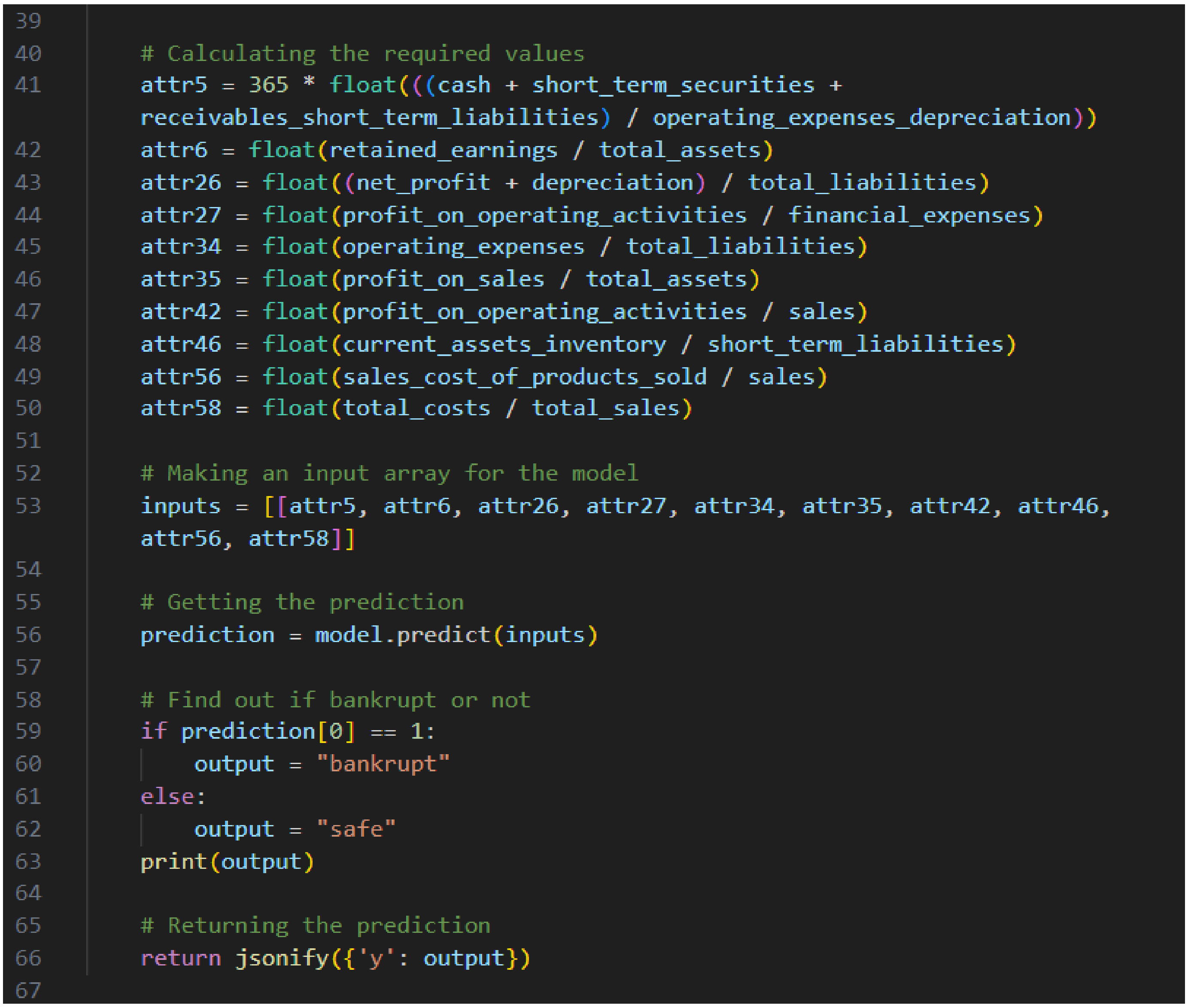
```python
from flask import Flask
from flask import render_template
from flask import request
from flask import jsonify
import pickle


app = Flask(__name__)
model = pickle.load(open('predictor.pkl', 'rb'))


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

```

The Flask route facilitates the interaction between the user interface and the machine learning model, enabling real-time bankruptcy predictions for companies based on the provided financial data.



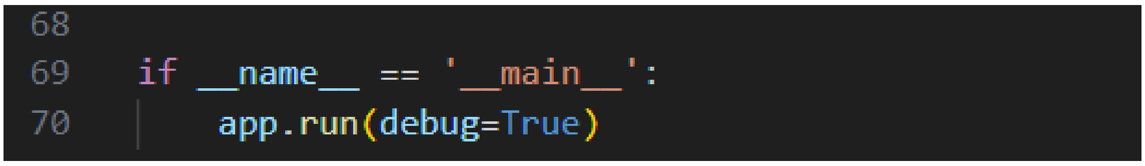
```python

@app.route('/login', methods=['POST'])
def login():
    # Getting the input values from the application
    cash = float(request.form["cash"])
    short_term_securities = float(request.form["short_term_securities"])
    receivables_short_term_liabilities = float(request.form["receivables_short_term_liabilities"])
    operating_expenses_depreciation = float(request.form["operating_expenses_depreciation"])
    retained_earnings = float(request.form["retained_earnings"])
    total_assets = float(request.form["total_assets"])
    net_profit = float(request.form["net_profit"])
    depreciation = float(request.form["depreciation"])
    total_liabilities = float(request.form["total_liabilities"])
    profit_on_operating_activities = float(request.form["profit_on_operating_activities"])
    financial_expenses = float(request.form["financial_expenses"])
    operating_expenses = float(request.form["operating_expenses"])
    profit_on_sales = float(request.form["profit_on_sales"])
    sales = float(request.form["sales"])
    current_assets_inventory = float(request.form["current_assets_inventory"])
    short_term_liabilities = float(request.form["short_term_liabilities"])
    sales_cost_of_products_sold = float(request.form["sales_cost_of_products_sold"])
    total_costs = float(request.form["total_costs"])
    total_sales = float(request.form["total_sales"])
```

Back to Index



```python

    # Calculating the required values
    attr5 = 365 * float((cash + short_term_securities + receivables_short_term_liabilities) / operating_expenses_depreciation))
    attr6 = float(retained_earnings / total_assets)
    attr26 = float((net_profit + depreciation) / total_liabilities)
    attr27 = float(profit_on_operating_activities / financial_expenses)
    attr34 = float(operating_expenses / total_liabilities)
    attr35 = float(profit_on_sales / total_assets)
    attr42 = float(profit_on_operating_activities / sales)
    attr46 = float(current_assets_inventory / short_term_liabilities)
    attr56 = float(sales_cost_of_products_sold / sales)
    attr58 = float(total_costs / total_sales)

    # Making an input array for the model
    inputs = [[attr5, attr6, attr26, attr27, attr34, attr35, attr42, attr46, attr56, attr58]]

    # Getting the prediction
    prediction = model.predict(inputs)

    # Find out if bankrupt or not
    if prediction[0] == 1:
        output = "bankrupt"
    else:
        output = "safe"
    print(output)

    # Returning the prediction
    return jsonify({'y': output})

```

The `/login` Flask route handles POST requests from the predictive application's user interface. It retrieves financial attribute data from the interface, calculates derived attributes, prepares input data for the model, predicts bankruptcy likelihood, and sends the result in JSON format while printing it for reference. The code also enables debugging mode for local testing.

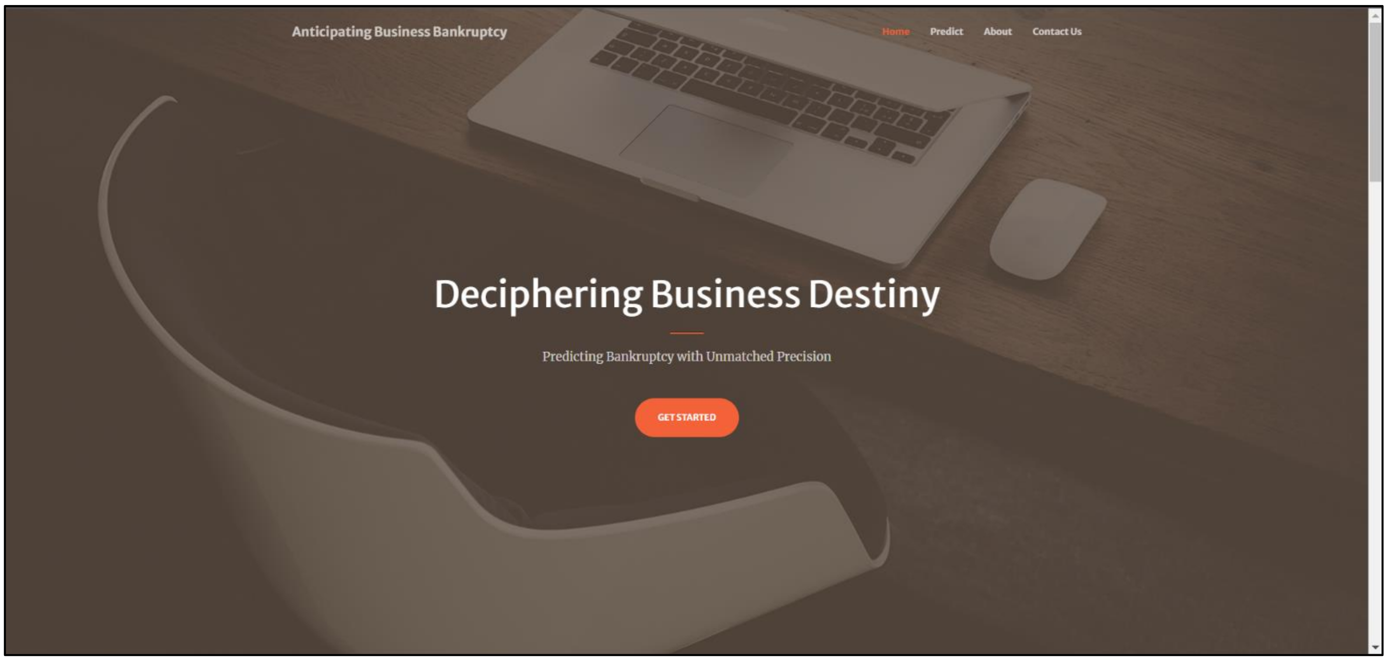


```python

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

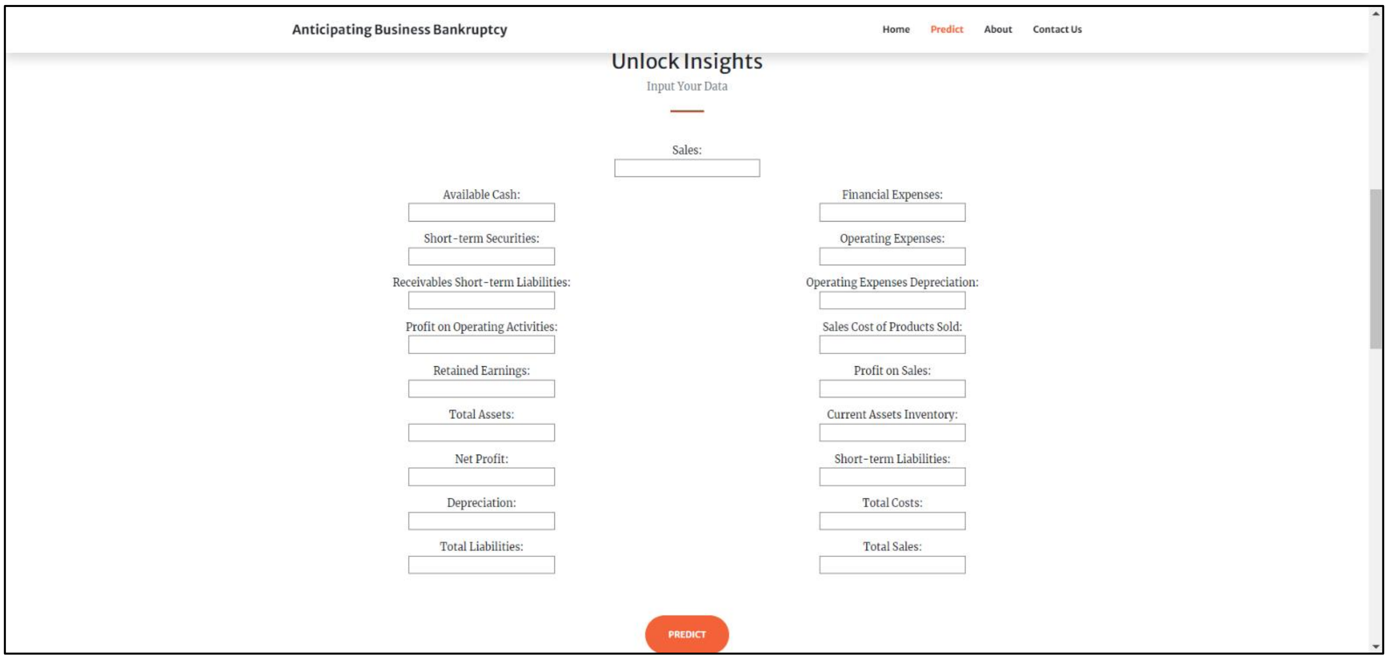
Back to Index

# PROJECT DEMONSTRATION

The landing page or the homepage of the website:


Anticipating Business Bankruptcy
Home
Predict
About
Contact Us
Deciphering Business Destiny
Predicting Bankruptcy with Unmatched Precision
GET STARTED

The part of the website that allows the user to enter the values to generate the prediction:



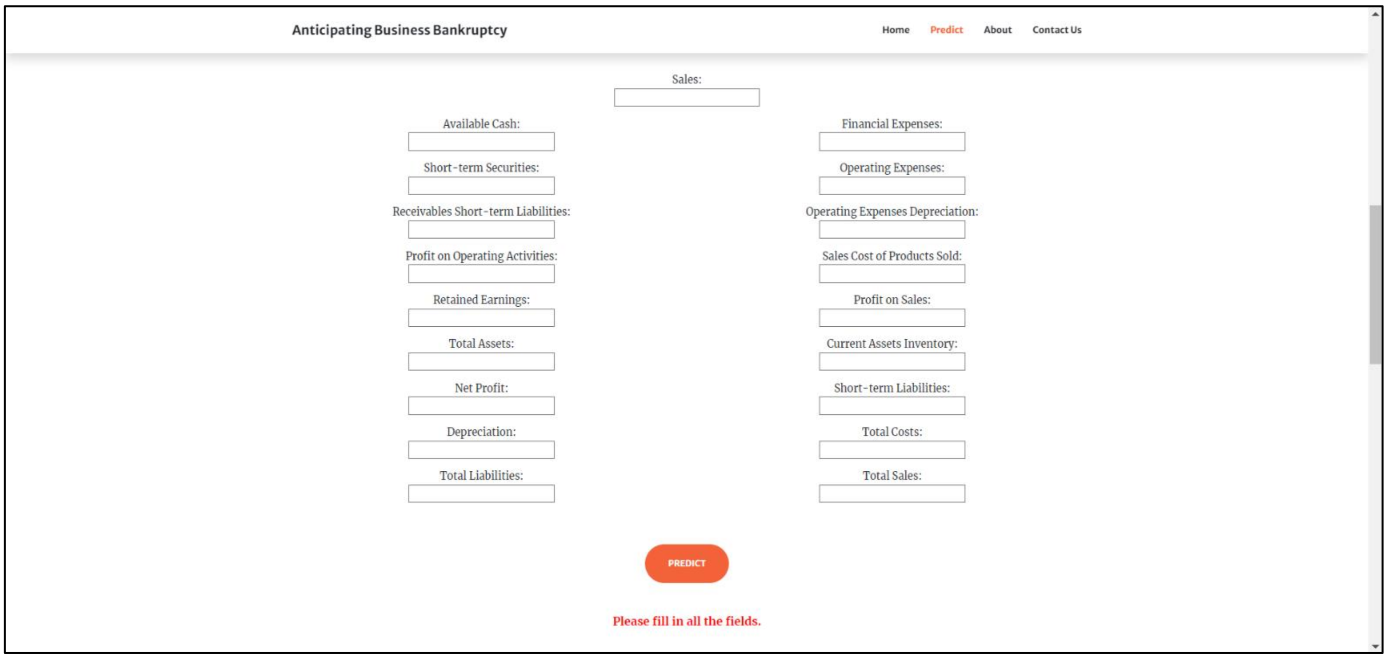
Unlock Insights

Input Your Data

Sales:

| | |
|---|---|
| Available Cash: | Financial Expenses: |
| Short-term Securities: | Operating Expenses: |
| Receivables Short-term Liabilities: | Operating Expenses Depreciation: |
| Profit on Operating Activities: | Sales Cost of Products Sold: |
| Retained Earnings: | Profit on Sales: |
| Total Assets: | Current Assets Inventory: |
| Net Profit: | Short-term Liabilities: |
| Depreciation: | Total Costs: |
| Total Liabilities: | Total Sales: |

*__Prediction Case 1__*: If any of the field is left empty, the site throws an error "Please fill in all the fields."

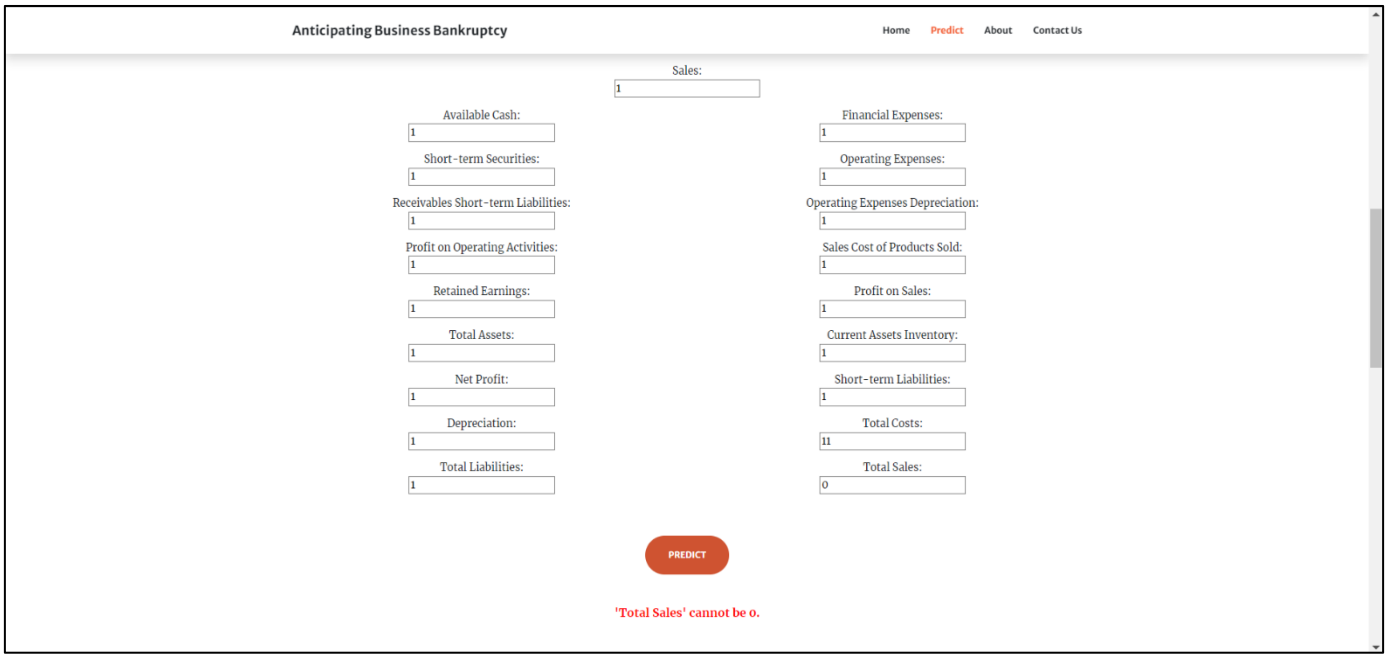


Anticipating Business Bankruptcy

| | |
|---|---|
| Sales: | |
| Available Cash: | Financial Expenses: |
| Short-term Securities: | Operating Expenses: |
| Receivables Short-term Liabilities: | Operating Expenses Depreciation: |
| Profit on Operating Activities: | Sales Cost of Products Sold: |
| Retained Earnings: | Profit on Sales: |
| Total Assets: | Current Assets Inventory: |
| Net Profit: | Short-term Liabilities: |
| Depreciation: | Total Costs: |
| Total Liabilities: | Total Sales: |

Please fill in all the fields.

*__Prediction Case 2__*: Since the values are used for calculations, few of them are a part of the denominator. Hence, if those values are equal to 0, then the site throws the error "'{Field}' cannot be 0."



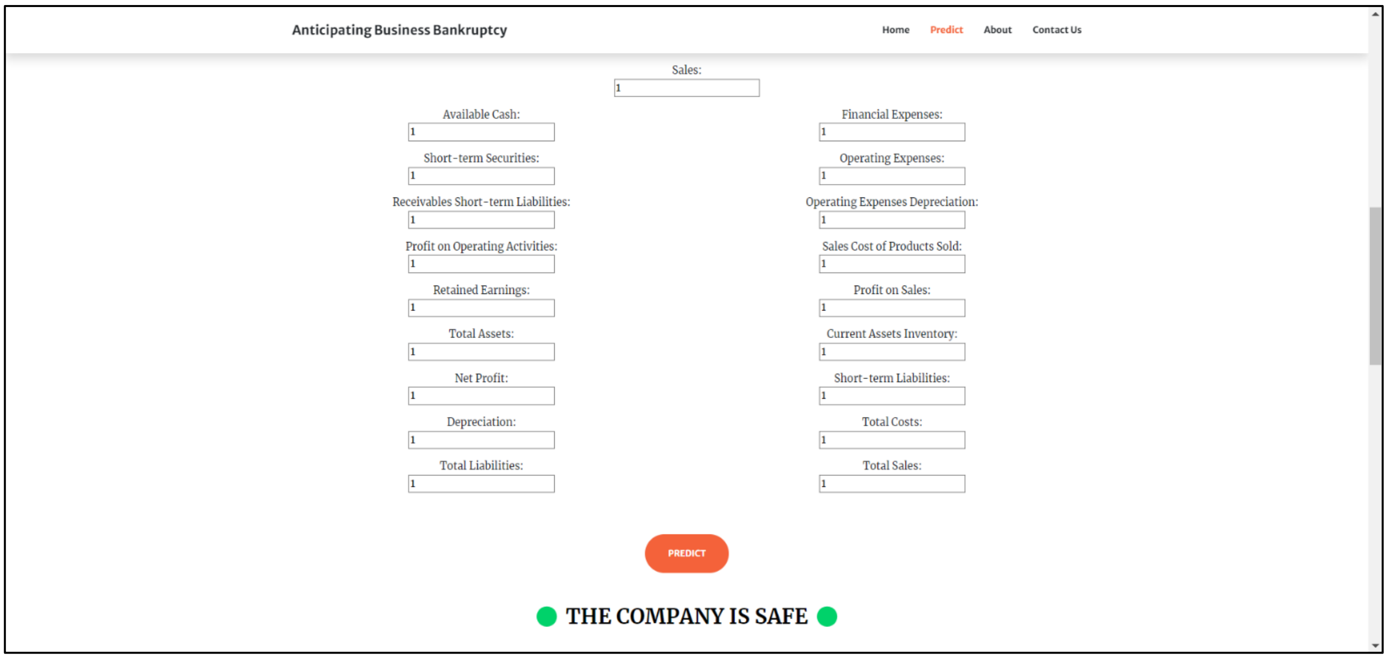
Anticipating Business Bankruptcy

| | | | |
|---|---|---|---|
| Sales: | 1 | | |
| Available Cash: | 1 | Financial Expenses: | 1 |
| Short-term Securities: | 1 | Operating Expenses: | 1 |
| Receivables Short-term Liabilities: | 1 | Operating Expenses Depreciation: | 1 |
| Profit on Operating Activities: | 1 | Sales Cost of Products Sold: | 1 |
| Retained Earnings: | 1 | Profit on Sales: | 1 |
| Total Assets: | 1 | Current Assets Inventory: | 1 |
| Net Profit: | 1 | Short-term Liabilities: | 1 |
| Depreciation: | 1 | Total Costs: | 11 |
| Total Liabilities: | 1 | Total Sales: | 0 |

'Total Sales' cannot be 0.

Back to Index

*Prediction Case 3*: When all the values are perfectly entered, the site gives a prediction. In this case, it says that the company is safe.


Anticipating Business Bankruptcy

Home Predict About Contact Us

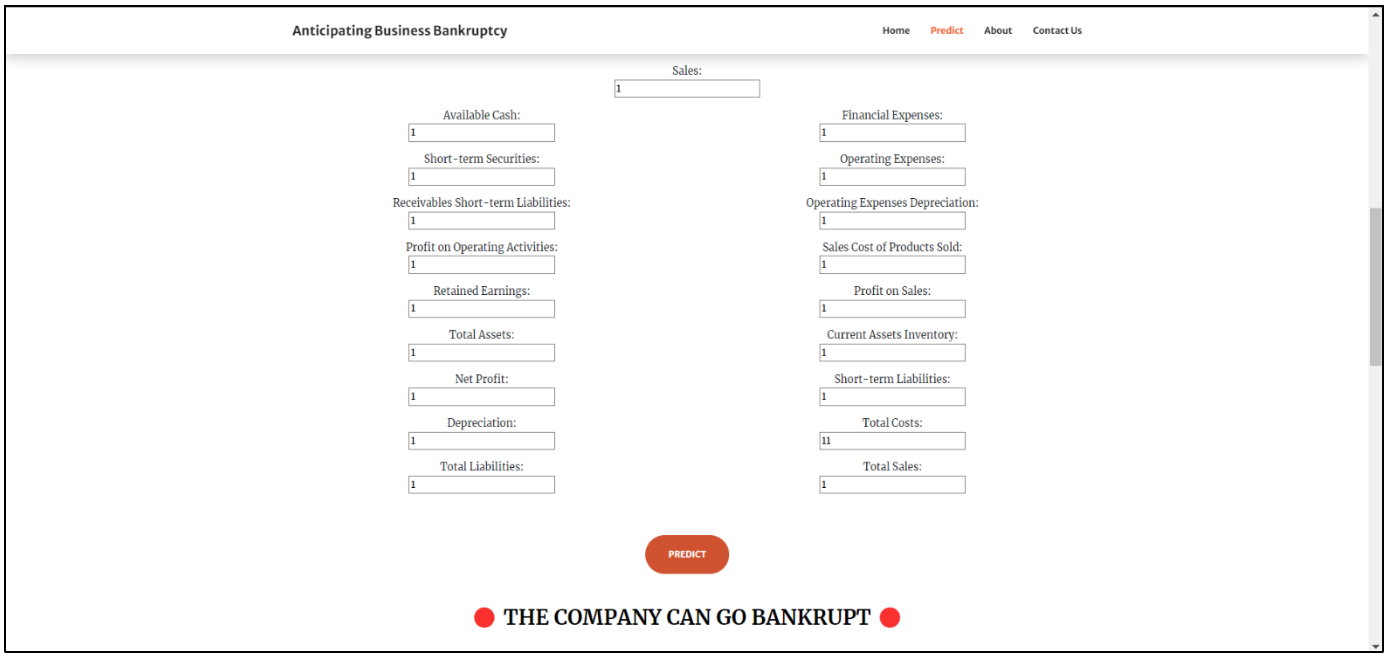
Sales: 1

| | | | |
|---|---|---|---|
| Available Cash: | 1 | Financial Expenses: | 1 |
| Short-term Securities: | 1 | Operating Expenses: | 1 |
| Receivables Short-term Liabilities: | 1 | Operating Expenses Depreciation: | 1 |
| Profit on Operating Activities: | 1 | Sales Cost of Products Sold: | 1 |
| Retained Earnings: | 1 | Profit on Sales: | 1 |
| Total Assets: | 1 | Current Assets Inventory: | 1 |
| Net Profit: | 1 | Short-term Liabilities: | 1 |
| Depreciation: | 1 | Total Costs: | 1 |
| Total Liabilities: | 1 | Total Sales: | 1 |

PREDICT

● THE COMPANY IS SAFE ●

*Prediction Case 4*: When all the values are perfectly entered, the site gives a prediction. In this case, it says that the company is not safe and may go bankrupt.


Anticipating Business Bankruptcy

Home Predict About Contact Us

Sales: 1

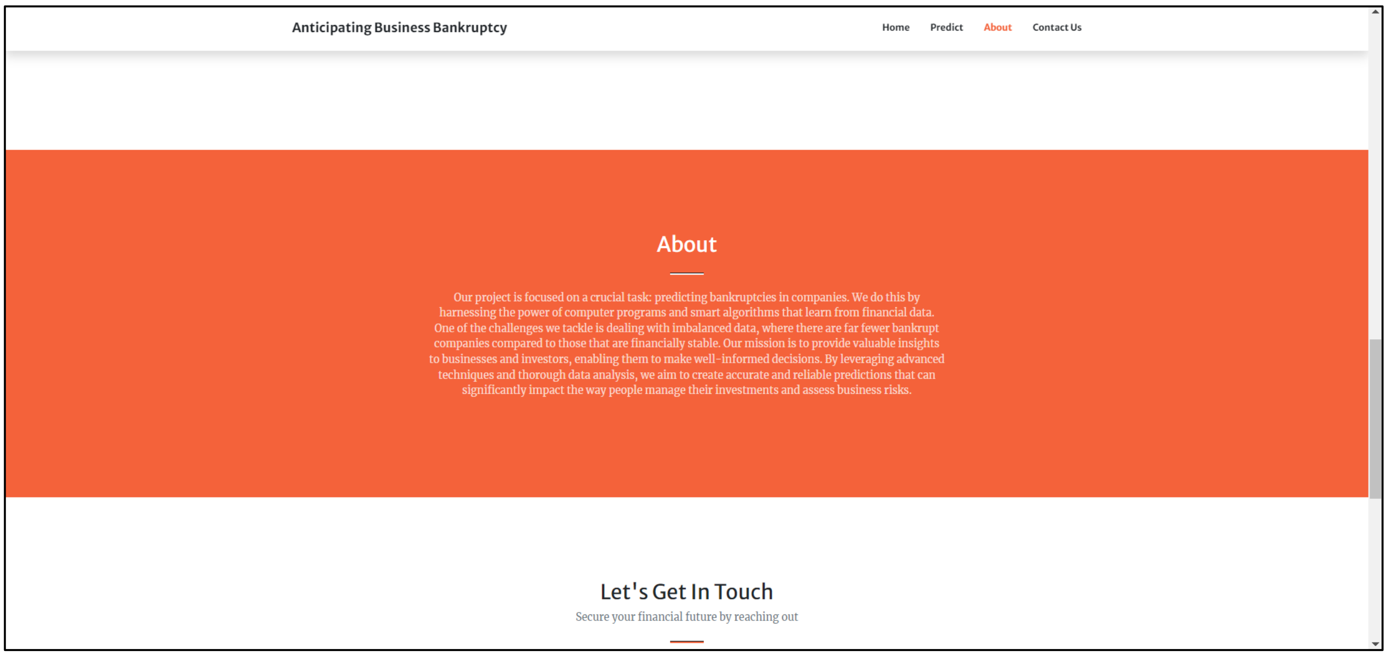
| | | | |
|---|---|---|---|
| Available Cash: | 1 | Financial Expenses: | 1 |
| Short-term Securities: | 1 | Operating Expenses: | 1 |
| Receivables Short-term Liabilities: | 1 | Operating Expenses Depreciation: | 1 |
| Profit on Operating Activities: | 1 | Sales Cost of Products Sold: | 1 |
| Retained Earnings: | 1 | Profit on Sales: | 1 |
| Total Assets: | 1 | Current Assets Inventory: | 1 |
| Net Profit: | 1 | Short-term Liabilities: | 1 |
| Depreciation: | 1 | Total Costs: | 11 |
| Total Liabilities: | 1 | Total Sales: | 1 |

PREDICT

● THE COMPANY CAN GO BANKRUPT ●

Back to Index

The 'About' section of the website:



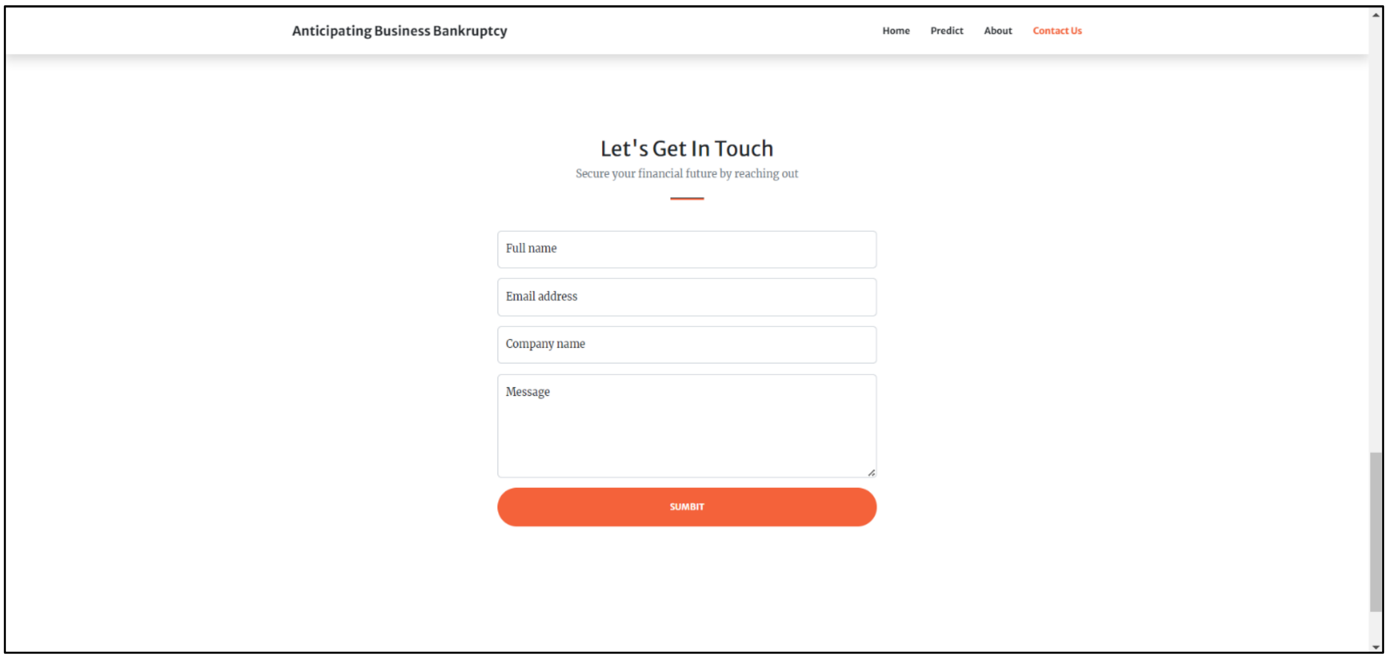
About

Our project is focused on a crucial task: predicting bankruptcies in companies. We do this by harnessing the power of computer programs and smart algorithms that learn from financial data. One of the challenges we tackle is dealing with imbalanced data, where there are far fewer bankrupt companies compared to those that are financially stable. Our mission is to provide valuable insights to businesses and investors, enabling them to make well-informed decisions. By leveraging advanced techniques and thorough data analysis, we aim to create accurate and reliable predictions that can significantly impact the way people manage their investments and assess business risks.

Let's Get In Touch

Secure your financial future by reaching out

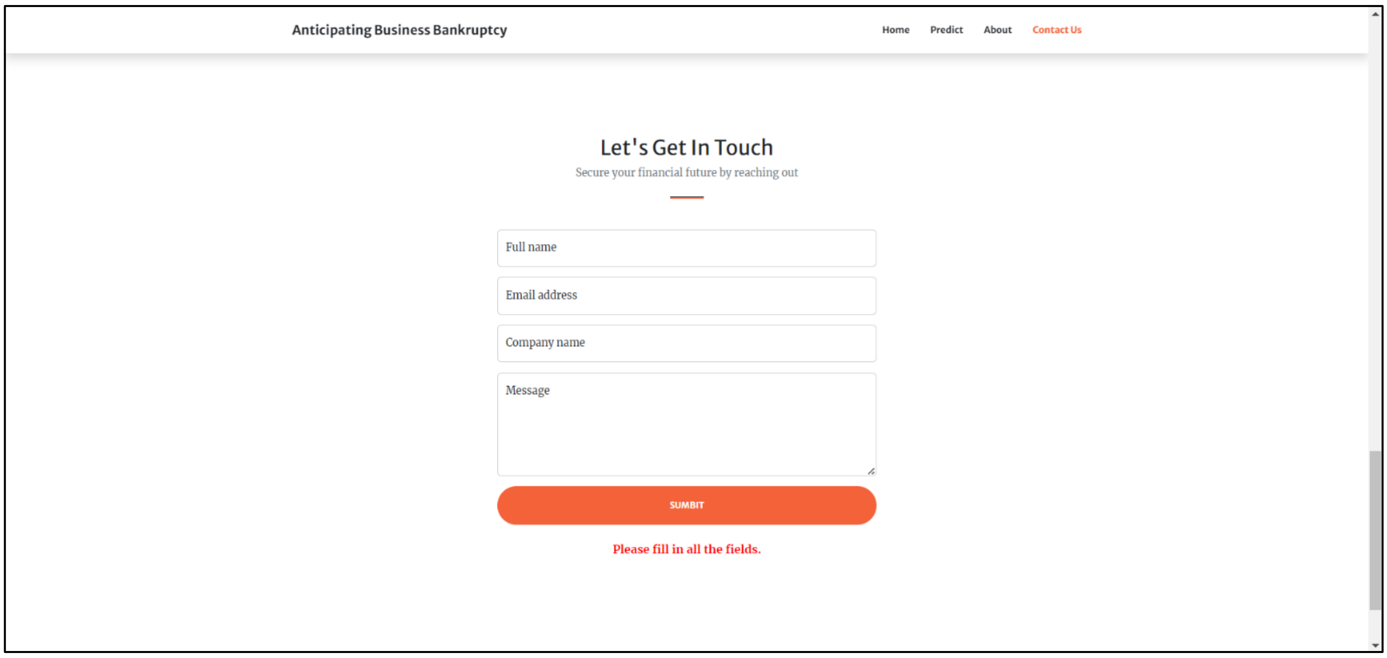
The 'Contact Us' section of the website:



Let's Get In Touch

Secure your financial future by reaching out

*Contact Us Case 1*: If any of the field is left empty, the site throws an error "Please fill in all the fields."



Let's Get In Touch

Secure your financial future by reaching out

Full name

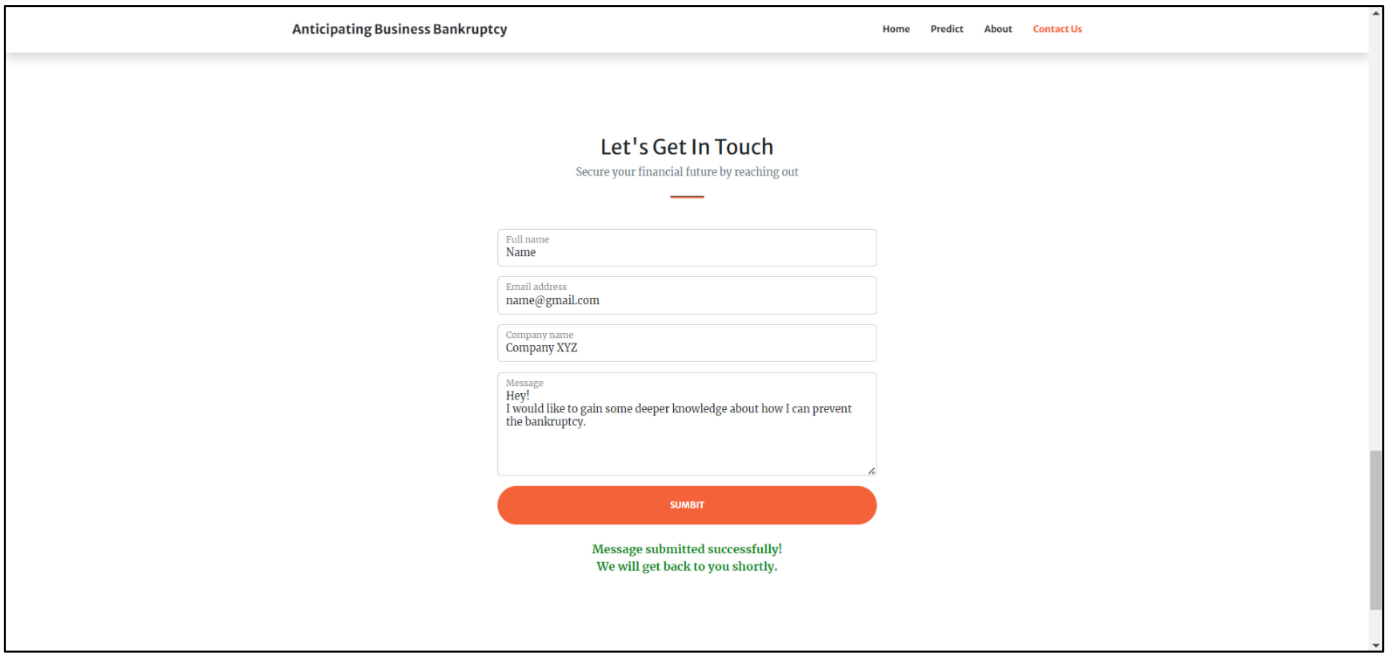
Email address

Company name

Message

Please fill in all the fields.

*Contact Us Case 2*: When all the values are perfectly entered, the site return the message "Message submitted successfully! We will get back to you shortly."



Let's Get In Touch

Secure your financial future by reaching out

Full name
Name

Email address
name@gmail.com

Company name
Company XYZ

Message
Hey!
I would like to gain some deeper knowledge about how I can prevent the bankruptcy.

Message submitted successfully!
We will get back to you shortly.

Back to Index