```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc
from imblearn.under_sampling import NearMiss
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#read data from dataset
bank_data = pd.read_csv("/content/drive/MyDrive/FYP2025/res_data.csv")

bank_data.head()
```

	ORGNR	Year	ZipCode	Municipality	Bankrupt	Business_Category	Size_Category	OperatingProfit	F
0	5560024712	1998	23163	1287.0	0	30	2	-558.0	
1	5560024712	1999	23163	1287.0	0	30	2	-634.0	
2	5560024712	2000	23163	1287.0	0	30	2	3809.0	
3	5560024712	2001	23163	1287.0	0	30	2	-1147.0	
4	5560024712	2002	23163	1287.0	0	30	2	-1250.0	

5 rows × 102 columns

```
#0 bankcrpt no
#1 bancrpt yes
print(bank_data['Bankrupt'].value_counts())

```

```
Bankrupt
0    90966
1      335
Name: count, dtype: int64
```

```
#Visual representation of dataset
bank_data.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91301 entries, 0 to 91300
Columns: 102 entries, ORGNR to Profit_Margin
dtypes: float64(68), int64(34)
memory usage: 71.1 MB
```

```
#calculate average inventry
average_inventory_series =bank_data.groupby('ORGNR')['Inventory'].transform(lambda x: (x + x.shift(-1)) /
```

```


#add average inventry as column in dataframe
bank_data['Average_Inventory'] = average_inventory_series

```

```
# Replace NaN values in 'Average_Inventory' with the previous values
bank_data['Average_Inventory'] = bank_data['Average_Inventory'].ffill()

```

```


#calculating total liabilities
bank_data['Total_Liabilities'] = bank_data['Total_Curr_Liabilities'] + bank_data['Total_Non_curr_lia']

```

```

bank_data
```

	ORGNR	Year	ZipCode	Municipality	Bankrupt	Business_Category	Size_Category	OperatingProfi
0	5560024712	1998	23163	1287.0	0	30	2	-558
1	5560024712	1999	23163	1287.0	0	30	2	-634
2	5560024712	2000	23163	1287.0	0	30	2	3809
3	5560024712	2001	23163	1287.0	0	30	2	-1147
4	5560024712	2002	23163	1287.0	0	30	2	-1250
...	...	...	...	...	...	...	...	
91296	5567417315	2009	43430	1384.0	0	30	2	134
91297	5567417315	2010	43430	1384.0	0	30	2	70
91298	5567417315	2011	43430	1384.0	0	30	2	125
91299	5567417315	2012	43430	1384.0	0	30	2	76
91300	5567417315	2013	40013	1480.0	0	30	2	26

91301 rows × 104 columns

```
#eleminating assests ROE
# Check if columns exist before dropping
if 'Equity_to_Assets' in bank_data.columns and 'ROE' in bank_data.columns:
    bank_data = bank_data.drop(['Equity_to_Assets', 'ROE'], axis=1)
else:
    # Print a message indicating which columns are not found
    missing_columns = [col for col in ['Equity_to_Assets', 'ROE'] if col not in bank_data.columns]
    print(f"Columns not found in DataFrame: {missing_columns}")
bank_data.head()
```

	ORGNR	Year	ZipCode	Municipality	Bankrupt	Business_Category	Size_Category	OperatingProfit	F
0	5560024712	1998	23163	1287.0	0	30	2	-558.0	
1	5560024712	1999	23163	1287.0	0	30	2	-634.0	
2	5560024712	2000	23163	1287.0	0	30	2	3809.0	
3	5560024712	2001	23163	1287.0	0	30	2	-1147.0	
4	5560024712	2002	23163	1287.0	0	30	2	-1250.0	

5 rows × 102 columns

```

#opreating cost
bank_data['Operating_CF'] = (bank_data['NetProfit'] + bank_data['Depreciation']) - (bank_data['Total_CurrA

```

```
#calculating liablites to asset
bank_data['Liabilities_to_Assets'] = bank_data['Total_Liabilities'] / bank_data['Total_Assets']

```

```


#calculating return of assets
bank_data['Return_on_Assets'] = bank_data['NetProfit'] / bank_data['Total_Assets']

```

```
#deb to equity
bank_data['Debt_to_Equity'] = bank_data['Total_Liabilities'] / bank_data['Total_Equity']

```

```
#profit margin and gross margin
bank_data['Profit_Margin'] = bank_data['NetProfit'] / bank_data['Net_Sales']
bank_data['Gross_Margin'] = (bank_data['Net_Sales']-bank_data['Production_Costs']) / bank_data['Net_Sales

```

```
#networth to assets
bank_data['NetWC_to_Assets'] = (bank_data['Total_CurrA'] - bank_data['Total_Curr_Liabilities']) / bank_dat

```

```
#ebit
bank_data['EBIT_to_Assets'] = bank_data['OperatingProfit'] / bank_data['Total_Assets']
bank_data['EBIT_to_Sales'] = bank_data['OperatingProfit'] /bank_data['Net_Sales']

```

```
#finanical ratio
bank_data['Current_Ratio'] = bank_data['Total_CurrA'] / bank_data['Total_Curr_Liabilities']
bank_data['Cash_Ratio'] = (bank_data['CashnBank']+bank_data['Securities_Investments']+bank_data['Total_re
bank_data['Working_Capital_Ratio'] = bank_data['Total_CurrA'] / bank_data['Total_Curr_Liabilities']

```

```
bank_data['Equity_to_Liabilities'] = bank_data['Total_Equity'] / bank_data['Total_Liabilities']
bank_data['Net_Sales_to_Assets'] = bank_data['Net_Sales'] / bank_data['Total_Assets']
bank_data['Net_Worth_to_Debt'] = (bank_data['Total_Assets'] - bank_data['Intangible FxdA'] - bank_data['To

```

```
bank_data["CF_to_debt"] = bank_data["Operating_CF"] / bank_data["Total_Liabilities"]
bank_data["CF_to_sales"] = bank_data["Operating_CF"] / bank_data["Net_Sales"]
```

```
bank_data["CF_coverage"] = bank_data["Operating_CF"] / (bank_data["Other_oper_exp"] + bank_data["Financia
bank_data["CF_to_equity"] = bank_data["Operating_CF"] / bank_data["Total_Equity"]
bank_data["CF_to_Assets"] = bank_data["Operating_CF"] / bank_data["Total_Assets"]

```

```
bank_data["RE_to_Sales"] = bank_data["Accumulated_profit"] / bank_data["Net_Sales"]
bank_data["RE_to_Assets"] = bank_data["Accumulated_profit"] / bank_data["Total_Assets"]
bank_data["RE_to_NetIncome"] = bank_data["Accumulated_profit"] / bank_data["NetProfit"]

```

```

bank_data["Inventory_Turnover"] = bank_data["Production_Costs"] / bank_data["Average_Inventory"]
bank_data["Inventory_to_Sales"] = bank_data["Inventory"] / bank_data["Net_Sales"]

```

```
 bank_data.head(7)

```

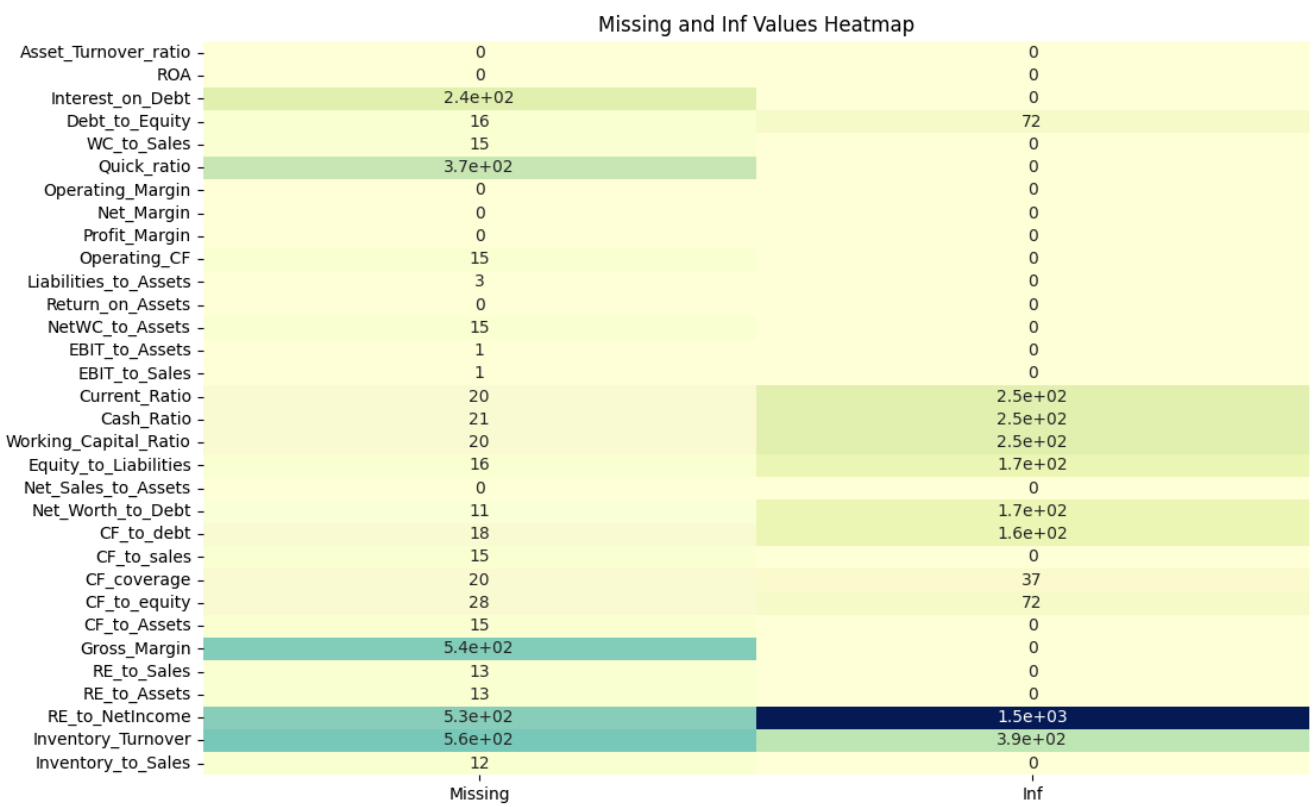
	ORGNR	Year	ZipCode	Municipality	Bankrupt	Business_Category	Size_Category	OperatingProfit	F
0	5560024712	1998	23163	1287.0	0	30	2	-558.0	
1	5560024712	1999	23163	1287.0	0	30	2	-634.0	
2	5560024712	2000	23163	1287.0	0	30	2	3809.0	
3	5560024712	2001	23163	1287.0	0	30	2	-1147.0	
4	5560024712	2002	23163	1287.0	0	30	2	-1250.0	
5	5560024712	2003	23163	1287.0	0	30	2	-1492.0	
6	5560024712	2004	23163	1287.0	0	30	2	-1024.0	

7 rows × 125 columns

```
predictor_columns = [
    'Asset_Turnover_ratio', 'ROA', 'Interest_on_Debt', 'Debt_to_Equity',
    'WC_to_Sales', 'Quick_ratio', 'Operating_Margin',
    'Net_Margin', 'Profit_Margin', 'Operating_CF',
    'Liabilities_to_Assets', 'Return_on_Assets', 'NetWC_to_Assets',
    'EBIT_to_Assets', 'EBIT_to_Sales', 'Current_Ratio', 'Cash_Ratio',
    'Working_Capital_Ratio', 'Equity_to_Liabilities', 'Net_Sales_to_Assets',
    'Net_Worth_to_Debt', 'CF_to_debt', 'CF_to_sales', 'CF_coverage',
    'CF_to_equity', 'CF_to_Assets', 'Gross_Margin', 'RE_to_Sales',
    'RE_to_Assets', 'RE_to_NetIncome', 'Inventory_Turnover', 'Inventory_to_Sales'
]

```

```



# Create a DataFrame indicating missing and inf values
missing_inf_df = pd.DataFrame(index=predictor_columns)
# Fill the DataFrame with zeros (no missing or inf values)
missing_inf_df['Missing'] = 0
missing_inf_df['Inf'] = 0
# Find missing (NaN) values and inf values in the DataFrame
for column in predictor_columns:
    missing_inf_df.at[column, 'Missing'] = bank_data[column].isnull().sum()
    missing_inf_df.at[column, 'Inf'] = np.isinf(bank_data[column]).sum()

```

```
# Create a heatmap to visualize missing and inf values
plt.figure(figsize=(12, 8))
sns.heatmap(missing_inf_df, annot=True, cmap='YlGnBu', cbar=False)
plt.title('Missing and Inf Values Heatmap')
plt.show()

```




```
# Count missing and inf values in each predictor column
missing_counts = {}
inf_counts = {}
for column in predictor_columns:
    missing_counts[column] = bank_data[column].isnull().sum()
    inf_counts[column] = np.isinf(bank_data[column]).sum()

missing_counts_df = pd.DataFrame(missing_counts, index=['Missing Count'])
inf_counts_df = pd.DataFrame(inf_counts, index=['Inf Count'])

# Create a mask to identify rows with at least one missing or inf value in predictor columns
rows_with_missing_or_inf = bank_data[predictor_columns].isnull().any(axis=1) | bank_data[predictor_columns

missing_or_inf_df = bank_data[rows_with_missing_or_inf]

# Print the missing and inf value counts and the DataFrame with affected observations
print("Missing Value Counts:")
print(missing_counts_df)
print("\nInf Value Counts:")
print(inf_counts_df)
print("\nDataFrame with Observations Containing Missing or Inf Values:")
missing_or_inf_df.head(10)

```

```
Missing Value Counts:
               Asset_Turnover_ratio  ROA  Interest_on_Debt  Debt_to_Equity  \
Missing Count                     0    0               238              16

               WC_to_Sales  Quick_ratio  Operating_Margin  Net_Margin  \
Missing Count           15          368                 0           0

               Profit_Margin  Operating_CF  ...  CF_to_sales  CF_coverage  \
Missing Count              0            15  ...           15           20

               CF_to_equity  CF_to_Assets  Gross_Margin  RE_to_Sales  \
Missing Count            28            15           543           13

               RE_to_Assets  RE_to_NetIncome  Inventory_Turnover  \
Missing Count            13              531                 562

               Inventory_to_Sales
Missing Count                  12

[1 rows x 32 columns]

Inf Value Counts:
           Asset_Turnover_ratio  ROA  Interest_on_Debt  Debt_to_Equity  \
Inf Count                     0    0                 0              72

           WC_to_Sales  Quick_ratio  Operating_Margin  Net_Margin  \
Inf Count            0            0                 0           0

           Profit_Margin  Operating_CF  ...  CF_to_sales  CF_coverage  \
Inf Count              0             0  ...            0           37

           CF_to_equity  CF_to_Assets  Gross_Margin  RE_to_Sales  \
Inf Count            72             0             0            0

           RE_to_Assets  RE_to_NetIncome  Inventory_Turnover  \
Inf Count             0             1480                 387

           Inventory_to_Sales
Inf Count                   0

[1 rows x 32 columns]

DataFrame with Observations Containing Missing or Inf Values:
```

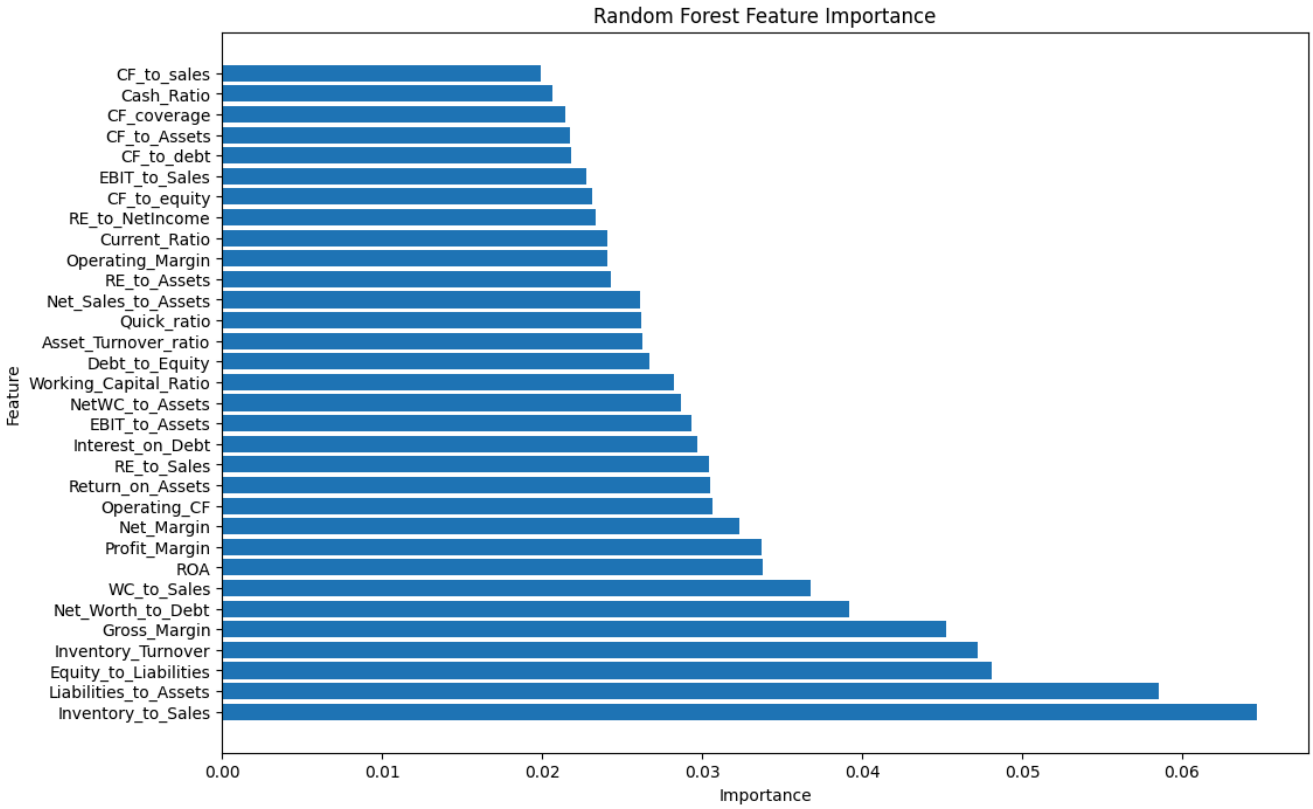
	ORGNR	Year	ZipCode	Municipality	Bankrupt	Business_Category	Size_Category	OperatingProfit
7	5560064403	1998	24131	1285.0	0	30	0	40.0
13	5560064403	2004	24131	1285.0	0	30	0	32.0
16	5560064403	2007	24131	1285.0	0	30	0	37.0
130	5560234238	1998	18205	162.0	0	30	3	-519.0
131	5560234238	1999	18205	162.0	0	30	3	-499.0
153	5560284761	2001	11152	180.0	0	30	0	0.0
157	5560284761	2007	11122	180.0	0	30	0	0.0
169	5560312000	2006	18205	162.0	0	30	0	10.0
173	5560312000	2010	18205	162.0	0	30	0	216.0
174	5560312000	2012	18205	162.0	0	30	0	58.0

10 rows × 125 columns

```
bank_data.replace([np.inf, -np.inf], 0, inplace=True)
bank_data = bank_data.dropna()
print(bank_data['Bankrupt'].value_counts())
print(bank_data.shape)

```

```
Bankrupt
0    73069
1      298
Name: count, dtype: int64
(73367, 125)
```

```
# Select predictors and target column
predictors = [
    'Asset_Turnover_ratio', 'ROA', 'Interest_on_Debt', 'Debt_to_Equity',
    'WC_to_Sales', 'Quick_ratio', 'Operating_Margin',
    'Net_Margin', 'Profit_Margin', 'Operating_CF',
    'Liabilities_to_Assets', 'Return_on_Assets', 'NetWC_to_Assets',
    'EBIT_to_Assets', 'EBIT_to_Sales', 'Current_Ratio', 'Cash_Ratio',
    'Working_Capital_Ratio', 'Equity_to_Liabilities', 'Net_Sales_to_Assets',
    'Net_Worth_to_Debt', 'CF_to_debt', 'CF_to_sales', 'CF_coverage',
    'CF_to_equity', 'CF_to_Assets', 'Gross_Margin', 'RE_to_Sales',
    'RE_to_Assets', 'RE_to_NetIncome', 'Inventory_Turnover', 'Inventory_to_Sales'
]

target = 'Bankrupt'

# Separate predictors and target
X = bank_data[predictors]
y = bank_data[target]

# Fit a Random Forest model
rf_model = RandomForestClassifier(random_state=42, class_weight='balanced')
rf_model.fit(X, y)

# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame to store feature names and their importance scores
feature_importance_df = pd.DataFrame({'Feature': predictors, 'Importance': feature_importances})

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot the feature importance
plt.figure(figsize=(12, 8))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Random Forest Feature Importance')
plt.show()

# Print the sorted feature importance
print("Feature Importance:")
print(feature_importance_df)

```



Feature Importance:

	Feature	Importance
31	Inventory_to_Sales	0.064670
10	Liabilities_to_Assets	0.058577
18	Equity_to_Liabilities	0.048071
30	Inventory_Turnover	0.047256
26	Gross_Margin	0.045235
20	Net_Worth_to_Debt	0.039217
4	WC_to_Sales	0.036796
1	ROA	0.033806
8	Profit_Margin	0.033724
7	Net_Margin	0.032320
9	Operating_CF	0.030627
11	Return_on_Assets	0.030475
27	RE_to_Sales	0.030462
2	Interest_on_Debt	0.029720
13	EBIT_to_Assets	0.029335
12	NetWC_to_Assets	0.028678
17	Working_Capital_Ratio	0.028277
3	Debt_to_Equity	0.026739
0	Asset_Turnover_ratio	0.026295
5	Quick_ratio	0.026226
19	Net_Sales_to_Assets	0.026163
28	RE_to_Assets	0.024271
6	Operating_Margin	0.024085
15	Current_Ratio	0.024064
29	RE_to_NetIncome	0.023322
24	CF_to_equity	0.023149
14	EBIT_to_Sales	0.022773
21	CF_to_debt	0.021807
25	CF_to_Assets	0.021771
23	CF_coverage	0.021480
16	Cash_Ratio	0.020682
22	CF_to_sales	0.019927

```
feature_scores = pd.Series(rf_model.feature_importances_, index=X.columns).sort_values(ascending=False)

feature_scores

```

	0
Inventory_to_Sales	0.064670
Liabilities_to_Assets	0.058577
Equity_to_Liabilities	0.048071
Inventory_Turnover	0.047256
Gross_Margin	0.045235
Net_Worth_to_Debt	0.039217
WC_to_Sales	0.036796
ROA	0.033806
Profit_Margin	0.033724
Net_Margin	0.032320
Operating_CF	0.030627
Return_on_Assets	0.030475
RE_to_Sales	0.030462
Interest_on_Debt	0.029720
EBIT_to_Assets	0.029335
NetWC_to_Assets	0.028678
Working_Capital_Ratio	0.028277
Debt_to_Equity	0.026739
Asset_Turnover_ratio	0.026295
Quick_ratio	0.026226
Net_Sales_to_Assets	0.026163
RE_to_Assets	0.024271
Operating_Margin	0.024085
Current_Ratio	0.024064
RE_to_NetIncome	0.023322
CF_to_equity	0.023149
EBIT_to_Sales	0.022773
CF_to_debt	0.021807
CF_to_Assets	0.021771
CF_coverage	0.021480
Cash_Ratio	0.020682
CF_to_sales	0.019927

dtype: float64

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Use SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)


```

```
print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (44020, 32)
X_test shape:  (29347, 32)
y_train shape:  (44020,)
y_test shape:  (29347,)
```

```
# Count the number of bankrupt and non-bankrupt companies
bankrupt_count_train = y_train.sum()
non_bankrupt_count_train = len(y_train) - bankrupt_count_train
bankrupt_count_test = y_test.sum()
non_bankrupt_count_test = len(y_test) - bankrupt_count_test

print("Train Set:")
print("Bankrupt Companies:", bankrupt_count_train)
print("Non-Bankrupt Companies:", non_bankrupt_count_train)
print("\nTest Set:")
print("Bankrupt Companies:", bankrupt_count_test)
print("Non-Bankrupt Companies:", non_bankrupt_count_test)

```

```
Train Set:
Bankrupt Companies: 182
Non-Bankrupt Companies: 43838

Test Set:
Bankrupt Companies: 116
Non-Bankrupt Companies: 29231
```

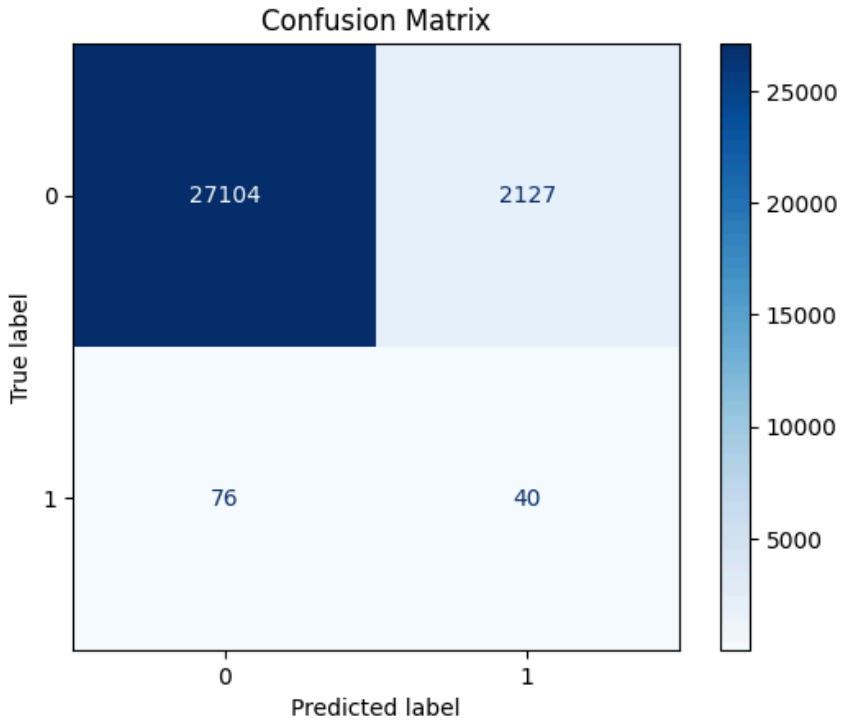
```
# ✅ Import the classifier first
from sklearn.ensemble import RandomForestClassifier

# 1. Define the Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=50,
    max_depth=10,
    class_weight='balanced',
    random_state=42
)#hyper parameter tuning

# 2. Fit on resampled data
rf_model.fit(X_train_resampled, y_train_resampled)

# 3. Predict on test data
y_pred = rf_model.predict(X_test)
```

```
import pickle

# Save the trained Random Forest model
with open("rf_model.pkl", "wb") as f:
    pickle.dump(rf_model, f)

```

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

```

```
Confusion matrix

[[27104  2127]
[   76    40]]
```

```
# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```



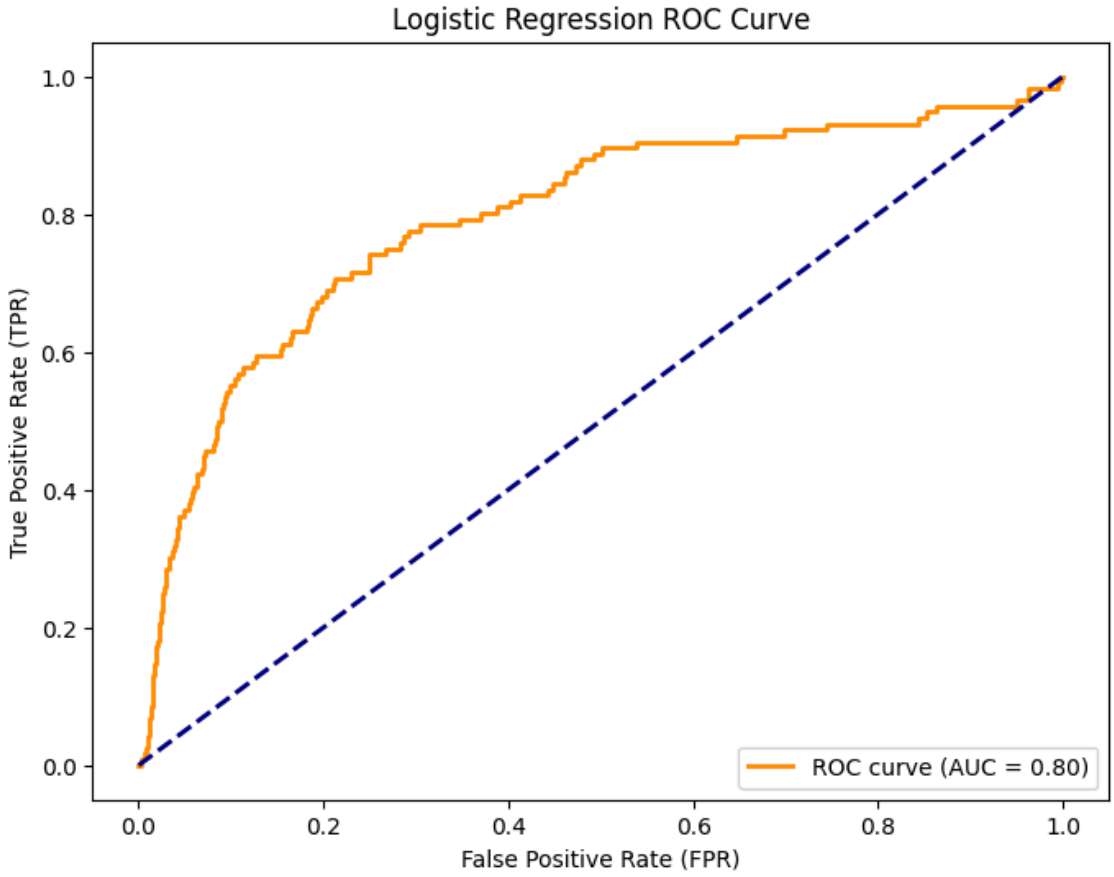

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96     29231
           1       0.02      0.34      0.04       116

    accuracy                           0.92     29347
   macro avg       0.51      0.64      0.50     29347
weighted avg       0.99      0.92      0.96     29347
```

```
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.metrics import roc_curve, roc_auc_score
import joblib
import matplotlib.pyplot as plt

# Assuming you already have your X_train, y_train, X_test, y_test data loaded
# If using resampling techniques like SMOTE for imbalanced data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Initialize Logistic Regression model
logreg_model = LogisticRegression(random_state=426)
```

```
14
15  # Fit the model to the resampled data
16  logreg_model.fit(X_train_resampled, y_train_resampled)
17
18  # Save the logistic regression model
19  joblib.dump(logreg_model, 'logistic_regression_model.pkl')
20
21  # Make predictions on the test set using the logistic regression model
22  y_pred = logreg_model.predict(X_test)
23
24  # Predict probabilities for the positive class
25  y_prob = logreg_model.predict_proba(X_test)[:, 1]
26
27  # Calculate the ROC curve
28  fpr, tpr, thresholds = roc_curve(y_test, y_prob)
29
30  # Calculate the AUC (Area Under the Curve)
31  auc = roc_auc_score(y_test, y_prob)
32
33  # Plot the ROC curve
34  plt.figure(figsize=(8, 6))
35  plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.format(auc))
36  plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
37  plt.xlabel('False Positive Rate (FPR)')
38  plt.ylabel('True Positive Rate (TPR)')
39  plt.title('Logistic Regression ROC Curve')
40  plt.legend(loc='lower right')
41  plt.show()
42
43  # Print the AUC score
44  print("AUC Score:", auc)
45
46  # Now your logistic regression model is ready for predictions.
47
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```




```
AUC Score: 0.7956131834530888
```

```
pip install shap
```

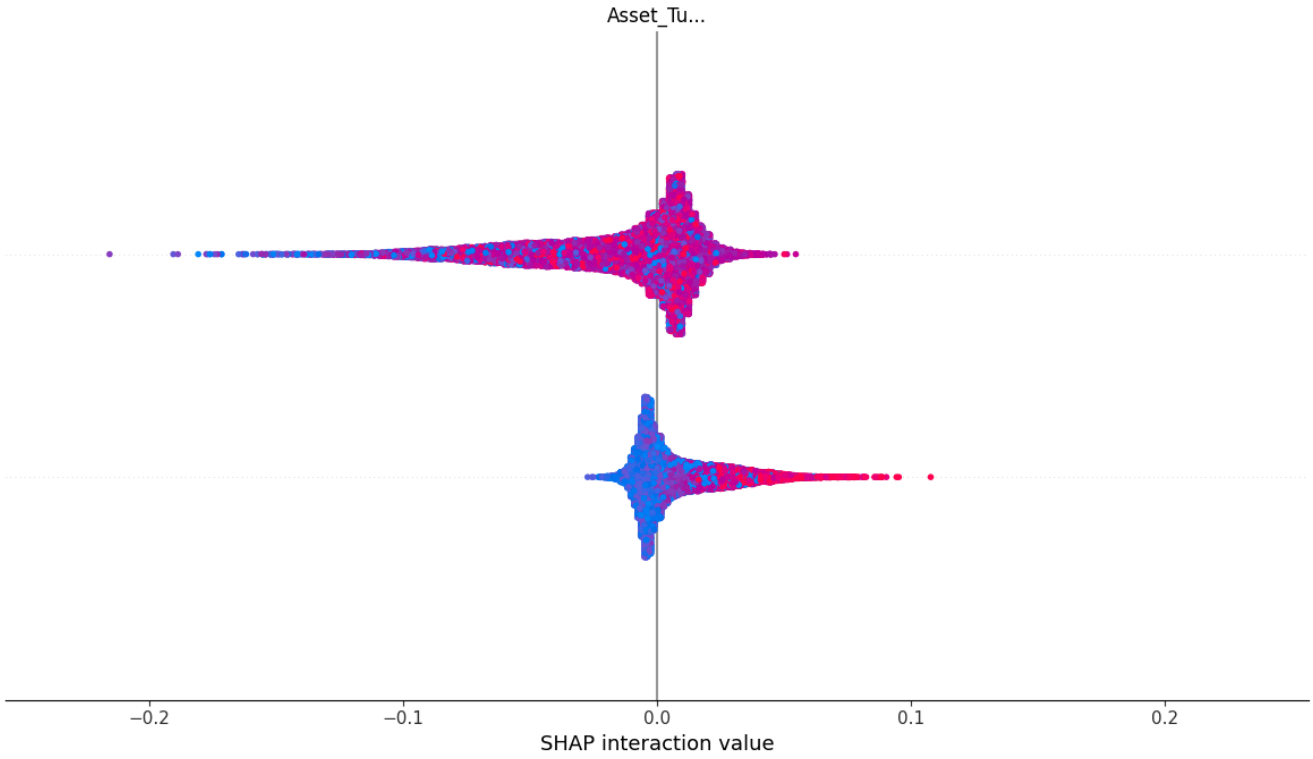
```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages (0.47.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from shap) (1.14.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from shap) (1.6.1
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-packages (from shap) (4.67.
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-packages (from shap) (24.
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-packages (from shap) (0.0.
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-packages (from shap) (0.60.0
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from shap) (4
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->shap
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->sh
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-lear
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scik
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>
```

```
# Only use a sample (e.g., 500 rows) for explanation
import shap
sample_data = X_test.sample(500, random_state=42)

explainer = shap.TreeExplainer(rf_model)
```

```
shap_values = explainer.shap_values(sample_data)

```

```
import shap


# Create a SHAP explainer using the base logistic regression model
explainer = shap.Explainer(rf_model, X_train_resampled)

# Compute SHAP values for the test set (or whichever data you want to explain)
shap_values = explainer.shap_values(X_test)

# Plot SHAP summary plot
shap.summary_plot(shap_values, X_test)

```

```
100%|==================| 58666/58694 [15:44<00:00]
<Figure size 640x480 with 0 Axes>
```




```
import pandas as pd
import shap


# If X_train is a NumPy array, make sure to convert it to a DataFrame with column names
X_train_df = pd.DataFrame(X_train_resampled, columns=predictor_columns)
X_test_df = pd.DataFrame(X_test, columns=predictor_columns)

# Initialize the SHAP explainer
explainer = shap.Explainer(logreg_model, X_train_df)

# Calculate SHAP values
shap_values = explainer(X_test_df)

# Calculate mean absolute SHAP value for each feature (global importance)
shap_summary = pd.DataFrame({
    'Feature': X_test_df.columns,
    'Mean_Abs_SHAP_Value': np.abs(shap_values.values).mean(axis=0)
})

```

```
# Sort by importance
shap_summary_sorted = shap_summary.sort_values(by='Mean_Abs_SHAP_Value', ascending=False)

# Save top 10 (or any number) to CSV
top_n = 10
shap_summary_sorted.head(top_n).to_csv('top_shap_features.csv', index=False)

print("Top SHAP features saved to 'top_shap_features.csv'")

```

```
Top SHAP features saved to 'top_shap_features.csv'
```

```
# Step 1: Import libraries
from sklearn.linear_model import LogisticRegression
import joblib


# Step 3: Save the model
joblib.dump(logreg_model, 'logistic_regression_model.pkl')

print("✅ Model saved as 'logistic_regression_model.pkl'")

```

```
✅ Model saved as 'logistic_regression_model.pkl'
```

```
! pip install streamlit -q

```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 2.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.8/9.8 MB 41.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 49.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 5.2 MB/s eta 0:00:00
```

```
!wget -q -O - ipv4.icanhazip.com
```

```
35.233.148.243
```

```
!streamlit run /content/drive/MyDrive/FYP2025/app_py.py & npx localtunnel --port 8501
```

```
...
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.

⠙⠹⠸
  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.28.0.12:8501
  External URL: http://35.233.148.243:8501

⠼⠴⠦your url is: https://violet-comics-bathe.loca.lt
```

```
!ls /content/drive/MyDrive

```

```
'1st YEAR.zip'
'2023_01_02 3_53 pm Office Lens.pdf'
'311521104033 (1).jpg'
 311521104033arw.pdf
```