Tarush Gupta
DATS 6103 Project: Predicting Bankruptcy
Individual Final Report

1. Introduction

As part of our Data Mining project this semester, our group set out to use the methods learned during this course and solve a challenge that we collectively found interesting. As students in the Masters in Applied Economics program, we share an economic intuition, which we combined with our data mining skills, to explore bankruptcy prediction using a dataset with publicly-traded firms in Taiwan, between 1999 and 2009.

Recent financial collapses have highlighted the importance of regulating large firms and increasing oversight. Consequently, firms audit their financial health annually by reporting their financial ratios and corporate governance indicators (Shailer, 2004). Finance literature, and more recently, machine learning literature has used these indicators to assess the probability of a firm declaring bankruptcy (Altman, 1968; Lee and Yeh, 2004; Lin et al., 2011).

For the purposes of the project, we use a dataset from Kaggle, that was originally collected from the Taiwan Economic Journal between 1999 and 2009 (accessible at: https://www.finasia.biz). The Kaggle data were obtained from the University of California, Irvine, Machine Learning Repository (Liang et al., 2016). We begin by inspecting and preprocessing this data, over which we run inferential and predictive models to ascertain the determinants of bankruptcy.

During our Exploratory Data Analysis, we removed any feature variables with high multicollinearity--|95%| or greater. We still faced some collinearity related issues, which we address using Variance Inflation Factor. The data is well-organized and complete and did not require much attention. We fix the moderate imbalance problem in our binary Boolean target class variable, Bankruptcy (1, if a firm went bankrupt; 0, otherwise), wherein only 3.2% of the observations fall in the minority class, using the Synthetic Minority Oversampling Technique, or SMOTE, as developed by Chawla et al. (2002). We Winsorize any outliers and use the standardize our data before running any models.

Collectively, we run seven statistical models on our data to understand the impact of the feature variables Bankruptcy. To reduce dimensionality and understand the importance of the features within our data, we run the Principal Component Analysis. On this classification data with a binary target variable, we run the Logistic regression, and its capped-extension, the Probit regression. Additionally, we run four tree-based shallow machine learning models: Decision Tree Classifier, Multi-Layer Perceptron Classifier, Histogram-based Gradient Boosting Regressor, and the Linear Probability Model.

Our models indicate that debt-based measures, intuitively, had the most significant statistical impact on increasing the probability of a firm declaring bankruptcy. This is visible in our Linear Probability Model results, which show that a 1% increase in the (Total Debt)/(Total Net Worth) ratio increases the chance of bankruptcy by nearly 5.7 percentage points. Our PCA results show that just 8 components can explain a majority of the data. From our predictive methods, we observe that based on the F1-score, the Decision Tree Classifier fits and explains this data best (75% model accuracy).

We plot and tabulate our results, which we present interactively using a GUI. The GUI enables users to explore the data, its characteristics, and use it to run our statistical models. Our findings are important for investors and financial lenders who want to invest in firms, and use our indicators to assess the financial health of the firms. The results are also useful for building and training other machine learning models.

2. Description of the Individual Work

In the project, I contributed to the coding aspect, writing the report, and making the presentation. In coding, my contribution was two-fold: assisting with the exploratory data analysis, and building the GUI. Within the group report, I contributed the text for the first two sections with additions and edits from my group mates. For the presentation, I added my slides on the exploratory data analysis and the respective references, as well as some formatting changes.

Within the exploratory data analysis, I explored the properties of the data, added figures, and handled multicollinearity and class imbalance. For the GUI, I learned how to use QT Designer and the PyQt5 libraries to implement the interactive aspect. In the report, I led the research and literature review to complete the first section, and explored the properties of our data to complete the second section. I further explain these contributions in the next section.

3. Detailed Description of Individual Work

My additions to the exploratory data analysis include setting up my requirement in the initialization section, adding the ‘nan checker’, checking for multicollinearity using the Pearson and the Spearman Correlation matrices, dropping the multicollinear columns and any other variables without any information (‘Net Income Flag’). Here, I also contribute the income-to-bankruptcy graph. Next, I split the data into training, testing and validation subsets. I further split each of these into their respective target vector and feature matrices. I then standardize this data using the Standard Scaler. On this data, I use a scatter plot to visually inspect the imbalance in our target variable. I implement the solution to this using SMOTE, and once again plot the new target class. Over this data, I run a two-component Principal Component Analysis. My code for this part is as follows:

```
# Initialization ------------------------------------------------------------
-----------
# Ignore Warnings
warnings.filterwarnings('ignore')
# Seaborn Plot Area
sns.set_style('whitegrid')
# Matplotlib Plot Area
plt.rc('font', size=20)
plt.rc('axes', titlesize=20)
plt.rc('axes', labelsize=20)
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('legend', fontsize=20)
plt.rc('figure', titlesize=20)
# Random Seed
seed = 101
# The StandardScaler
ss = StandardScaler()
```

```
# os Path
path = os.path.abspath(os.getcwd())
# Hard encoding data.csv
bankrupt = pd.read_csv("data.csv", sep=',',header=0, index_col=None)
# nan checker
def nan_checker(df):

    # Get the dataframe of variables with NaN, their proportion of NaN and
data type
    df_nan = pd.DataFrame([[var, df[var].isna().sum() / df.shape[0],
df[var].dtype]
                           for var in df.columns if df[var].isna().sum() >
0],
                          columns=['var', 'proportion', 'dtype'])

    # Sort df_nan in ascending order of the proportion of NaN
    df_nan = df_nan.sort_values(by='proportion',
ascending=False).reset_index(drop=True)

    return df_nan
df_nan = nan_checker(bankrupt)
print("No variables with missing values:")
print(df_nan)     # Shows there are no columns with any missing values

# TG - multicollinearity suggestion - remove 17 cols that are >=|95%|
collinearly related - NOT SPEARMAN
multicoll_mat = []
for i in range(len(corr_mat.columns)):
    for j in range(i):
        if(corr_mat.iloc[i,j] >= 0.95 or corr_mat.iloc[i,j] <= -0.95):
            if corr_mat.columns[j] not in multicoll_mat:
                multicoll_mat.append(corr_mat.columns[j])
print("Spearman Correlation Matrix Multicollinearity
Columns:",len(multicoll_mat))
print(multicoll_mat)

corr_mtx = bankrupt.corr()
corr_mtx = corr_mtx.iloc[1:,1:]
multicoll_mtx = []
for i in range(len(corr_mtx.columns)):
    for j in range(i):
        if (corr_mtx.iloc[i, j] >= 0.95 or corr_mtx.iloc[i, j] <= -0.95):
            if corr_mtx.columns[j] not in multicoll_mtx:
                multicoll_mtx.append(corr_mtx.columns[j])
print("Pearson Correlation Matrix Multicollinearity
Columns:",len(multicoll_mtx))
print(multicoll_mtx)
mcm_diff = [ele for ele in multicoll_mat if ele not in multicoll_mtx]
print("Columns identified by Spearman but not Pearson: ",mcm_diff) # Some are
useful, removing only

#Removing multicollinearity columns identified by Pearson Corr
bankrupt = bankrupt.drop(multicoll_mtx,axis=1)
# Encoding Data and Handling Non-Numerical Data
# - No identifiers to remove, no date-time vars to standardize
# - Dropping 'NetIncomeFlag' since it has no values
```

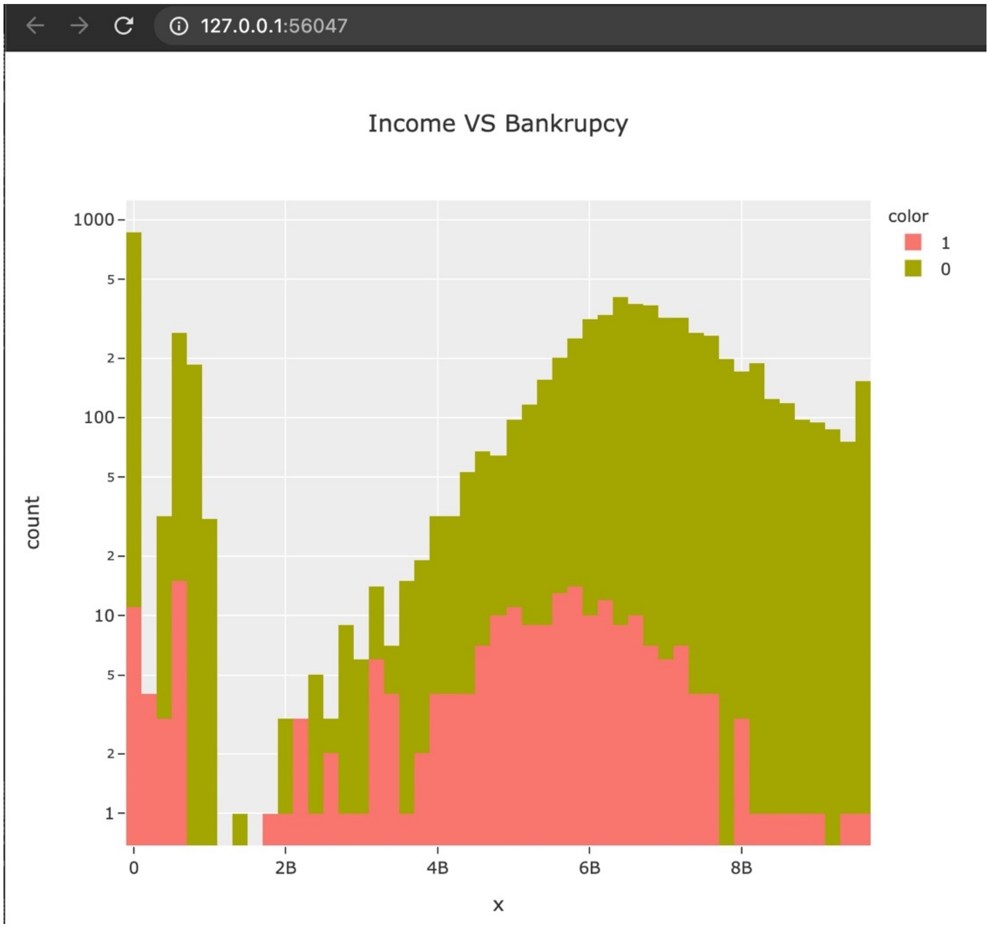
```python
bankrupt['NetIncomeFlag'].value_counts()
bankrupt.drop(['NetIncomeFlag'],axis=1,inplace=True)
#Income-to-Bankruptcy Graph
itob = px.histogram(x=bankrupt['TotalAssetGrowthRate'],
                    color=bankrupt['Bankrupt'],
                   log_y=True,
                   template='ggplot2',
                  title='Income VS Bankrupcy',
                  width=700)
itob.show()
# Splitting the data
# Splits: Training (60%), Validation (20%), Testing (20%)
# Dividing dataset into training (60%) and test (40%)
df_train, df_test = train_test_split(bankrupt,
                                     train_size=0.6,
                                     random_state=seed,
                                     stratify=bankrupt['Bankrupt'])
# Dividing testing data into validation (50%) and testing (50%)
df_val, df_test = train_test_split(df_test,
                                   train_size=0.5,
                                   random_state=seed,
                                   stratify=df_test['Bankrupt'])

# Resetting the index
df_train, df_val, df_test = df_train.reset_index(drop=True),
df_val.reset_index(drop=True), df_test.reset_index(drop=True)
# Shapes of new splits
## Train
pd.DataFrame([[df_train.shape[0], df_train.shape[1]]], columns=['# rows', '#
columns'])
## Val
pd.DataFrame([[df_val.shape[0], df_val.shape[1]]], columns=['# rows', '#
columns'])
## Test
pd.DataFrame([[df_test.shape[0], df_test.shape[1]]], columns=['# rows', '#
columns'])

#Splitting feature and target matrices
# Get the feature matrix
X_train = df_train[np.setdiff1d(df_train.columns, ['Bankrupt'])].values
X_val = df_val[np.setdiff1d(df_val.columns, ['Bankrupt'])].values
X_test = df_test[np.setdiff1d(df_test.columns, ['Bankrupt'])].values

# Get the target vector
y_train = df_train['Bankrupt'].values
y_val = df_val['Bankrupt'].values
y_test = df_test['Bankrupt'].values

# Standardizing the features
# Standardize the training data
X_train = ss.fit_transform(X_train)
# Standardize the validation data
X_val = ss.transform(X_val)
# Standardize the test data
X_test = ss.transform(X_test)

# Handling class imbalance
```

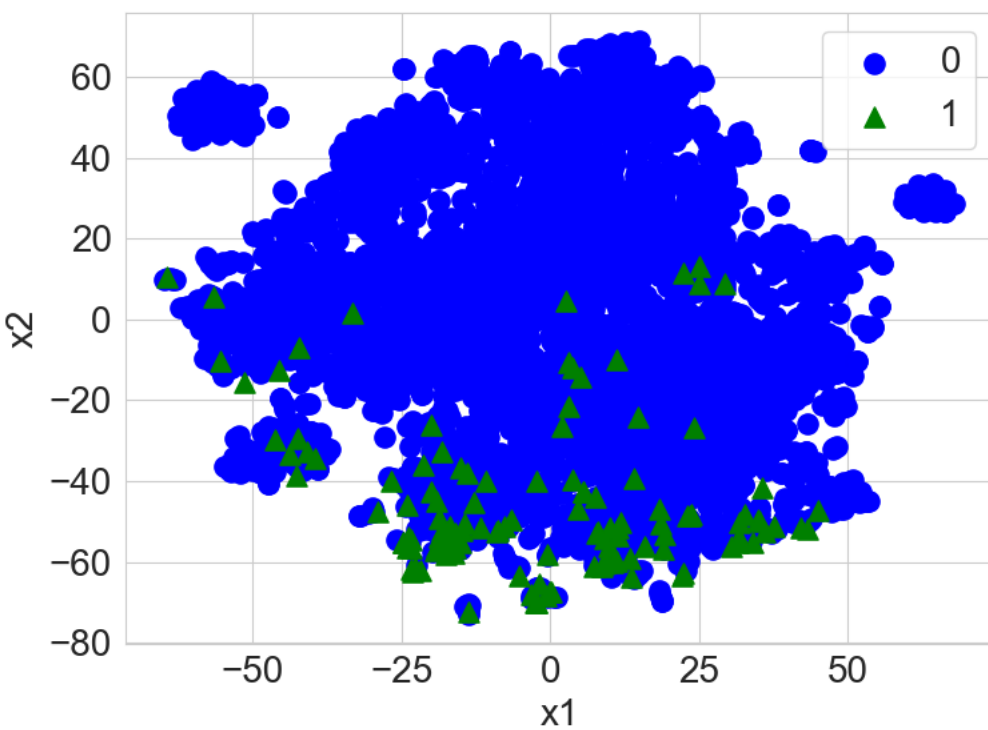
```
# Imbalance class distribution
pd.Series(y_train).value_counts()

# TSNE Scatter Plot
def plot_scatter_tsne(X, y, classes, labels, colors, markers, loc, dir_name,
fig_name, random_seed):

    # Make directory
    directory = os.path.dirname(dir_name)
    if not os.path.exists(directory):
        os.makedirs(directory)

    # Get the tsne transformed training feature matrix
    X_embedded = TSNE(n_components=2,
random_state=random_seed).fit_transform(X)

    # Get the tsne dataframe
    tsne_df = pd.DataFrame(np.column_stack((X_embedded, y)), columns=['x1',
'x2', 'y'])

    # Get the data
    data = {}
    for class_ in classes:
        data_x1 = [tsne_df['x1'][i] for i in range(len(tsne_df['y'])) if
tsne_df['y'][i] == class_]
        data_x2 = [tsne_df['x2'][i] for i in range(len(tsne_df['y'])) if
tsne_df['y'][i] == class_]
        data[class_] = [data_x1, data_x2]

    # The scatter plot
    fig = plt.figure(figsize=(8, 6))

    for class_, label, color, marker in zip(classes, labels, colors,
markers):
        data_x1, data_x2 = data[class_]
        plt.scatter(data_x1, data_x2, c=color, marker=marker, s=120,
label=label)

    # Set x-axis
    plt.xlabel('x1')

    # Set y-axis
    plt.ylabel('x2')

    # Set legend
    plt.legend(loc='best')

    # Save and show the figure
    plt.tight_layout()
    plt.savefig(dir_name + fig_name)
    plt.show()
# Plot
plot_scatter_tsne(X_train,
                  y_train,
                  [0, 1],
                  ['0', '1'],
                  ['blue', 'green'],
```

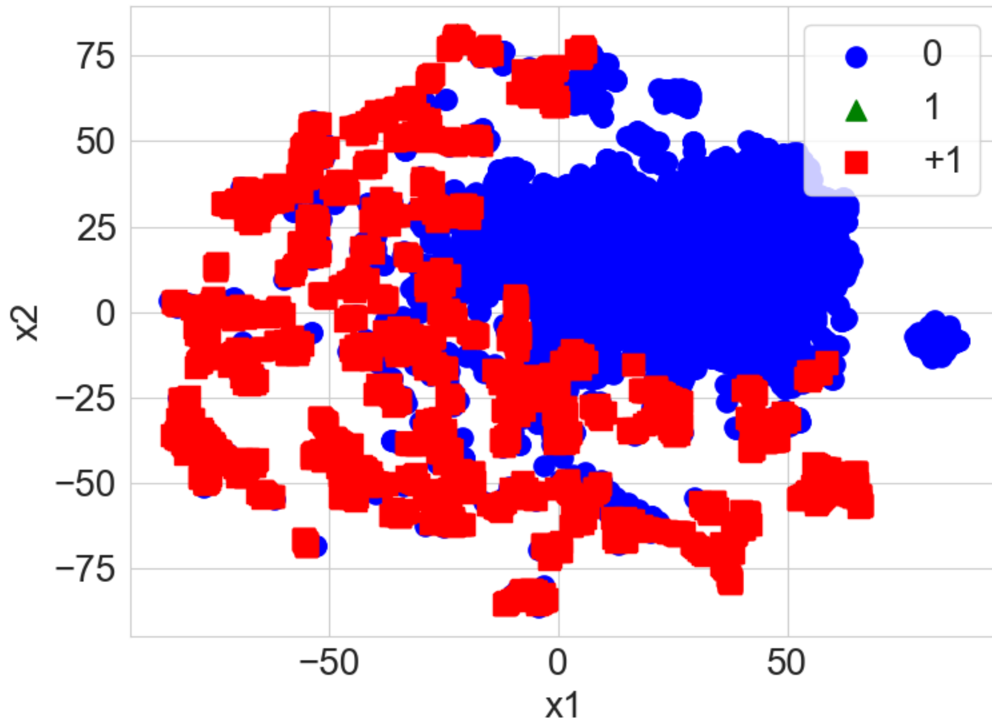
```
                   ['o', '^'],
                   'bottom-left',
                   path,
                   'scatter_plot_baseline.pdf',
                   seed)
# Handling class imbalance in the target using SMOTE -
smote = SMOTE(random_state=seed)
# Augment the training data
X_smote_train, y_smote_train = smote.fit_resample(X_train, y_train)
# See new distributions
pd.Series(y_smote_train).value_counts()
# Separate generated class from original class
def separate_generate_original(X_aug_train, y_aug_train, X_train, y_train,
minor_class):
    # Make a copy of y_aug_train
    y_aug_gen_ori_train = np.array(y_aug_train)

    # For each sample in the augmented data
    for i in range(X_aug_train.shape[0]):
        # If the sample has the minor class
        if y_aug_gen_ori_train[i] == minor_class:
            # Flag variable, indicating whether a sample in the augmented
data is the same as a sample in the original data
            same = False

            # For each sample in the original data
            for j in range(X_train.shape[0]):
                # If the sample has the minor class
                if y_train[j] == minor_class:
                    if len(np.setdiff1d(X_aug_train[i, :], X_train[j, :])) ==
0:
                        # The two samples are the same
                        same = True
                        break

            # If the two samples are different
            if same is False:
                y_aug_gen_ori_train[i] = 2

    return y_aug_gen_ori_train
y_smote_gen_ori_train = separate_generate_original(X_smote_train,
y_smote_train, X_train, y_train, 1)
# Plot of new SMOTE classes
plot_scatter_tsne(X_smote_train,
                   y_smote_gen_ori_train,
                   [0, 1, 2],
                   ['0', '1', '+1'],
                   ['blue', 'green', 'red'],
                   ['o', '^', 's'],
                   'bottom-right',
                   path,
                   'scatter_plot_smote.pdf',
                   seed)

# PCA
# Standardize the training data
X_train = ss.fit_transform(X_train)
```

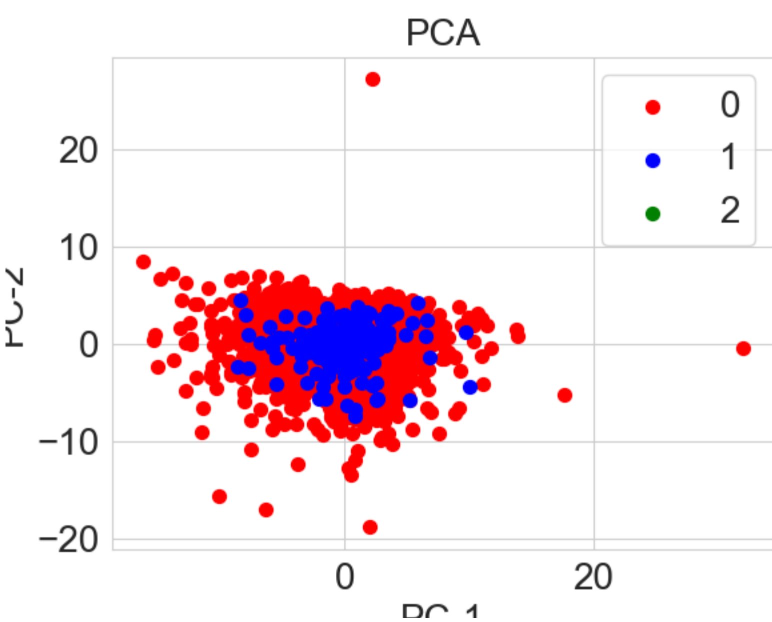
```
y_train = ss.fit_transform(y_train.reshape(-1, 1)).reshape(-1)
# Standardize the validation data
X_val = ss.fit_transform(X_val)
y_val = ss.fit_transform(y_val.reshape(-1, 1)).reshape(-1)
# Standardize the test data
X_test = ss.fit_transform(X_test)
y_test = ss.fit_transform(y_test.reshape(-1, 1)).reshape(-1)

# Encoding Training data for PCA
X_t = X_train.reshape(-1)
l_e = preprocessing.LabelEncoder()
enc = l_e.fit_transform(X_t)

# Applying PCA on training
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
print("Explained variation per Principal Component:
{}".format(pca.explained_variance_ratio_))
# PC1 holds 13.92% of info; PC2 holds 7.6% of info

#PCA Plot
#PC Values
bank = pd.DataFrame(data=X_train,columns=['PC-1','PC-2'])
plt.figure()
plt.xlabel("PC-1")
plt.ylabel("PC-2")
plt.title("PCA")
targets = [0,1,2]
colors = ['r','b','g']
for target, color in zip(targets, colors):
    ind = bankrupt['Bankrupt']==target
    plt.scatter(bank.loc[ind,'PC-1'],bank.loc[ind,'PC-2'],c=color,s=50)
plt.legend(targets,prop={'size':20})
plt.show()
```

The resulting figures from this code are:


127.0.0.1:56047
Income VS Bankrupcy
color
1
0
count
1000
5
2
100
5
2
10
5
2
1
0
2B
4B
6B
8B
x


0
1
x2
60
40
20
0
−20
−40
−60
−80
−50
−25
0
25
50
x1





For the GUI, I learned how to use QT Designer to build our application. This created a .ui file, which we converted to a .py file, which I use to connect our code with the GUI. The code from the .ui converted to the .py file is as follows:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file
'C:\Users\Amar\BR_GUI.ui'
```

```
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(667, 417)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.widget = QtWidgets.QWidget(self.centralwidget)
        self.widget.setGeometry(QtCore.QRect(30, 10, 537, 161))
        self.widget.setObjectName("widget")
        self.gridLayout = QtWidgets.QGridLayout(self.widget)
        self.gridLayout.setContentsMargins(0, 0, 0, 0)
        self.gridLayout.setObjectName("gridLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.pushButton_4 = QtWidgets.QPushButton(self.widget)
        self.pushButton_4.setObjectName("pushButton_4")
        self.horizontalLayout.addWidget(self.pushButton_4)
        self.pushButton_2 = QtWidgets.QPushButton(self.widget)
        self.pushButton_2.setObjectName("pushButton_2")
        self.horizontalLayout.addWidget(self.pushButton_2)
        self.pushButton_3 = QtWidgets.QPushButton(self.widget)
        self.pushButton_3.setObjectName("pushButton_3")
        self.horizontalLayout.addWidget(self.pushButton_3)
        self.pushButton = QtWidgets.QPushButton(self.widget)
        self.pushButton.setObjectName("pushButton")
        self.horizontalLayout.addWidget(self.pushButton)
        self.gridLayout.addLayout(self.horizontalLayout, 3, 0, 1, 1)
        self.label_2 = QtWidgets.QLabel(self.widget)
        self.label_2.setObjectName("label_2")
        self.gridLayout.addWidget(self.label_2, 6, 0, 1, 1)
        self.label = QtWidgets.QLabel(self.widget)
        self.label.setObjectName("label")
        self.gridLayout.addWidget(self.label, 1, 0, 1, 1)
        spacerItem = QtWidgets.QSpacerItem(20, 48,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.gridLayout.addItem(spacerItem, 5, 0, 1, 1)
        spacerItem1 = QtWidgets.QSpacerItem(20, 48,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.gridLayout.addItem(spacerItem1, 4, 0, 1, 1)
        spacerItem2 = QtWidgets.QSpacerItem(20, 48,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.gridLayout.addItem(spacerItem2, 2, 0, 1, 1)
        spacerItem3 = QtWidgets.QSpacerItem(568, 20,
QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem3, 0, 0, 1, 1)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
```

```
        self.menubar.setGeometry(QtCore.QRect(0, 0, 667, 24))
        self.menubar.setObjectName("menubar")
        self.menuStart = QtWidgets.QMenu(self.menubar)
        self.menuStart.setObjectName("menuStart")
        self.menuImport_Data = QtWidgets.QMenu(self.menuStart)
        self.menuImport_Data.setObjectName("menuImport_Data")
        self.menuPlot = QtWidgets.QMenu(self.menubar)
        self.menuPlot.setObjectName("menuPlot")
        self.menuCorrelation_Matrix = QtWidgets.QMenu(self.menuPlot)
        self.menuCorrelation_Matrix.setObjectName("menuCorrelation_Matrix")
        self.menuInferential_Plots = QtWidgets.QMenu(self.menuPlot)
        self.menuInferential_Plots.setObjectName("menuInferential_Plots")
        self.menuProperties = QtWidgets.QMenu(self.menubar)
        self.menuProperties.setObjectName("menuProperties")
        self.menuDataset = QtWidgets.QMenu(self.menuProperties)
        self.menuDataset.setObjectName("menuDataset")
        self.menuVariables_2 = QtWidgets.QMenu(self.menuProperties)
        self.menuVariables_2.setObjectName("menuVariables_2")
        self.menuTablulate = QtWidgets.QMenu(self.menubar)
        self.menuTablulate.setObjectName("menuTablulate")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.toolBar = QtWidgets.QToolBar(MainWindow)
        self.toolBar.setObjectName("toolBar")
        MainWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.toolBar)
        self.actionVariables = QtWidgets.QAction(MainWindow)
        self.actionVariables.setObjectName("actionVariables")
        self.actionClean_Data = QtWidgets.QAction(MainWindow)
        self.actionClean_Data.setObjectName("actionClean_Data")
        self.actionBankrupt = QtWidgets.QAction(MainWindow)
        self.actionBankrupt.setObjectName("actionBankrupt")
        self.actionROA_C_beforeinterestanddepreceiationbeforeinterest =
QtWidgets.QAction(MainWindow)

self.actionROA_C_beforeinterestanddepreceiationbeforeinterest.setObjectName("
actionROA_C_beforeinterestanddepreceiationbeforeinterest")
        self.actionROA_A_beforeinterestsnd_aftertax =
QtWidgets.QAction(MainWindow)

self.actionROA_A_beforeinterestsnd_aftertax.setObjectName("actionROA_A_before
interestsnd_aftertax")
        self.actionROA_B_beforeinterestanddespreciationaftertax =
QtWidgets.QAction(MainWindow)

self.actionROA_B_beforeinterestanddespreciationaftertax.setObjectName("action
ROA_B_beforeinterestanddespreciationaftertax")
        self.actionOperating_Gross_Margin = QtWidgets.QAction(MainWindow)

self.actionOperating_Gross_Margin.setObjectName("actionOperating_Gross_Margin
")
        self.actionRealized_Sales_Gross_Margin =
QtWidgets.QAction(MainWindow)

self.actionRealized_Sales_Gross_Margin.setObjectName("actionRealized_Sales_Gr
oss_Margin")
```

```
        self.actionOperating_Profit_Rate = QtWidgets.QAction(MainWindow)

self.actionOperating_Profit_Rate.setObjectName("actionOperating_Profit_Rate")
        self.actionPre_tax_Net_Interest_rate = QtWidgets.QAction(MainWindow)

self.actionPre_tax_Net_Interest_rate.setObjectName("actionPre_tax_Net_Interes
t_rate")
        self.actionAfter_tax_Net_Interest_Rate =
QtWidgets.QAction(MainWindow)

self.actionAfter_tax_Net_Interest_Rate.setObjectName("actionAfter_tax_Net_Int
erest_Rate")
        self.actionNon_industry_Income_and_Expenditure_Revenue =
QtWidgets.QAction(MainWindow)

self.actionNon_industry_Income_and_Expenditure_Revenue.setObjectName("actionN
on_industry_Income_and_Expenditure_Revenue")
        self.actionContinuous_Interest_Rate_After_Tax =
QtWidgets.QAction(MainWindow)

self.actionContinuous_Interest_Rate_After_Tax.setObjectName("actionContinuous
_Interest_Rate_After_Tax")
        self.actionOperating_Expense_Rate = QtWidgets.QAction(MainWindow)

self.actionOperating_Expense_Rate.setObjectName("actionOperating_Expense_Rate
")
        self.actionResearch_and_Development_Expense_Rate =
QtWidgets.QAction(MainWindow)

self.actionResearch_and_Development_Expense_Rate.setObjectName("actionResearc
h_and_Development_Expense_Rate")
        self.actionCash_Flow_Rate = QtWidgets.QAction(MainWindow)
        self.actionCash_Flow_Rate.setObjectName("actionCash_Flow_Rate")
        self.actionTax_Rate_A = QtWidgets.QAction(MainWindow)
        self.actionTax_Rate_A.setObjectName("actionTax_Rate_A")
        self.actionNet_Value_Per_Share_A = QtWidgets.QAction(MainWindow)

self.actionNet_Value_Per_Share_A.setObjectName("actionNet_Value_Per_Share_A")
        self.actionNet_Value_Per_Share_B = QtWidgets.QAction(MainWindow)

self.actionNet_Value_Per_Share_B.setObjectName("actionNet_Value_Per_Share_B")
        self.actionNet_Value_Per_Share_C = QtWidgets.QAction(MainWindow)

self.actionNet_Value_Per_Share_C.setObjectName("actionNet_Value_Per_Share_C")
        self.actionPersistent_EPS_in_the_Last_Four_Seasons =
QtWidgets.QAction(MainWindow)

self.actionPersistent_EPS_in_the_Last_Four_Seasons.setObjectName("actionPersi
stent_EPS_in_the_Last_Four_Seasons")
        self.actionCash_Flow_Per_Share = QtWidgets.QAction(MainWindow)

self.actionCash_Flow_Per_Share.setObjectName("actionCash_Flow_Per_Share")
        self.actionRevenue_Per_Share_YuanY = QtWidgets.QAction(MainWindow)

self.actionRevenue_Per_Share_YuanY.setObjectName("actionRevenue_Per_Share_Yua
nY")
        self.actionOperating_Profit_Per_Share_YuanY =
```

```
QtWidgets.QAction(MainWindow)

self.actionOperating_Profit_Per_Share_YuanY.setObjectName("actionOperating_Pr
ofit_Per_Share_YuanY")
        self.actionPer_SHare_Net_Profit_Before_Tax_YuanY =
QtWidgets.QAction(MainWindow)

self.actionPer_SHare_Net_Profit_Before_Tax_YuanY.setObjectName("actionPer_SHa
re_Net_Profit_Before_Tax_YuanY")
        self.actionRealized_Sales_Gross_Profit_Growth_Rate =
QtWidgets.QAction(MainWindow)

self.actionRealized_Sales_Gross_Profit_Growth_Rate.setObjectName("actionReali
zed_Sales_Gross_Profit_Growth_Rate")
        self.actionOperating_Profit_Growth_Rate =
QtWidgets.QAction(MainWindow)

self.actionOperating_Profit_Growth_Rate.setObjectName("actionOperating_Profit
_Growth_Rate")
        self.actionAfter_tax_Net_Profit_Growth_Rate =
QtWidgets.QAction(MainWindow)

self.actionAfter_tax_Net_Profit_Growth_Rate.setObjectName("actionAfter_tax_Ne
t_Profit_Growth_Rate")
        self.actionRegular_Net_Profit_Growth_Rate =
QtWidgets.QAction(MainWindow)

self.actionRegular_Net_Profit_Growth_Rate.setObjectName("actionRegular_Net_Pr
ofit_Growth_Rate")
        self.actionContinuous_Net_Profit_Growth_Rate =
QtWidgets.QAction(MainWindow)

self.actionContinuous_Net_Profit_Growth_Rate.setObjectName("actionContinuous_
Net_Profit_Growth_Rate")
        self.actionTotal_Asset_Growth_Rate = QtWidgets.QAction(MainWindow)

self.actionTotal_Asset_Growth_Rate.setObjectName("actionTotal_Asset_Growth_Ra
te")
        self.actionNet_Value_Growth_Rate = QtWidgets.QAction(MainWindow)

self.actionNet_Value_Growth_Rate.setObjectName("actionNet_Value_Growth_Rate")
        self.actionTotal_Asset_Return_Growth_Rate_Ratio =
QtWidgets.QAction(MainWindow)

self.actionTotal_Asset_Return_Growth_Rate_Ratio.setObjectName("actionTotal_As
set_Return_Growth_Rate_Ratio")
        self.actionCash_Reinvestment = QtWidgets.QAction(MainWindow)
        self.actionCash_Reinvestment.setObjectName("actionCash_Reinvestment")
        self.actionCurrent_Ratio = QtWidgets.QAction(MainWindow)
        self.actionCurrent_Ratio.setObjectName("actionCurrent_Ratio")
        self.actionQuick_Ratio = QtWidgets.QAction(MainWindow)
        self.actionQuick_Ratio.setObjectName("actionQuick_Ratio")
        self.actionInterest_Expense_Ratio = QtWidgets.QAction(MainWindow)

self.actionInterest_Expense_Ratio.setObjectName("actionInterest_Expense_Ratio
")
        self.actionTotal_Debt_Total_net_Worth = QtWidgets.QAction(MainWindow)
```

```
self.actionTotal_Debt_Total_net_Worth.setObjectName("actionTotal_Debt_Total_n
et_Worth")
        self.actionDebt_Ratio = QtWidgets.QAction(MainWindow)
        self.actionDebt_Ratio.setObjectName("actionDebt_Ratio")
        self.actionNet_Worth_Assets = QtWidgets.QAction(MainWindow)
        self.actionNet_Worth_Assets.setObjectName("actionNet_Worth_Assets")
        self.actionShape = QtWidgets.QAction(MainWindow)
        self.actionShape.setObjectName("actionShape")
        self.actionList = QtWidgets.QAction(MainWindow)
        self.actionList.setObjectName("actionList")
        self.actionSummary_Statistics = QtWidgets.QAction(MainWindow)

self.actionSummary_Statistics.setObjectName("actionSummary_Statistics")
        self.actionSize = QtWidgets.QAction(MainWindow)
        self.actionSize.setObjectName("actionSize")
        self.actionComplete = QtWidgets.QAction(MainWindow)
        self.actionComplete.setObjectName("actionComplete")
        self.actionPositive = QtWidgets.QAction(MainWindow)
        self.actionPositive.setObjectName("actionPositive")
        self.actionNegative = QtWidgets.QAction(MainWindow)
        self.actionNegative.setObjectName("actionNegative")
        self.actionPCA_Feature_Composition = QtWidgets.QAction(MainWindow)

self.actionPCA_Feature_Composition.setObjectName("actionPCA_Feature_Compositi
on")
        self.actionClass_Imbalance = QtWidgets.QAction(MainWindow)
        self.actionClass_Imbalance.setObjectName("actionClass_Imbalance")
        self.actionDebt_Ratio_to_Bankruptcy = QtWidgets.QAction(MainWindow)

self.actionDebt_Ratio_to_Bankruptcy.setObjectName("actionDebt_Ratio_to_Bankru
ptcy")
        self.actionDebt_Ratio_vs_Asset_Turnover =
QtWidgets.QAction(MainWindow)

self.actionDebt_Ratio_vs_Asset_Turnover.setObjectName("actionDebt_Ratio_vs_As
set_Turnover")
        self.actionGrouped_Means = QtWidgets.QAction(MainWindow)
        self.actionGrouped_Means.setObjectName("actionGrouped_Means")
        self.actionLPM_Results = QtWidgets.QAction(MainWindow)
        self.actionLPM_Results.setObjectName("actionLPM_Results")
        self.actionProbit_Results = QtWidgets.QAction(MainWindow)
        self.actionProbit_Results.setObjectName("actionProbit_Results")
        self.actionRaw_Data = QtWidgets.QAction(MainWindow)
        self.actionRaw_Data.setObjectName("actionRaw_Data")
        self.actionClean_Data_2 = QtWidgets.QAction(MainWindow)
        self.actionClean_Data_2.setObjectName("actionClean_Data_2")
        self.actionView_GitHub_Repository = QtWidgets.QAction(MainWindow)

self.actionView_GitHub_Repository.setObjectName("actionView_GitHub_Repository
")
        self.actionView_Project_Report = QtWidgets.QAction(MainWindow)

self.actionView_Project_Report.setObjectName("actionView_Project_Report")
        self.menuImport_Data.addAction(self.actionRaw_Data)
        self.menuImport_Data.addSeparator()
        self.menuImport_Data.addAction(self.actionClean_Data_2)
```

```
        self.menuStart.addAction(self.menuImport_Data.menuAction())
        self.menuStart.addSeparator()
        self.menuStart.addAction(self.actionClean_Data)
        self.menuStart.addAction(self.actionView_GitHub_Repository)
        self.menuStart.addAction(self.actionView_Project_Report)
        self.menuCorrelation_Matrix.addAction(self.actionComplete)
        self.menuCorrelation_Matrix.addSeparator()
        self.menuCorrelation_Matrix.addAction(self.actionPositive)
        self.menuCorrelation_Matrix.addAction(self.actionNegative)
        self.menuInferential_Plots.addAction(self.actionClass_Imbalance)

self.menuInferential_Plots.addAction(self.actionPCA_Feature_Composition)
        self.menuInferential_Plots.addSeparator()

self.menuInferential_Plots.addAction(self.actionDebt_Ratio_to_Bankruptcy)

self.menuInferential_Plots.addAction(self.actionDebt_Ratio_vs_Asset_Turnover)
        self.menuInferential_Plots.addSeparator()
        self.menuInferential_Plots.addAction(self.actionGrouped_Means)
        self.menuPlot.addAction(self.menuCorrelation_Matrix.menuAction())
        self.menuPlot.addSeparator()
        self.menuPlot.addAction(self.menuInferential_Plots.menuAction())
        self.menuDataset.addAction(self.actionShape)
        self.menuDataset.addAction(self.actionSize)
        self.menuVariables_2.addAction(self.actionList)
        self.menuVariables_2.addAction(self.actionSummary_Statistics)
        self.menuProperties.addAction(self.menuDataset.menuAction())
        self.menuProperties.addAction(self.menuVariables_2.menuAction())
        self.menuTablulate.addAction(self.actionLPM_Results)
        self.menuTablulate.addSeparator()
        self.menuTablulate.addAction(self.actionProbit_Results)
        self.menubar.addAction(self.menuStart.menuAction())
        self.menubar.addAction(self.menuProperties.menuAction())
        self.menubar.addAction(self.menuPlot.menuAction())
        self.menubar.addAction(self.menuTablulate.menuAction())
        self.toolBar.addAction(self.actionVariables)
        self.toolBar.addSeparator()

        self.retranslateUi(MainWindow)
        self.pushButton_4.clicked.connect(MainWindow.pushButton_click)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton_4.setText(_translate("MainWindow", "PCA Results"))
        self.pushButton_2.setText(_translate("MainWindow", "LPM Results"))
        self.pushButton_3.setText(_translate("MainWindow", "Probit Results"))
        self.pushButton.setText(_translate("MainWindow", "Correlation
Heatmap"))
        self.label_2.setText(_translate("MainWindow", "Predictive Results "))
        self.label.setText(_translate("MainWindow", "Inferential Results"))
        self.menuStart.setTitle(_translate("MainWindow", "Start"))
        self.menuImport_Data.setTitle(_translate("MainWindow", "Import
Data..."))
        self.menuPlot.setTitle(_translate("MainWindow", "Plot"))
        self.menuCorrelation_Matrix.setTitle(_translate("MainWindow",
```

```
"Correlation Matrix..."))
        self.menuInferential_Plots.setTitle(_translate("MainWindow",
"Inferential Plots..."))
        self.menuProperties.setTitle(_translate("MainWindow", "Properties"))
        self.menuDataset.setTitle(_translate("MainWindow", "Dataset..."))
        self.menuVariables_2.setTitle(_translate("MainWindow",
"Variables..."))
        self.menuTablulate.setTitle(_translate("MainWindow", "Tablulate"))
        self.toolBar.setWindowTitle(_translate("MainWindow", "toolBar"))
        self.actionVariables.setText(_translate("MainWindow", "Variables"))
        self.actionVariables.setToolTip(_translate("MainWindow", "Dataset
Variables"))
        self.actionClean_Data.setText(_translate("MainWindow", "View Kaggle
Competition"))
        self.actionBankrupt.setText(_translate("MainWindow", "Bankrupt"))

self.actionROA_C_beforeinterestanddepreceiationbeforeinterest.setText(_transl
ate("MainWindow", "ROA(C) before interest and depreceiation before
interest"))

self.actionROA_A_beforeinterestsnd_aftertax.setText(_translate("MainWindow",
"ROA(A) before interest and % after tax"))

self.actionROA_B_beforeinterestanddespreciationaftertax.setText(_translate("M
ainWindow", "ROA(B) before interest and depreciation after tax "))
        self.actionOperating_Gross_Margin.setText(_translate("MainWindow",
"Operating Gross Margin"))

self.actionRealized_Sales_Gross_Margin.setText(_translate("MainWindow",
"Realized Sales Gross Margin"))
        self.actionOperating_Profit_Rate.setText(_translate("MainWindow",
"Operating Profit Rate"))
        self.actionPre_tax_Net_Interest_rate.setText(_translate("MainWindow",
"Pre-tax Net Interest rate"))

self.actionAfter_tax_Net_Interest_Rate.setText(_translate("MainWindow",
"After-tax Net Interest Rate"))

self.actionNon_industry_Income_and_Expenditure_Revenue.setText(_translate("Ma
inWindow", "Non-industry Income and Expenditure / Revenue"))

self.actionContinuous_Interest_Rate_After_Tax.setText(_translate("MainWindow"
, "Continuous Interest Rate (After Tax)"))
        self.actionOperating_Expense_Rate.setText(_translate("MainWindow",
"Operating Expense Rate"))

self.actionResearch_and_Development_Expense_Rate.setText(_translate("MainWind
ow", "Research and Development Expense Rate"))
        self.actionCash_Flow_Rate.setText(_translate("MainWindow", "Cash Flow
Rate"))
        self.actionTax_Rate_A.setText(_translate("MainWindow", "Tax Rate
(A)"))
        self.actionNet_Value_Per_Share_A.setText(_translate("MainWindow",
"Net Value Per Share (A)"))
        self.actionNet_Value_Per_Share_B.setText(_translate("MainWindow",
"Net Value Per Share (B)"))
        self.actionNet_Value_Per_Share_C.setText(_translate("MainWindow",
```

```
"Net Value Per Share (C)"))

self.actionPersistent_EPS_in_the_Last_Four_Seasons.setText(_translate("MainWi
ndow", "Persistent EPS in the Last Four Seasons"))
        self.actionCash_Flow_Per_Share.setText(_translate("MainWindow", "Cash
Flow Per Share"))
        self.actionRevenue_Per_Share_YuanY.setText(_translate("MainWindow",
"Revenue Per Share (YuanY)"))

self.actionOperating_Profit_Per_Share_YuanY.setText(_translate("MainWindow",
"Operating Profit Per Share (YuanY)"))

self.actionPer_SHare_Net_Profit_Before_Tax_YuanY.setText(_translate("MainWind
ow", "Per Share Net Profit Before Tax (YuanY)"))

self.actionRealized_Sales_Gross_Profit_Growth_Rate.setText(_translate("MainWi
ndow", "Realized Sales Gross Profit Growth Rate"))

self.actionOperating_Profit_Growth_Rate.setText(_translate("MainWindow",
"Operating Profit Growth Rate"))

self.actionAfter_tax_Net_Profit_Growth_Rate.setText(_translate("MainWindow",
"After-tax Net Profit Growth Rate"))

self.actionRegular_Net_Profit_Growth_Rate.setText(_translate("MainWindow",
"Regular Net Profit Growth Rate"))

self.actionContinuous_Net_Profit_Growth_Rate.setText(_translate("MainWindow",
"Continuous Net Profit Growth Rate"))
        self.actionTotal_Asset_Growth_Rate.setText(_translate("MainWindow",
"Total Asset Growth Rate"))
        self.actionNet_Value_Growth_Rate.setText(_translate("MainWindow",
"Net Value Growth Rate"))

self.actionTotal_Asset_Return_Growth_Rate_Ratio.setText(_translate("MainWindo
w", "Total Asset Return Growth Rate Ratio"))
        self.actionCash_Reinvestment.setText(_translate("MainWindow", "Cash
Reinvestment %"))
        self.actionCurrent_Ratio.setText(_translate("MainWindow", "Current
Ratio"))
        self.actionQuick_Ratio.setText(_translate("MainWindow", "Quick
Ratio"))
        self.actionInterest_Expense_Ratio.setText(_translate("MainWindow",
"Interest Expense Ratio"))

self.actionTotal_Debt_Total_net_Worth.setText(_translate("MainWindow", "Total
Debt / Total Net Worth"))
        self.actionDebt_Ratio.setText(_translate("MainWindow", "Debt Ratio
%"))
        self.actionNet_Worth_Assets.setText(_translate("MainWindow", "Net
Worth / Assets"))
        self.actionShape.setText(_translate("MainWindow", "Shape"))
        self.actionList.setText(_translate("MainWindow", "List"))
        self.actionSummary_Statistics.setText(_translate("MainWindow",
"Summary Statistics"))
        self.actionSize.setText(_translate("MainWindow", "Size"))
        self.actionComplete.setText(_translate("MainWindow", "Complete"))
```

```
        self.actionPositive.setText(_translate("MainWindow", "Positive"))
        self.actionNegative.setText(_translate("MainWindow", "Negative"))
        self.actionPCA_Feature_Composition.setText(_translate("MainWindow",
"PCA Feature Composition"))
        self.actionClass_Imbalance.setText(_translate("MainWindow", "Class
Imbalance"))
        self.actionDebt_Ratio_to_Bankruptcy.setText(_translate("MainWindow",
"Debt Ratio to Bankruptcy"))

self.actionDebt_Ratio_vs_Asset_Turnover.setText(_translate("MainWindow",
"Debt Ratio vs Asset Turnover"))
        self.actionGrouped_Means.setText(_translate("MainWindow", "Grouped
Means"))
        self.actionLPM_Results.setText(_translate("MainWindow", "LPM
Results"))
        self.actionProbit_Results.setText(_translate("MainWindow", "Probit
Results"))
        self.actionRaw_Data.setText(_translate("MainWindow", "Raw Data"))
        self.actionClean_Data_2.setText(_translate("MainWindow", "Clean
Data"))
        self.actionView_GitHub_Repository.setText(_translate("MainWindow",
"View GitHub Repository"))
        self.actionView_Project_Report.setText(_translate("MainWindow", "View
Project Report"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Additionally, I used the PyQt5 library to connect this GUI with our code. The code for this is as follows:

```
# Import Relevant Libraries
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PyQt5.QtWidgets import (
    QApplication, QDialog, QMainWindow, QMessageBox
)
from PyQt5.uic import loadUi
from GUI_BR import Ui_MainWindow
import webbrowser

# Initialization
raw = pd.read_csv("data.csv", index_col=0)
clean = pd.read_csv("bankrupt_clean.csv", index_col=0)
corr_mat = clean.corr('spearman')
LPM_Results = pd.read_csv('LPM_results.csv', index_col=0)
PCA_Var_Weight = pd.read_csv('PCA_VariableWeight.csv')
```

```python
PCA_Var_Weight.set_index('Variables', inplace=True)

# Operate GUI Window
class MyBR_GUI(QMainWindow,Ui_MainWindow):
    def __init__(self):
        super(MyBR_GUI,self).__init__()
        self.setupUi(self)
        #self.showMaximized()
        self.pushButton_click()
        self.aktions()
        self.klose()
        self.data_r()
        self.data_c()

# Implement pushbutton_click() function
    def pushButton_click(self):
        self.pb_PCA.clicked.connect(dpca)
        self.pB_LPM.clicked.connect(dlpm)
        self.pB_Probit.clicked.connect(dpro)
        self.pB_CHMap.clicked.connect(dchm)
        self.pb_PredModelResults.clicked.connect(dpre)

# Implement actions
    def aktions(self):
        self.actionCM_Complete.triggered.connect(dchm)
        self.actionVariables.triggered.connect(dvar)
        self.actionLPM_Results.triggered.connect(dlpm)
        self.actionProbit_Results.triggered.connect(dpro)
        self.actionVar_List.triggered.connect(dvar)
        self.actionVar_SS.triggered.connect(dvin)
        self.actionData_Shape.triggered.connect(dshp)
        self.actionData_Size.triggered.connect(dsiz)
        self.actionCM_Negative.triggered.connect(acmn)
        self.actionCM_Positive.triggered.connect(acmp)
        self.actionIP_ClassImbal.triggered.connect(ipci)
        self.actionIP_Grouped_Means.triggered.connect(ipgm)
        self.actionIP_PCA_FComp.triggered.connect(ippc)
        self.actionIP_Debt_Ratio_to_Bankruptcy.triggered.connect(ipdb)
        self.actionIP_Debt_Ratio_vs_Asset_Turnover.triggered.connect(ipda)
        self.actionView_Git.triggered.connect(vgit)
        self.actionView_Report.triggered.connect(vrep)
        self.actionView_Kaggle.triggered.connect(vkag)

# Implement Window CLosing
    def klose(self):
        self.actionClose.triggered.connect(self.close)

# Implement Dialog Window - Raw
    def data_r(self):
        self.actionRaw_Data.triggered.connect(draw)
        self.actionRaw_Data.triggered.connect(raw_dial)

# Implement Dialog Window - Clean
    def data_c(self):
        self.actionClean_Data.triggered.connect(dcle)
        self.actionClean_Data.triggered.connect(clean_dial)
```

```python
# PCA Results Viewer
def dpca():
    PCA_Var_Weight = pd.read_csv('PCA_VariableWeight.csv')
    PCA_Var_Weight.set_index('Variables', inplace=True)
    print("\n \n")
    print("PCA Variables Weight:")
    print(PCA_Var_Weight)
    print("*"*75)

# LPM Results Viewer
def dlpm():
    LPM_Results = pd.read_csv('LPM_results.csv', index_col=0)
    print("\n \n")
    print("LPM Results:")
    print(LPM_Results)
    print("*"*75)

# Probit Results Viewer
def dpro():
    Probit_Results = pd.read_csv('Probit_Results.csv', index_col=0)
    print("\n \n")
    print("Probit Results:")
    print(Probit_Results)
    print("*" * 75)

# Correlation Heatmap Viewer
def dchm():
    corr_mat = clean.corr('spearman')
    f_heatmap, ax_heatmap = plt.subplots(figsize=(28, 25))
    color_scheme = sns.diverging_palette(240, 10, as_cmap=True)
    mask = np.triu(np.ones_like(corr_mat, dtype=bool))
    sns.heatmap(corr_mat, mask=mask, cmap=color_scheme, vmax=1, center=0,
square=True, linewidths=0.5)
    plt.title('Bankruptcy Data: Correlation Heatmap', fontsize=40)
    plt.show()

# Predictive Models Results Viewer
def dpre():
    modres = pd.read_csv("models_results.csv",index_col=0)
    print("\n \n")
    print("Shallow Machine Learning Models' Results:")
    print(modres)
    print("*" * 75)

# Raw Data Head Viewer
def draw():
    print("\n \n")
    print("First 5 Rows of Raw Data:")
    print(raw.head())
    print("*" * 75)

# Raw Data MessageBox Dialog
def raw_dial():
    qm = QMessageBox()
    qm.setText("Data Imported: Raw Data")
    qm.setWindowTitle("Import Data")
    retval = qm.exec_()
```

```python
# Clean Data Head Viewer
def dcle():
    print("\n \n")
    print("First 5 Rows of Clean Data:")
    print(clean.head())
    print("*" * 75)

# Clean Data MessageBox Dialog
def clean_dial():
    qn = QMessageBox()
    qn.setText("Data Imported: Clean Data")
    qn.setWindowTitle("Import Data")
    retvall = qn.exec_()

# Var List Viewer
def dvar():
    print("\n \n")
    print("Data Variables:")
    print(raw.info())
    print("*" * 75)

# Var Info Viewer
def dvin():
    print("\n \n")
    print("Data Variables' Summary Statistics:")
    print(raw.describe())
    print("*" * 75)

# Data Shape Viewer
def dshp():
    print("\n \n")
    print("Dataset Shape:")
    print(raw.shape)
    print("*" * 75)

# Data Size Viewer
def dsiz():
    print("\n \n")
    print("Dataset Size:")
    print(raw.size)
    print("*" * 75)

# Negative Corr Map Viewer
def acmn():
    corrNeg = corr_mat.iloc[:, 0].sort_values(ascending=True).iloc[0:53]
    print("\n \n")
    print("Negative Correlations with Bankruptcy:")
    print(corrNeg)
    print("*" * 75)

# Positive Corr Map Viewer
def acmp():
    corrPos = corr_mat.iloc[:, 0].sort_values(ascending=False).iloc[0:18]
    print("\n \n")
    print("Positive Correlations with Bankruptcy:")
    print(corrPos)
```

```
    print("*" * 75)

# Imbalance Plot Viewer
def ipci():
    f_imbalance, ax_imbalance = plt.subplots(figsize=(7, 7))
    sns.countplot(clean['Bankrupt'])
    plt.title('Bankruptcy Counts \n 0 = Not Bankrupt || 1 = Bankrupt')
    plt.show()

# Grouped Means Viewer
def ipgm():
    bankrupt_means = clean.groupby('Bankrupt').mean()
    # Filter on significant variables
    means_index = LPM_Results.head(5).index
    means_index = [i for i in means_index if i != 'Constant']
    bankrupt_means = bankrupt_means.loc[:, means_index]
    # Reshape
    bankrupt_means.reset_index(inplace=True)
    bankrupt_means = pd.melt(bankrupt_means, id_vars='Bankrupt',
value_vars=means_index)
    # Plot
    f_bar, ax_bar = plt.subplots(figsize=(10, 10))
    sns.barplot(x='variable', y='value', hue='Bankrupt', data=bankrupt_means)
    plt.show()

# PCA Feature Component Viewer
def ippc():
    f_pca, ax_pca = plt.subplots(figsize=(12, 12))
    sns.heatmap(PCA_Var_Weight, cmap='bwr')
    plt.title('Visualizing Component Structure: \n What Factors Contribute to
Bankruptcy?')
    plt.show()

# Debt-Ratio-to-Bankruptcy Viewer
def ipdb():
    f_debt, ax_debt = plt.subplots(figsize=(7, 7))
    sns.scatterplot(x='Operatingprofit/Paid-incapital',
y='Totaldebt/Totalnetworth', hue='Bankrupt',
                    data=clean)
    plt.show()

# Debt-Ratio-to-Asset-Turnover Viewer
def ipda():
    f_turn, ax_turn = plt.subplots(figsize=(7, 7))
    sns.scatterplot(x='Totaldebt/Totalnetworth', y='TotalAssetTurnover',
hue='Bankrupt', data=clean)
    plt.show()

# Redirect to GitRepo
def vgit():
    webbrowser.open("https://github.com/aadusumilli87/Data-Mining-Final-
Project")

# Redirect to Git Report
def vrep():
    webbrowser.open("https://github.com/aadusumilli87/Data-Mining-Final-
Project/blob/main/Report.pdf")
```

```python
# Redirect to Kaggle Competition
def vkag():
    webbrowser.open("https://www.kaggle.com/fedesoriano/company-bankruptcy-prediction/code")

# GUI RUNNER
if __name__ == '__main__':
    app = QApplication(sys.argv)
    my_pyqt_form = MyBR_GUI()
    my_pyqt_form.show()
    sys.exit(app.exec_())
```

4. Results

Below, I report the results from our seven statistical models., which are the Principal Component Analysis, the Logistic regression, the Probit regression, Decision Tree Classifier, Multi-Layer Perceptron Classifier, Histogram-based Gradient Boosting Regressor, and the Linear Probability Model.

The LPM results are below, sorted by magnitude:

Variable	Beta	Standard Error	T-Stat	P-values
Totaldebt/Totalnetworth	0.056928514	0.006524709	8.725066282	2.66E-18
Constant	0.032262795	0.001935207	16.67149746	2.11E-62
Operatingprofit/Paid-incapital	0.031497086	0.00615543	5.116959648	3.11E-07
TotalAssetTurnover	0.020661977	0.005881382	3.513115917	0.000442884
Totalexpense/Assets	0.01997927	0.005956478	3.35420888	0.000795923
Cash/CurrentLiability	0.018490886	0.00730577	2.530997459	0.011373867
NetValuePerShare(C)	0.017618261	0.003850694	4.575346833	4.75E-06
Revenueperperson	0.014133286	0.004593376	3.076883962	0.002091767
Liability-AssetsFlag	0.010022477	0.004268733	2.347881013	0.018880552
Contingentliabilities/Networth	0.009314309	0.001136143	8.198180515	2.44E-16
CurrentLiabilitytoLiability	0.00620577	0.002704328	2.294754419	0.021747202
Totalincome/Totalexpense	0.001790716	0.000767261	2.333906765	0.019600604

TotalAssetGrowthRate	-0.005394018	0.00183532	-2.93900705	0.003292656
Taxrate(A)	-0.006388254	0.001961173	-3.257364168	0.001124521
NetValueGrowthRate	-0.007907885	0.002270913	-3.482249769	0.00049722
AccountsReceivableTurnover	-0.010650307	0.002696245	-3.950052333	7.81E-05
Operatingprofitperperson	-0.014840975	0.003995807	-3.714137545	0.000203898
Netprofitbeforetax/Paid-incapital	-0.021578202	0.007452767	-2.895327463	0.003787632
ROA(B)beforeinterestanddepreciationaftertax	-0.026042856	0.007405364	-3.516755642	0.000436856
RevenuePerShare(YuanÂ¥)	-0.049013524	0.006823678	-7.182860249	6.83E-13

The results from the LPM in the table are for variables that are standardized, which allows us to compare the magnitudes against each other. However, it is important to note that none of our variables are insignificant at the 95% significance level (alpha = 5%) are included here. Explaining the most impactful variable here, a one-standard deviation increase in the (Total Debt)/(Total Net Worth) increases the possibility if a firm declaring bankruptcy by nearly 5.7 percentage points. he other coefficients have the same interpretation. We see that in general, debt and liability ratios increase the probability of bankruptcy, while value/profitability metrics decrease the probability of bankruptcy, which is logical. However, there are some curious results from this estimation – we see that Net Value per Share and Asset Turnover have positive coefficients, implying that as they increase, the probability of bankruptcy increases. This may be the result of omitted variables bias – indeed, it is easy to reason that an intangible omitted variable such as ‘Executive Competence’ would surely influence the probability of bankruptcy.

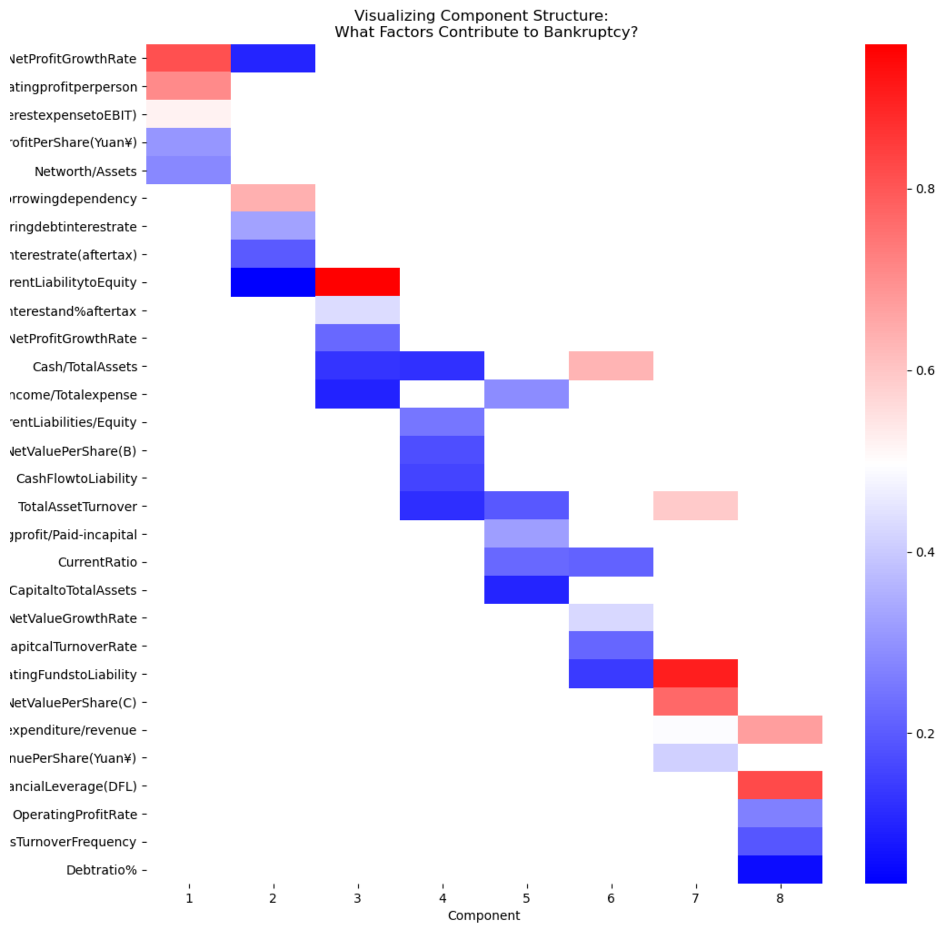
Our Probit Model results are as follows:

Variable	Parameters	T-Stat	P-values
Operatingprofit/Paid-incapital	0.467751671	3.192098992	0.001412429
Totaldebt/Totalnetworth	0.374387096	7.561067932	4.00E-14
Revenueperperson	0.185454735	3.111594243	0.001860801
GrossProfittoSales	0.097805344	2.107706823	0.035056357
AverageCollectionDays	0.087668754	2.009843246	0.044447782

Cash/CurrentLiability	0.079282144	2.459584177	0.013909807
Cashflowrate	-0.16056375	-2.572506752	0.010096499
QuickRatio	-0.212959966	-2.638603834	0.00832482
Cash/TotalAssets	-0.23139262	-2.152194726	0.031382021
AccountsReceivableTurnover	-0.24360273	-4.33921339	1.43E-05
RevenuePerShare(YuanÂ¥)	-0.302684324	-2.179360221	0.029304919
ROA(B)beforeinterestanddepreciationaftertax	-0.323479296	-3.803013532	0.000142947
Netprofitbeforetax/Paid-incapital	-0.529058932	-3.341383889	0.000833619
Constant	-2.797022563	-24.80639267	7.65E-136

We see that the most important determinant of bankruptcy in the Probit is (Operating Profit)/(Paid-in Capital), with a coefficient of approximately 0.467. This means that a one standard deviation increase in (Operating Profit/Paid-in Capital) is estimated to increase the Z-score by 0.467. As the Probit is a nonlinear model, the actual partial effect of this increase will change depending on where in the distribution the initial Z-score falls. Unfortunately, interpretation is further complicated by omitted variable bias, more severe in the probit than in the LPM. This is evidenced by variables such Revenue per Person having a positive coefficient, implying an increased probability of bankruptcy as Revenue per Person increases, which is nonsensical. The reason for this increased bias is that, unlike in OLS regression with the LPM, omitting features which are correlated with the outcome but not with the other predictors leads to bias in coefficient estimates of the predictors.

Our Principal Component Analysis suggests that without standardizing the data, just eight components alone can explain 97% of the variation in the data. The deconstruction of the components is as follows:



We see that each component is mostly comprised of a group of related variables. The first component is comprised of profitability metrics, the fifth component is mostly comprised of liquidity metrics, and so on. While the PCA did not feature heavily into our analysis, it provides an intriguing path for future analysis.

The results of our four machine learning models are as follows:

F1 Score	Estimator
0.75	Decision Tree Classifier
0.74	MLP Classifier
0.74	Histogram Gradient Boosting Classifier
0.61	Logistic Regression

We find that the decision tree classifier had the best performance, closely followed by the MLP classifier and the Histogram Gradient Boosting Classifier. Logistic Regression performed notably worse than the other models. All model parameters were tuned using Grid SearchCV.

5. Conclusion

The results of our inferential and predictive models suggest that debt indicators tend to increase the probability of a firm declaring bankruptcy, and these effects are stronger than the negative effects of the probability indices on bankruptcy. It is important to note that our inferential models suffer from an omitted variable bias, which may lead to spurious results.

Our predictive models show that a Random tree Classifier explains our data with 75% accuracy, and can be further trained to understand the determinants of bankruptcy in similar datasets. This data also suffers from two collection problems: high multicollinearity and class imbalance. We try to address these issues, but find high collinearity still persists in our data.

Future models can build on these results, and use our findings to hyper-tune the parameters of their predictive models to better build models that gauge bankruptcy based on these factors.

6. Code Percentage

Total code lines without blank lines or comments: 674
Total lines of code found on internet, other sources: 76
Total lines of code modified (that were found): 8
Total lines of code added (own): 598

Percentage: $\frac{(76 - 8) * 100}{(76 + 598)} = 9.79\%$.

7. References

Altman, E. I. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *The journal of finance*, *23*(4), 589-609.

Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." *Journal of artificial intelligence research* 16 (2002): 321-357.

Lee, Tsun-Siou, and Yin-Hua Yeh. "Corporate governance and financial distress: Evidence from Taiwan." (2004): 378-388.

Liang, Deron, Chia-Chi Lu, Chih-Fong Tsai, and Guan-An Shih. "Financial ratios and corporate governance indicators in bankruptcy prediction: A comprehensive study." *European Journal of Operational Research* 252, no. 2 (2016): 561-572.

Lin, Wei-Yang, Ya-Han Hu, and Chih-Fong Tsai. "Machine learning in financial crisis prediction: a survey." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, no. 4 (2011): 421-436.

Shailer, Gregory EP. *An introduction to corporate governance in Australia*. Pearson Education Australia, 2004.