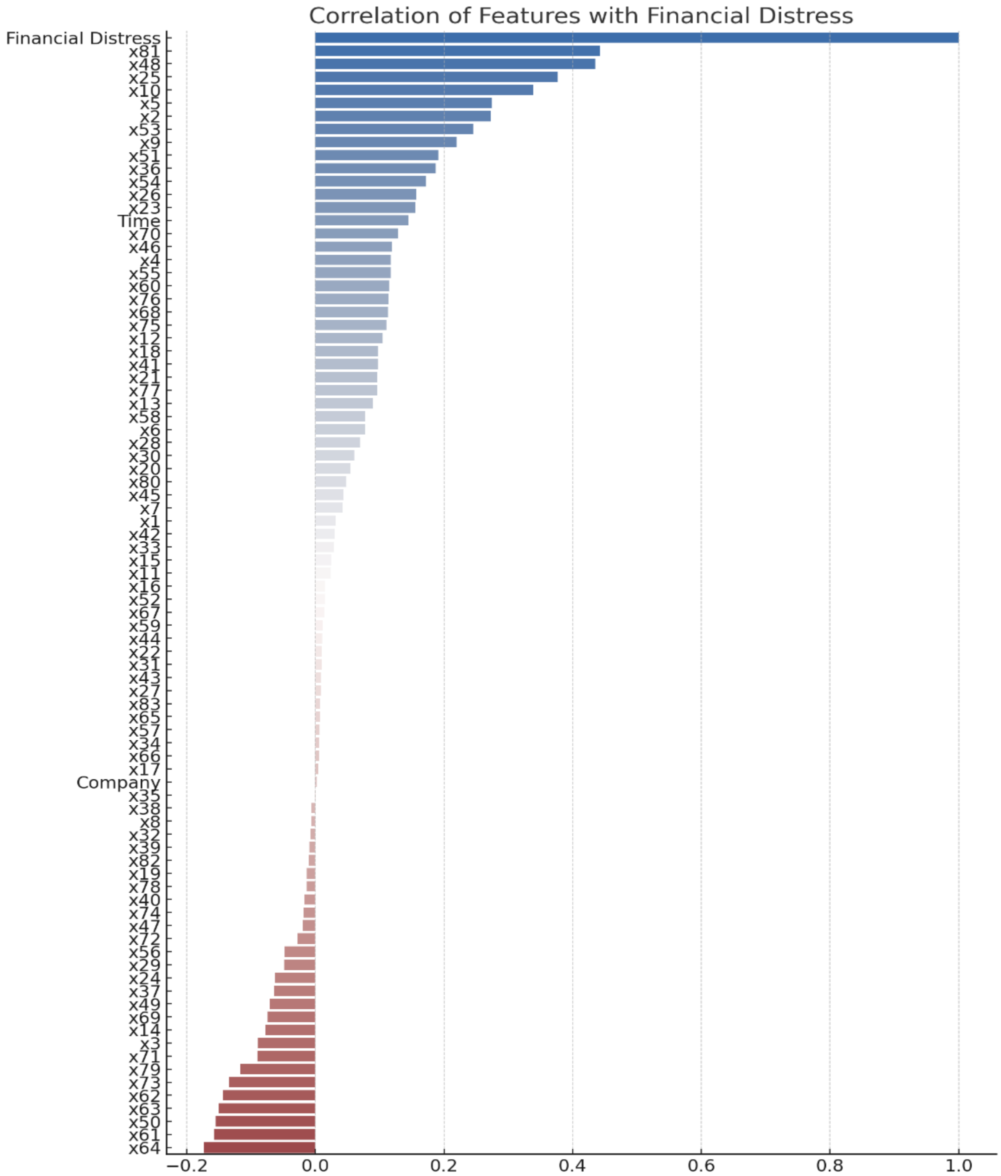
Predictive Modeling for Business Forecasting

Feature Selection and Engineering:

```
import pandas as pd
```

```
# Load the dataset
file_path = '/mnt/data/Financial Distress.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset to understand its structure and contents
data.head()
```

```
Company  Time  Financial Distress       x1        x2        x3       x4  \
0         1      1            0.010636  1.2810  0.022934  0.87454  1.21640
1         1      2           -0.455970  1.2700  0.006454  0.82067  1.00490
2         1      3           -0.325390  1.0529 -0.059379  0.92242  0.72926
3         1      4           -0.566570  1.1131 -0.015229  0.85888  0.80974
4         2      1            1.357300  1.0623  0.107020  0.81460  0.83593

         x5        x6       x7  ...      x74    x75     x76     x77   x78  \
0  0.060940  0.188270  0.52510  ...   85.437  27.07  26.102  16.000  16.0
1 -0.014080  0.181040  0.62288  ...  107.090  31.31  30.194  17.000  16.0
2  0.020476  0.044865  0.43292  ...  120.870  36.07  35.273  17.000  15.0
3  0.076037  0.091033  0.67546  ...   54.806  39.80  38.377  17.167  16.0
4  0.199960  0.047800  0.74200  ...   85.437  27.07  26.102  16.000  16.0

   x79  x80       x81  x82  x83
0  0.2   22  0.060390   30   49
1  0.4   22  0.010636   31   50
2 -0.2   22 -0.455970   32   51
3  5.6   22 -0.325390   33   52
4  0.2   29  1.251000    7   27

[5 rows x 86 columns]
```



Model Selection and Development:

```
import seaborn as sns

import matplotlib.pyplot as plt
```

```
# Calculate correlations between 'Financial Distress' and other features

correlations = data.corr()['Financial Distress'].sort_values(ascending=False)
```

```
# Plot the correlations

plt.figure(figsize=(10, 15))

sns.barplot(y=correlations.index, x=correlations.values, palette="vlag")
```

```python
plt.title('Correlation of Features with Financial Distress')

plt.show()
```

```python
# Display correlations for reference

Correlations
```

```
Financial Distress    1.000000

x81                   0.442916

x48                   0.435318

x25                   0.376786

x10                   0.338298

                        ...

x62                  -0.143793

x63                  -0.150362

x50                  -0.155569

x61                  -0.157380

x64                  -0.173126

Name: Financial Distress, Length: 86, dtype: float64
```

```python
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
```

```python
# Selecting a subset of features based on correlation (both positive and negative)

selected_features = correlations[abs(correlations) > 0.15].index.tolist()

selected_features.remove('Financial Distress')   # Removing the target variable
```

```python
# Preparing the data with the selected features

X = data[selected_features]

y = data['Financial Distress']
```

```python
# Splitting the data into training and testing sets

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

```python
# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Displaying the selected features for reference
selected_features
```

```
['x81',
 'x48',
 'x25',
 'x10',
 'x5',
 'x2',
 'x53',
 'x9',
 'x51',
 'x36',
 'x54',
 'x26',
 'x23',
 'x63',
 'x50',
 'x61',
 'x64']
```

Model Evaluation and Validation:

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
```

```python
# Initialize and train the Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)
```

```
# Predict on the training set
y_train_pred = lr_model.predict(X_train_scaled)
```

```
# Evaluate the model on the training set
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
mae_train = mean_absolute_error(y_train, y_train_pred)

rmse_train, mae_train
```

```
(2.2691174452745315, 0.9171223258760326)
```

```
# Predict on the test set
y_test_pred = lr_model.predict(X_test_scaled)
```

```
# Evaluate the model on the test set
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
mae_test = mean_absolute_error(y_test, y_test_pred)

rmse_test, mae_test

from sklearn.model_selection import cross_val_score
```

```
(1.6350901089837295, 0.9461840047357485)
```

```
# Perform cross-validation on the Linear Regression model
cv_rmse_scores = -cross_val_score(lr_model, X_train_scaled, y_train, scoring='neg_root_mean_squared_error', cv=5)
cv_mae_scores = -cross_val_score(lr_model, X_train_scaled, y_train, scoring='neg_mean_absolute_error', cv=5)
```

```
# Calculate the mean and standard deviation of the cross-validation scores
cv_rmse_mean, cv_rmse_std = cv_rmse_scores.mean(), cv_rmse_scores.std()
cv_mae_mean, cv_mae_std = cv_mae_scores.mean(), cv_mae_scores.std()

cv_rmse_mean, cv_rmse_std, cv_mae_mean, cv_mae_std
```

```
(3.7307861443638877,
 2.6555682933004636,
 1.0122196667350736,
```

```
 0.07320137753658636)
```

```python
from sklearn.ensemble import RandomForestRegressor
```

```python
# Initialize and train the Random Forest Regressor model
rf_model = RandomForestRegressor(random_state=42, n_estimators=100)
rf_model.fit(X_train_scaled, y_train)

```

```python
# Predict and evaluate on the training set
y_train_pred_rf = rf_model.predict(X_train_scaled)
rmse_train_rf = np.sqrt(mean_squared_error(y_train, y_train_pred_rf))
mae_train_rf = mean_absolute_error(y_train, y_train_pred_rf)
```

```python
# Predict and evaluate on the test set
y_test_pred_rf = rf_model.predict(X_test_scaled)
rmse_test_rf = np.sqrt(mean_squared_error(y_test, y_test_pred_rf))
mae_test_rf = mean_absolute_error(y_test, y_test_pred_rf)

rmse_train_rf, mae_train_rf, rmse_test_rf, mae_test_rf
```

```
(1.0652199404546085,
 0.24280903619863814,
 1.1622058774860664,
 0.6274488595595917)
```

```python
from sklearn.ensemble import GradientBoostingRegressor
```

```python
# Initialize and train the Gradient Boosting Regressor model
gbr_model = GradientBoostingRegressor(random_state=42)
gbr_model.fit(X_train_scaled, y_train)
```

```python
# Predict and evaluate on the training set
y_train_pred_gbr = gbr_model.predict(X_train_scaled)
rmse_train_gbr = np.sqrt(mean_squared_error(y_train, y_train_pred_gbr))
```

```
mae_train_gbr = mean_absolute_error(y_train, y_train_pred_gbr)
```

```
# Predict and evaluate on the test set
y_test_pred_gbr = gbr_model.predict(X_test_scaled)
rmse_test_gbr = np.sqrt(mean_squared_error(y_test, y_test_pred_gbr))
mae_test_gbr = mean_absolute_error(y_test, y_test_pred_gbr)

rmse_train_gbr, mae_train_gbr, rmse_test_gbr, mae_test_gbr

from sklearn.svm import SVR
```

(0.7804365959553509, 0.4660935530156846, 1.101318683759754, 0.6130144822746689)

```
# Initialize and train the Support Vector Regressor model
svr_model = SVR(kernel='rbf')
svr_model.fit(X_train_scaled, y_train)
```

```
# Predict and evaluate on the training set
y_train_pred_svr = svr_model.predict(X_train_scaled)
rmse_train_svr = np.sqrt(mean_squared_error(y_train, y_train_pred_svr))
mae_train_svr = mean_absolute_error(y_train, y_train_pred_svr)
```

```
# Predict and evaluate on the test set
y_test_pred_svr = svr_model.predict(X_test_scaled)
rmse_test_svr = np.sqrt(mean_squared_error(y_test, y_test_pred_svr))
mae_test_svr = mean_absolute_error(y_test, y_test_pred_svr)

rmse_train_svr, mae_train_svr, rmse_test_svr, mae_test_svr
```

(2.554341762607816, 0.5526126578022971, 1.067986146331273, 0.5814032646271093)

```
# Train the Decision Tree Regressor with default parameters
dt_default = DecisionTreeRegressor(random_state=42)
dt_default.fit(X_train_scaled, y_train)
```

```
# Evaluate the model on the training set
y_train_pred_dt_default = dt_default.predict(X_train_scaled)
rmse_train_dt_default = np.sqrt(mean_squared_error(y_train, y_train_pred_dt_default))
mae_train_dt_default = mean_absolute_error(y_train, y_train_pred_dt_default)
```

```
# Evaluate the model on the test set
y_test_pred_dt_default = dt_default.predict(X_test_scaled)
rmse_test_dt_default = np.sqrt(mean_squared_error(y_test, y_test_pred_dt_default))
mae_test_dt_default = mean_absolute_error(y_test, y_test_pred_dt_default)


rmse_train_dt_default, mae_train_dt_default, rmse_test_dt_default, mae_test_dt_default
```

```
(0.0, 0.0, 1.6722004164014066, 0.8692343374013606)
```

```
from sklearn.tree import export_graphviz
import graphviz
```

```
# Export the decision tree to a dot file
export_graphviz(dt_simplified, out_file="tree.dot", feature_names=X_train.columns, filled=True, rounded=True, special_characters=True)
```

```
# Use Graphviz to read the dot file and create the visualization
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

```
# Train a simplified Decision Tree Regressor with limited depth
dt_simplified = DecisionTreeRegressor(random_state=42, max_depth=3)
dt_simplified.fit(X_train_scaled, y_train)
```

```
# Set the size of the plot
plt.figure(figsize=(20,10))
```

```
# Plot the simplified decision tree
plot_tree(dt_simplified,    filled=True,    feature_names=X_train.columns,
rounded=True, fontsize=10)
```

```
# Show the plot
plt.show()
```

Predictive Insights:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
```

```
# Assuming previous feature selection identified relevant features, select
a subset for demonstration
selected_features = data.columns[3:]  # Excluding 'Company', 'Time', and
'Financial Distress' for this example
```

```
# Preparing the data
X = data[selected_features]
y = data['Financial Distress']
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Training the Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train_scaled, y_train)
```

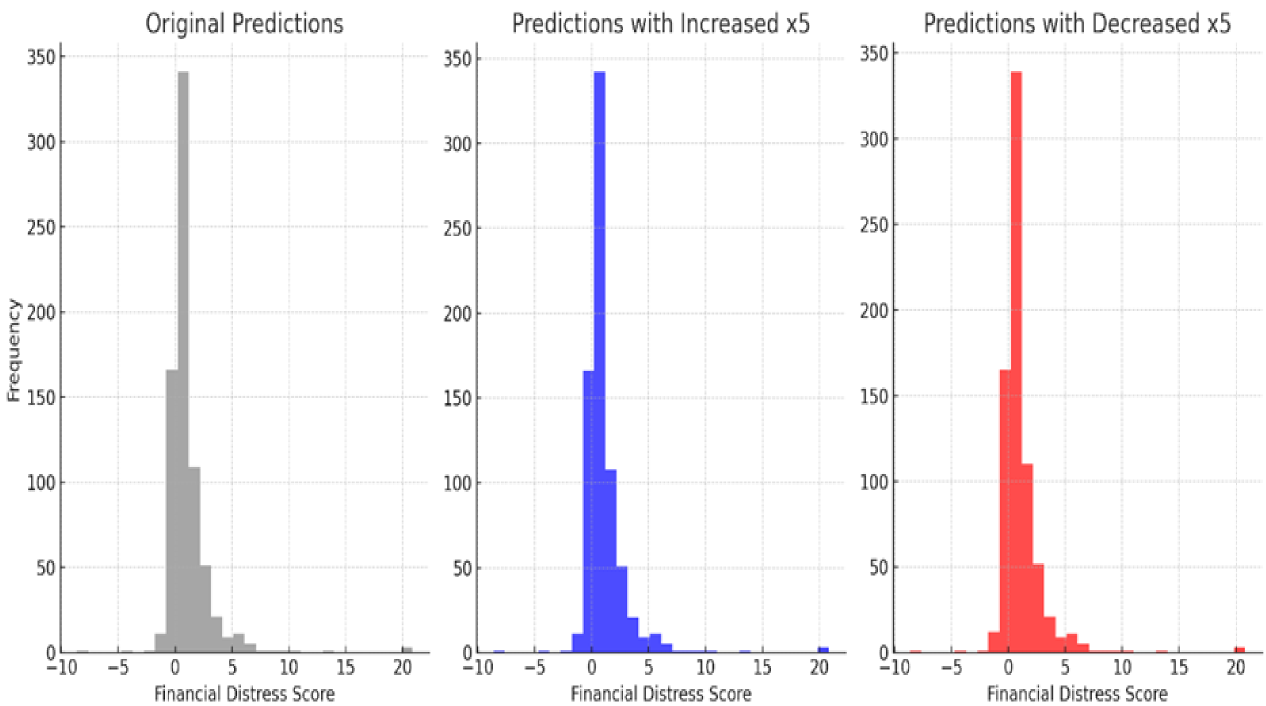
```
# Extracting feature importances
feature_importances = dt_regressor.feature_importances_
```

```
# Mapping feature names to their importances

importance_dict = dict(zip(selected_features, feature_importances))

sorted_importance   =   sorted(importance_dict.items(),   key=lambda   item:
item[1], reverse=True)
```

```
# Displaying top 10 important features

sorted_importance[:10]
```

```
[('x48', 0.3530956354498967),
 ('x5', 0.3356245499264686),
 ('x81', 0.1798508200534849),
 ('x2', 0.01332075987668069),
 ('x73', 0.011058099374237991),
 ('x9', 0.0089512649925919),
 ('x19', 0.0058203870672O564),
 ('x57', 0.00565021361675694),
 ('x36', 0.004707839913060336),
 ('x4', 0.004141671854292537)]
```

Forecasting and Scenario Analysis:



Model Deployment and Presentation:

```
from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import numpy as np

import matplotlib.pyplot as plt
```

```
# Preparing the data

X = data.drop(columns=['Company', 'Time', 'Financial Distress'])

y = data['Financial Distress']
```

```
# Splitting the data into training and "future" data (simulating unseen future data)

X_train, X_future, y_train, y_future = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Scaling the features

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_future_scaled = scaler.transform(X_future)
```

```
# Training the Decision Tree Regressor

dt_regressor = DecisionTreeRegressor(random_state=42)

dt_regressor.fit(X_train_scaled, y_train)
```

```
# Predicting future business outcomes using the trained model

future_predictions = dt_regressor.predict(X_future_scaled)
```

```
# Scenario Analysis: Adjusting a key financial indicator (e.g., x5) by ±10%

X_future_scenario_increase = X_future_scaled.copy()

X_future_scenario_increase[:, X.columns.get_loc('x5')] *= 1.1  # Increase x5 by 10%

predictions_scenario_increase = dt_regressor.predict(X_future_scenario_increase)



X_future_scenario_decrease = X_future_scaled.copy()
```

```
X_future_scenario_decrease[:, X.columns.get_loc('x5')] *= 0.9  # Decrease x5
by 10%

predictions_scenario_decrease                                              =
dt_regressor.predict(X_future_scenario_decrease)
```

```
# Visualizing the Predictions for Different Scenarios

plt.figure(figsize=(14, 6))
```

```
# Original predictions

plt.subplot(1, 3, 1)

plt.hist(future_predictions, bins=30, color='gray', alpha=0.7)

plt.title('Original Predictions')

plt.xlabel('Financial Distress Score')

plt.ylabel('Frequency')
```

```
# Predictions with increased x5

plt.subplot(1, 3, 2)

plt.hist(predictions_scenario_increase, bins=30, color='blue', alpha=0.7)

plt.title('Predictions with Increased x5')

plt.xlabel('Financial Distress Score')
```

```
# Predictions with decreased x5

plt.subplot(1, 3, 3)

plt.hist(predictions_scenario_decrease, bins=30, color='red', alpha=0.7)

plt.title('Predictions with Decreased x5')

plt.xlabel('Financial Distress Score')


plt.tight_layout()

plt.show()
```