Intelligent Supplier Selection - Capstone Project

By Sourajit Ghosh

Contents

Files & Code Repository

All codes and raw data files are uploaded here:
https://github.com/sourajitaghosh/DataScienceSupplierRiskPrediction

Also Python Code uploaded here too:

https://www.coursera.org/learn/iitr-intelligent-supply-chain-strategies/ungradedLab/sHdRr/jupyter-lab-for-python/lab?path=%2Fnotebooks%2FSupplierDataScienceCapstone_SourajitGhosh_Final.ipynb

Situation Analysis & Problem Statement

Consider a multinational corporation that operates in multiple countries and has several departments. The company is looking to standardize its supply chain operations and improve its overall efficiency. To achieve this, it has decided to select suppliers for the goods it requires. The goal of the multinational corporation is to choose the best supplier on various criteria such as cost, quality, delivery time, and reliability. At the same time, the company would like to minimize the risk. This project will ask you to predict supplier's risk using supervised learning (classification) paradigm and incorporate this in an optimization problem to finally make efficient supplier selection.

Summary Methodology

1. Detailed notes and explanations are provided as comments in each step in the python code notebook uploaded.
2. Detailed data exploration analysis was done using the existing dataset along-with data management (missing data, data standardization & categorization, data balancing, etc).
3. Performance Metrics for Classification Models was done (combination of Accuracy, Sensitivity, Specificity, Precision, F1 Score, Probability Threshold, AUC…ROC Curve)
4. The classifier prediction model thus selected was used in the new supplier data to predict the probability of supplier default
5. In a different analysis the supplier selection plan was computed on the Vivo Supplier dataset – and using linear programming and combining the supplier prediction probabilistic model – the calculation was done on the new supplier selection
6. Further sensitivity analysis was done using varying cost rates

Since this is an executive report, only key outputs and analysis are summarized below in this report. For detailed comments and insights into actual code written, please refer to the Jupyter Notebook file submitted along-with the document

Statistics insights

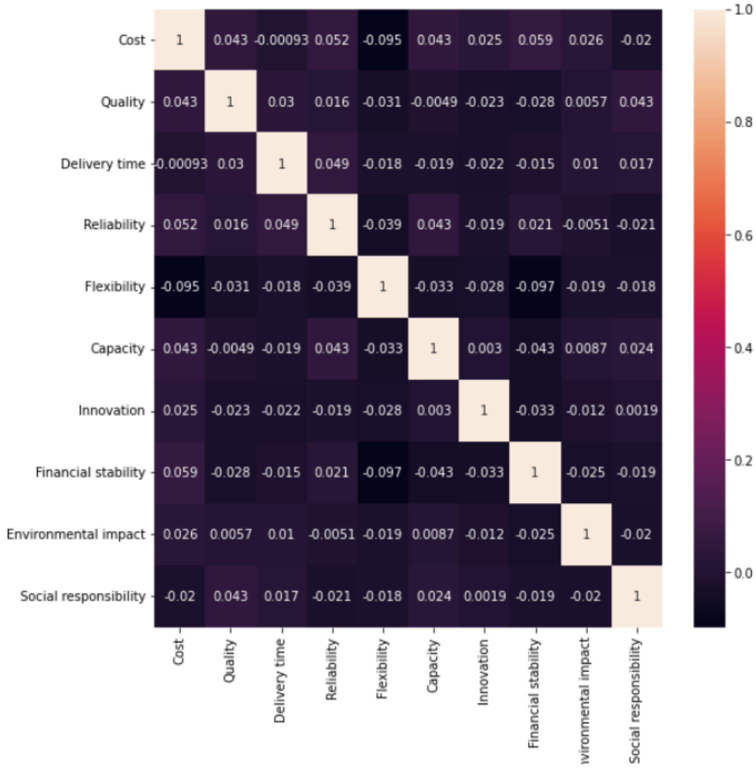
```
#To get statistics of the data
dtf.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Supplier ID	1000.0	1500.500000	288.819436	1001.000	1250.75000	1500.5000	1750.25000	2000.000
Cost	1000.0	0.487184	0.289946	0.001	0.23650	0.4815	0.73525	0.999
Quality	999.0	0.510847	0.283552	0.001	0.27600	0.5060	0.75300	0.994
Delivery time	1000.0	0.500643	0.294141	0.000	0.23300	0.5030	0.76175	0.999
Reliability	998.0	0.513186	0.285107	0.003	0.27100	0.5115	0.76075	0.998
Flexibility	998.0	0.503326	0.294474	0.003	0.24400	0.5110	0.76200	0.997
Capacity	999.0	0.505993	0.286334	0.001	0.26200	0.5080	0.74600	1.000
Innovation	998.0	0.518637	0.286429	0.001	0.27250	0.5295	0.76775	0.998
Financial stability	1000.0	0.491741	0.290615	0.000	0.23000	0.4870	0.74225	0.998
Environmental impact	1000.0	0.513256	0.290597	0.001	0.25900	0.5300	0.76725	0.999
Social responsibility	1000.0	0.506644	0.290358	0.000	0.24575	0.5175	0.74825	0.999
Safety record	1000.0	0.500012	0.291850	0.000	0.24000	0.5150	0.75000	1.000
Technical expertise	1000.0	0.504658	0.294110	0.001	0.24775	0.4955	0.76825	0.998
Customer service	1000.0	0.497444	0.284664	0.001	0.25375	0.4875	0.75000	1.000
Lead time	1000.0	0.494643	0.283233	0.000	0.25075	0.4940	0.73425	1.000
Responsiveness	999.0	0.489079	0.290497	0.001	0.22950	0.4860	0.74950	0.998
Contract terms	999.0	0.500069	0.293173	0.002	0.24050	0.5080	0.76450	1.000
Diversity	1000.0	0.501644	0.285467	0.000	0.26375	0.5075	0.74425	0.999
Sustainability	999.0	0.500696	0.294314	0.001	0.23950	0.4960	0.74600	0.999
Scalability	999.0	0.497949	0.292739	0.002	0.24900	0.4870	0.76700	1.000
Supplier’s default	1000.0	1.272000	0.445213	1.000	1.00000	1.0000	2.00000	2.000

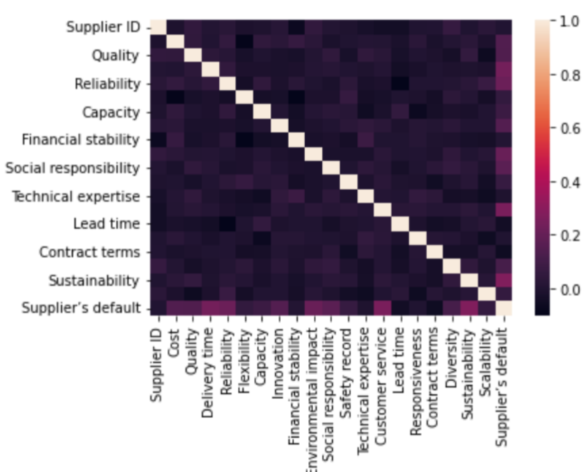
Correlation Matrix

```
# Correlation matrix

plt.figure(figsize=(9,9))
sns.heatmap(dtf.iloc[:,1:12].corr(),yticklabels=True,annot=True)
```

```
<AxesSubplot:>
```

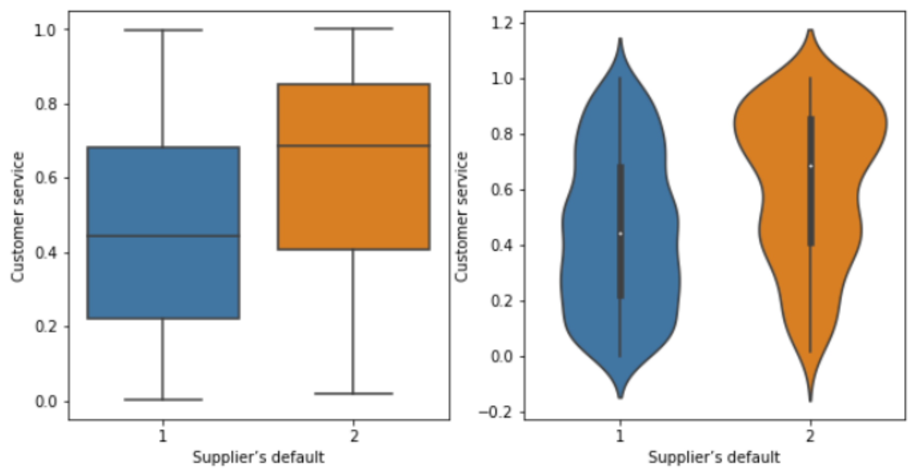



```
[15] sns.heatmap(cormat);
```

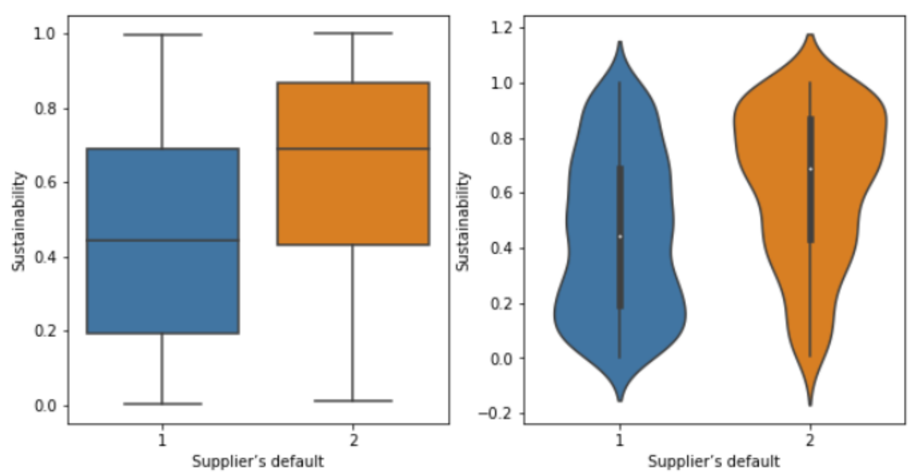


Box Plot & Violin Plot

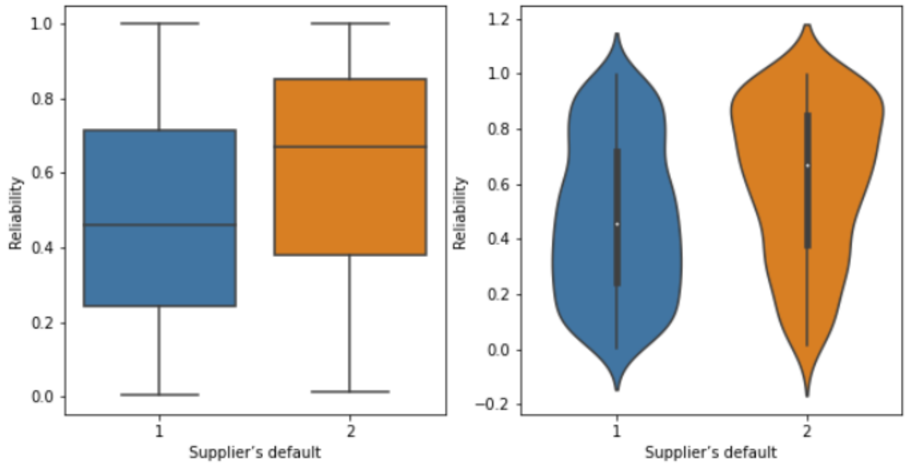
```python
# Plotting correlation between supplier default & customer survice

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
sns.boxplot(x="Supplier’s default", y="Customer service", data=dtf)
plt.subplot(1,2,2)
sns.violinplot(x="Supplier’s default", y="Customer service", data=dtf)
plt.show()
```




```python
# Plotting correlation between supplier default & customer survice

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
sns.boxplot(x="Supplier’s default", y="Sustainability", data=dtf)
plt.subplot(1,2,2)
sns.violinplot(x="Supplier’s default", y="Sustainability", data=dtf)
plt.show()
```



```python
# Plotting correlation between supplier default & customer survice

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
sns.boxplot(x="Supplier’s default", y="Reliability", data=dtf)
plt.subplot(1,2,2)
sns.violinplot(x="Supplier’s default", y="Reliability", data=dtf)
plt.show()
```



given the data, we drilled
down more into 1) Customer Service 2) Sustainabilty 3) Reliabilty

Above analyis shows Supplier Default increases when these 3 reduces: Customer Service, Sustainabilty, Reliabilty

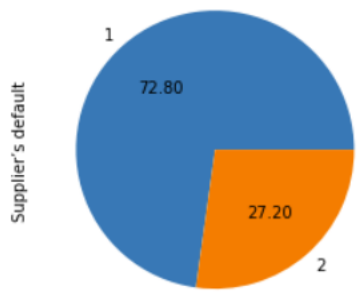
Data imbalance, missing data, hot encoding

```
X1 = dtf.drop(['Supplier’s default'], axis=1)
y1 = dtf['Supplier’s default']
supplierdefault_count = y1.value_counts()
print(supplierdefault_count)
```

```
1    728
2    272
Name: Supplier’s default, dtype: int64
```

```
[22] y1.value_counts().plot.pie(autopct='%.2f')
```

```
<AxesSubplot:ylabel='Supplier’s default'>
```



0s

```python
#Reputation is the only column with non-numeric data
dtf['Reputation'].value_counts()
```

```
Bad          261
Excellent    258
Average      241
Good         238
Name: Reputation, dtype: int64
```

0s [25]

```python
#importing libraries
from sklearn.preprocessing import OneHotEncoder


# Converting type of columns to category
dtf['Reputation']=dtf['Reputation'].astype('category')

#Assigning numerical values and storing it in another columns
dtf['Reputation_new']=dtf['Reputation'].cat.codes

#Create an instance of One-hot-encoder
enc=OneHotEncoder()

#Passing encoded columns
'''
NOTE: we have converted the enc.fit_transform() method to array because the fit_transform method
of OneHotEncoder returns SpiPy sparse matrix this enables us to save space when we
have huge number of categorical variables
'''
enc_data=pd.DataFrame(enc.fit_transform(dtf[['Reputation_new']]).toarray())

#Merge with main
New_df=dtf.join(enc_data)

print(New_df)
```

```
[29] #In table Reputation New, chnage Bad to 10, Average to 20, Good to 30, Excellent to 40
dtf2['Reputation_new']= dtf2['Reputation_new'].replace(1, 10)
dtf2['Reputation_new']= dtf2['Reputation_new'].replace(3, 30)
dtf2['Reputation_new']= dtf2['Reputation_new'].replace(2, 40)
dtf2['Reputation_new']= dtf2['Reputation_new'].replace(0, 20)
dtf2.head()
```

	Supplier ID	Cost	Quality	Delivery time	Reliability	Flexibility	Capacity	Innovation	Reputation	Fi st
0	1001	0.874	0.758	0.082	0.114	0.082	0.871	0.240	Bad	
1	1002	0.972	0.694	0.955	0.963	0.244	0.849	0.001	Good	
2	1003	0.827	0.413	0.199	0.577	0.127	0.232	0.476	Excellent	
3	1004	0.422	0.555	0.969	0.760	0.773	0.011	0.358	Average	
4	1005	0.767	0.958	0.915	0.719	0.715	0.553	0.651	Bad	

5 rows × 28 columns

missing values & hot encoding

+ Code + Text

```
[34] dtf3.head()
```

	Supplier ID	Cost	Quality	Delivery time	Reliability	Flexibility	Capacity	Innovation	Financial stability	Environme im
0	1001	0.874	0.758	0.082	0.114	0.082	0.871	0.240	0.478	(
1	1002	0.972	0.694	0.955	0.963	0.244	0.849	0.001	0.713	(
2	1003	0.827	0.413	0.199	0.577	0.127	0.232	0.476	0.805	(
3	1004	0.422	0.555	0.969	0.760	0.773	0.011	0.358	0.847	(
4	1005	0.767	0.958	0.915	0.719	0.715	0.553	0.651	0.578	(

5 rows × 22 columns

```
[35] dtf3.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
```

```
dtf5.head()
```

	Supplier ID	Cost	Quality	D
0	1001	0.874	0.758	
1	1002	0.972	0.694	
2	1003	0.827	0.413	
3	1004	0.422	0.555	
4	1005	0.767	0.958	

5 rows × 22 columns

```
[40] dtf5.isnull().values.any()
```

```
False
```

```
[41] dtf5.isnull().values.sum()
```

```
0
```

	Supplier ID	Cost	Quality	Delivery time	Reliability	Flexibility	Capacity	Innovation	Financial stability	Environmental impact	...	Technical expertise	Customer service	Lead time	Responsiveness	Contract terms	Diversity	Sustainability	Scalability	Reputation_new	Supplier's default
0	1001	0.874	0.758	0.082	0.114	0.082	0.871	0.240	0.478	0.806	...	0.947	0.463	0.969	0.044	0.812	0.044	0.853	0.663	10	1
1	1002	0.972	0.694	0.955	0.963	0.244	0.849	0.001	0.713	0.776	...	0.242	0.867	0.523	0.583	0.968	0.490	0.515	0.028	30	2
2	1003	0.827	0.413	0.199	0.577	0.127	0.232	0.476	0.805	0.465	...	0.192	0.446	0.145	0.568	0.081	0.822	0.826	0.158	40	2
3	1004	0.422	0.555	0.969	0.760	0.773	0.011	0.358	0.847	0.091	...	0.280	0.699	0.863	0.423	0.271	0.960	0.469	0.914	20	1
4	1005	0.767	0.958	0.915	0.719	0.715	0.553	0.651	0.578	0.989	...	0.131	0.406	0.963	0.939	0.981	0.004	0.013	0.575	10	1

5 rows × 22 columns

Feature extraction

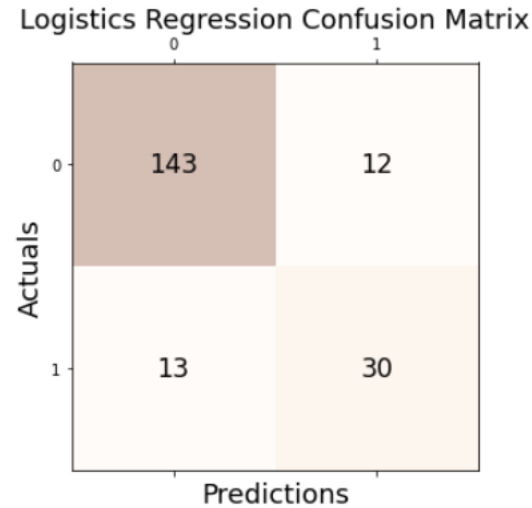
```
#Extracting the feature into X,
X = dtf5.iloc[:, 1:-1].values

#Extracting the target into y
y = dtf5.iloc[:, -1].values
print(X)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)
```

```
[[8.74e-01 7.58e-01 8.20e-02 ... 8.53e-01 6.63e-01 1.00e+01]
 [9.72e-01 6.94e-01 9.55e-01 ... 5.15e-01 2.80e-02 3.00e+01]
 [8.27e-01 4.13e-01 1.99e-01 ... 8.26e-01 1.58e-01 4.00e+01]
 ...
 [9.10e-01 3.08e-01 3.81e-01 ... 6.90e-02 4.03e-01 2.00e+01]
 [4.08e-01 2.71e-01 9.21e-01 ... 8.63e-01 9.59e-01 1.00e+01]
 [5.18e-01 3.19e-01 3.44e-01 ... 4.99e-01 5.41e-01 2.00e+01]]
```

Logistic Regression

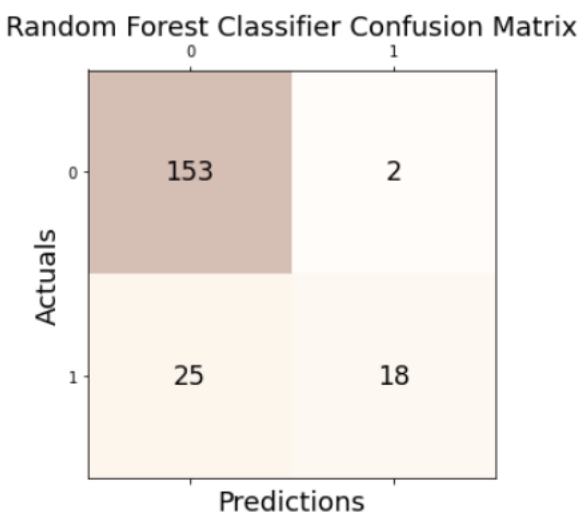
Logistic regression is the right algorithm to start with classification algorithms. It uses a logistic function to frame binary output model. The output of the logistic regression will be a probability ($0 \leq x \leq 1$)

Logistics Regression & Confusion Matrix



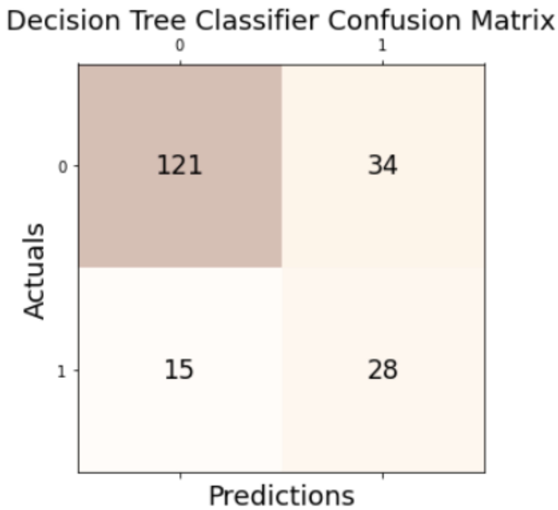

```
Accuracy Score = 0.8737373737373737
Precision Score = 0.9166666666666666
Recall Score = 0.9225806451612903
F1 Score = 0.9196141479099678
```

Random Forest Classifier & Confusion Matrix




```
Accuracy Score = 0.8737373737373737
Precision Score = 0.9166666666666666
Recall Score = 0.9225806451612903
F1 Score = 0.9196141479099678
```

Decision Tree Classifier & Confusion Matrix

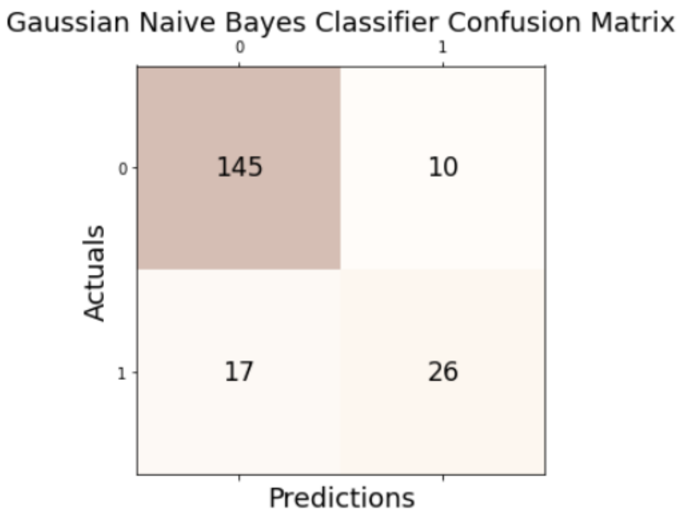



```
Accuracy Score = 0.8737373737373737
Precision Score = 0.9166666666666666
Recall Score = 0.9225806451612903
F1 Score = 0.9196141479099678
```

Decision Tree

Decision tree is a tree based algorithm used to solve regression and classification problems. An inverted tree is framed which is branched off from a homogeneous probability distributed root node, to highly heterogeneous leaf nodes, for deriving the output. Regression trees are used for dependent variable with continuous values and classification trees are used for dependent variable with discrete values.

Gaussian Naïve Bayes & Confusion Matrix

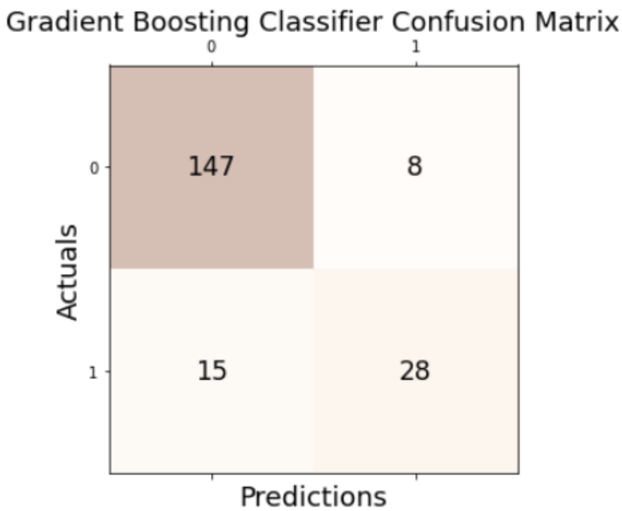



```
Accuracy Score = 0.8737373737373737
Precision Score = 0.9166666666666666
Recall Score = 0.9225806451612903
F1 Score = 0.9196141479099678
```

Gaussian Naive Bayes

Gaussian Naive Bayes is a generative model. (Gaussian) Naive Bayes assumes that each class follow a Gaussian distribution. Naive Bayes assumes independence of the features, which means the covariance matrices are diagonal matrices.

Gradient Boosting Classifier & Confusion Matrix




```
Accuracy Score = 0.8737373737373737
Precision Score = 0.9166666666666666
Recall Score = 0.9225806451612903
F1 Score = 0.9196141479099678
```

<u>Gradient Boosting Classifier</u>

In Gradient Boosting, each predictor tries to improve on its predecessor by reducing the errors. In Gradient Boosting, instead of fitting a predictor on the data at each iteration, it actually fits a new predictor to the residual errors made by the previous predictor.

Classifier Model Comparison & Analysis

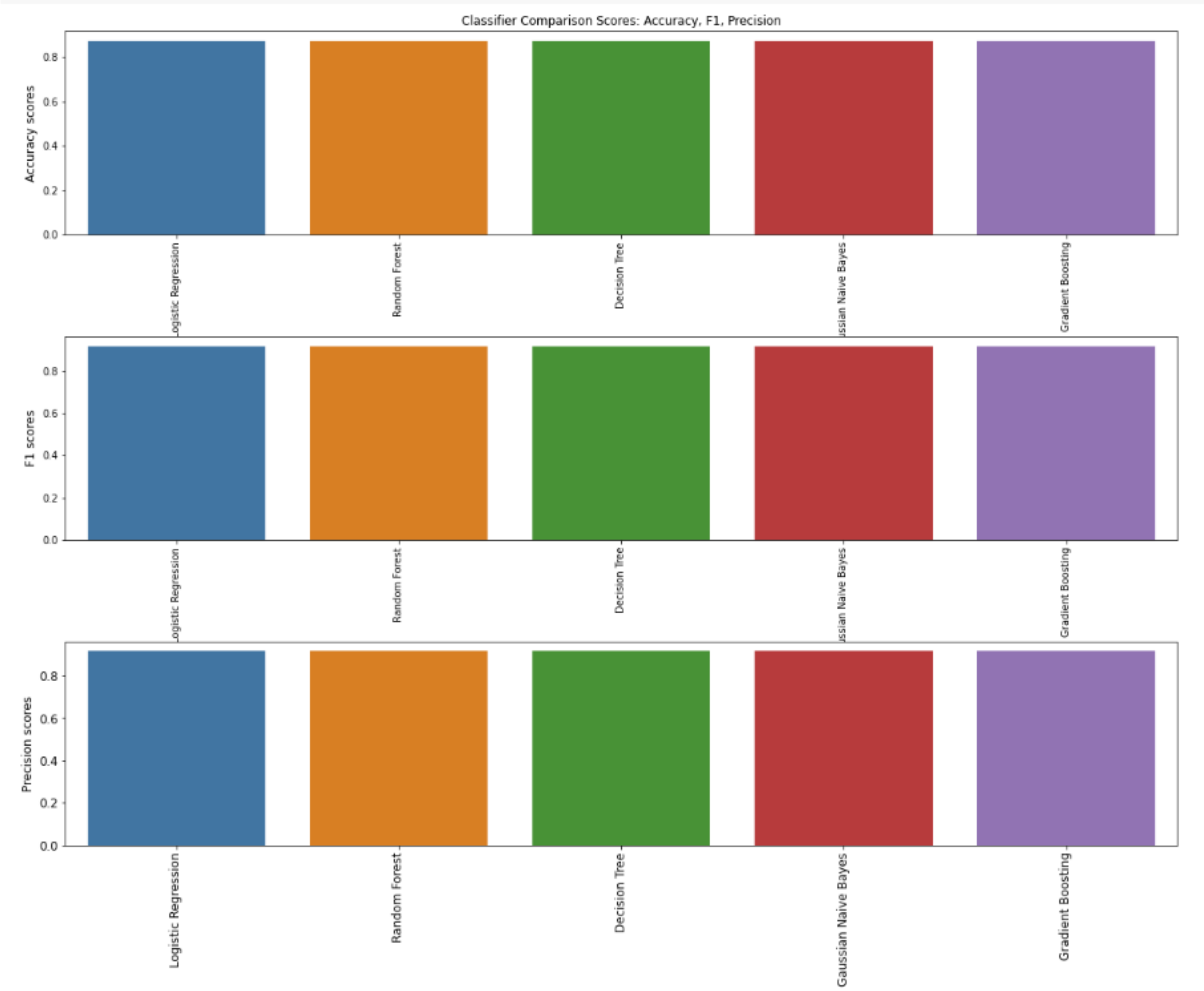
Classifier Model Comparison and Analysis

```
# Classifier Model Comparison and Analysis
classifier_names =['Logistic Regression','Random Forest','Decision Tree','Gaussian Naive Bayes','Gradient Boosting']
accuracy_scores = [accuracylr, accuracyrfc, accuracydtc, accuracygnb, accuracygbc]
precision_scores = [precisionlr, precisionrfc, precisiondtc, precisiongnb, precisiongbc]
recall_scores = [recalllr, recallrfc, recalldtc, recallgnb, recallgbc]
f1score_scores = [f1scorelr, f1scorerfc, f1scoredtc, f1scoregnb, f1scoregbc]
print(classifier_names)
print(accuracy_scores)
print(precision_scores)
print(recall_scores)
print(f1score_scores)
```

```
['Logistic Regression', 'Random Forest', 'Decision Tree', 'Gaussian Naive Bayes', 'Gradient Boosting']
[0.8737373737373737, 0.8737373737373737, 0.8737373737373737, 0.8737373737373737, 0.8737373737373737]
[0.9166666666666666, 0.9166666666666666, 0.9166666666666666, 0.9166666666666666, 0.9166666666666666]
[0.9225806451612903, 0.9225806451612903, 0.9225806451612903, 0.9225806451612903, 0.9225806451612903]
[0.9196141479099678, 0.9196141479099678, 0.9196141479099678, 0.9196141479099678, 0.9196141479099678]
```

\+ Code + Text

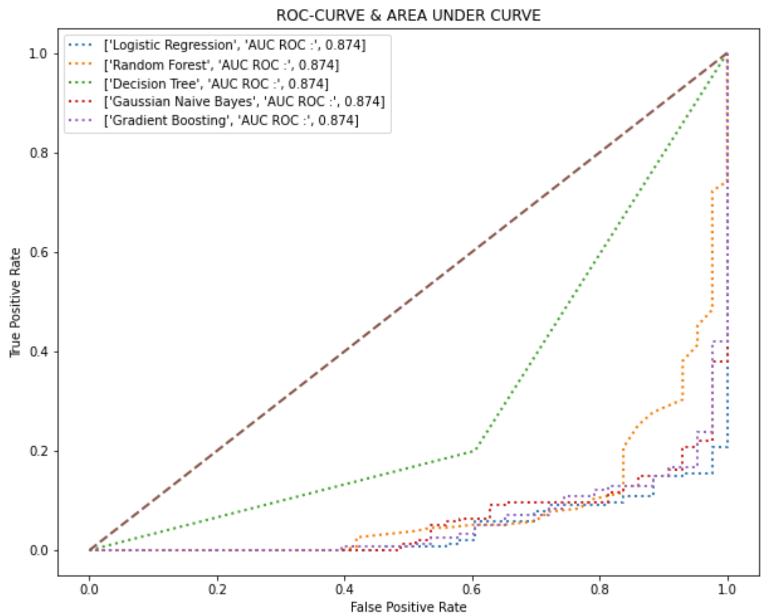
```
[63] sns.set_color_codes("colorblind")
```


Classifier Comparison Scores: Accuracy, F1, Precision
Accuracy scores
F1 scores
Precision scores
0.0
0.2
0.4
0.6
0.8
Logistic Regression
Random Forest
Decision Tree
Gaussian Naive Bayes
Gradient Boosting

ROC curve plot analysis

```python
fig = plt.figure(figsize=(10,8))
ax  = fig.add_subplot(111)

ax.plot(fpr_lr,tpr_lr,label = [classifier_names[0], "AUC ROC :", round(accuracylr,3)],linewidth=2,linestyle="dotted")
ax.plot(fpr_rfc,tpr_rfc,label = [classifier_names[1], "AUC ROC :", round(accuracyrfc,3)],linewidth=2,linestyle="dotted")
ax.plot(fpr_dtc,tpr_dtc,label = [classifier_names[2], "AUC ROC :", round(accuracydtc,3)],linewidth=2,linestyle="dotted")
ax.plot(fpr_gnb,tpr_gnb,label = [classifier_names[3], "AUC ROC :", round(accuracygnb,3)],linewidth=2,linestyle="dotted")
ax.plot(fpr_gbc,tpr_gbc,label = [classifier_names[4], "AUC ROC :", round(accuracygbc,3)],linewidth=2,linestyle="dotted")

ax.plot([0,1],[0,1],linewidth=2,linestyle="dashed")
plt.legend(loc="best")
plt.title("ROC-CURVE & AREA UNDER CURVE")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
Text(0, 0.5, 'True Positive Rate')
```



Decision of ML Model & ML model being used to predict supplier risk default in new dataset

Given the high scores of accuracy in the confusion matrix and overall high area under the curve of receiver operating characteristic curve, logistics regression model is being used for predicting the supplier default. Also the nature of the similar shape of data and given that logistics regression loss function will always be convex and this model is simple, fast and cbe used for multiclass classifications also.. I proceeded with Logistics Regression in this model.

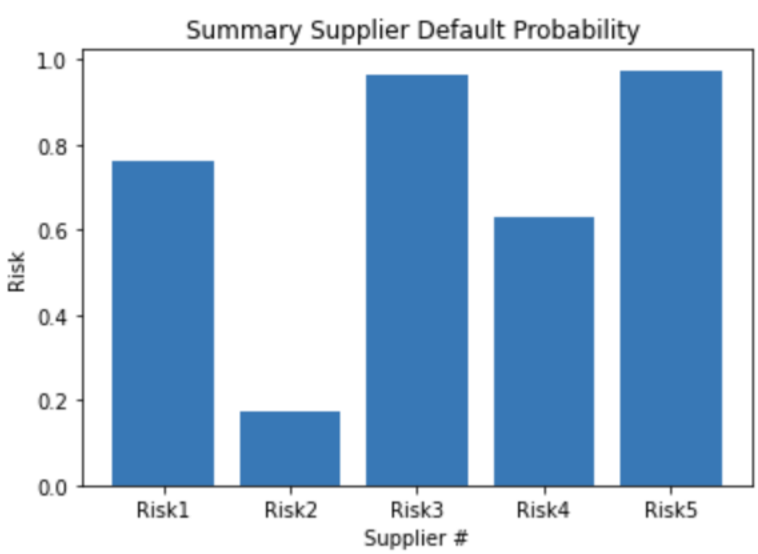
```
sgdtf3_supplierrisk_prob_1 = classifier.predict_proba(sgdtf3)
sgdtf3_supplierrisk_prob_1
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:413: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
array([[0.24002002, 0.75997998],
       [0.82555216, 0.17444784],
       [0.03640958, 0.96359042],
       [0.37072116, 0.62927884],
       [0.02615246, 0.97384754]])
```

```
[79] sgdtf3_supplierdefault_prob = sgdtf3_supplierrisk_prob_1[:, 1]
     sgdtf3_supplierdefault_prob
```

```
array([0.75997998, 0.17444784, 0.96359042, 0.62927884, 0.97384754])
```

```
import matplotlib.pyplot as plt

x_axis = ['Risk1', 'Risk2', 'Risk3', 'Risk4', 'Risk5']
y_axis = [sgdtf3_supplierdefault_prob[0], sgdtf3_supplierdefault_prob[1], sgdtf3_supplierdefault_prob[2], sgdtf3_supplierdefault_prob[3], sgdtf3_supplierdefault_prob[4] ]

plt.bar(x_axis, y_axis)
plt.title('Summary Supplier Default Probability ')
plt.xlabel('Supplier #')
plt.ylabel('Risk')
plt.show()
```



Foundation Objective function - Linear Programming

```
Minimum Cost for satisfying demand =  1933110.72

y[1]  =  0.0
y[2]  =  1.0
y[3]  =  0.0
y[4]  =  1.0
y[5]  =  1.0

X[1][1]  =  0.0 |  X[1][2]  =  0.0 |  X[1][3]  =  0.0 |  X[1][4]  =  118.4 |  X[1][5]  =  473.6 |
X[2][1]  =  0.0 |  X[2][2]  =  0.0 |  X[2][3]  =  0.0 |  X[2][4]  =  356.8 |  X[2][5]  =  89.2 |
X[3][1]  =  0.0 |  X[3][2]  =  438.4 |  X[3][3]  =  0.0 |  X[3][4]  =  0.0 |  X[3][5]  =  109.6 |
X[4][1]  =  0.0 |  X[4][2]  =  129.4 |  X[4][3]  =  0.0 |  X[4][4]  =  517.6 |  X[4][5]  =  0.0 |
X[5][1]  =  0.0 |  X[5][2]  =  49.0 |  X[5][3]  =  0.0 |  X[5][4]  =  0.0 |  X[5][5]  =  196.0 |
X[6][1]  =  0.0 |  X[6][2]  =  159.4 |  X[6][3]  =  0.0 |  X[6][4]  =  637.6 |  X[6][5]  =  0.0 |
X[7][1]  =  0.0 |  X[7][2]  =  0.0 |  X[7][3]  =  0.0 |  X[7][4]  =  482.4 |  X[7][5]  =  120.6 |
X[8][1]  =  0.0 |  X[8][2]  =  80.2 |  X[8][3]  =  0.0 |  X[8][4]  =  0.0 |  X[8][5]  =  320.8 |
```

Sensitivity Analysis: Objective function with supplier risk + Cost=80000

```
Minimum Cost for satisfying demand =  164298.13462803524

y[1]  =  -0.0
y[2]  =  1.0
y[3]  =  0.0
y[4]  =  1.0
y[5]  =  0.0

X[1][1]  =  0.0 |  X[1][2]  =  355.2 |  X[1][3]  =  0.0 |  X[1][4]  =  236.8 |  X[1][5]  =  0.0 |
X[2][1]  =  0.0 |  X[2][2]  =  267.6 |  X[2][3]  =  0.0 |  X[2][4]  =  178.4 |  X[2][5]  =  0.0 |
X[3][1]  =  0.0 |  X[3][2]  =  328.8 |  X[3][3]  =  0.0 |  X[3][4]  =  219.2 |  X[3][5]  =  0.0 |
X[4][1]  =  0.0 |  X[4][2]  =  388.2 |  X[4][3]  =  0.0 |  X[4][4]  =  258.8 |  X[4][5]  =  0.0 |
X[5][1]  =  0.0 |  X[5][2]  =  147.0 |  X[5][3]  =  0.0 |  X[5][4]  =  98.0 |  X[5][5]  =  0.0 |
X[6][1]  =  0.0 |  X[6][2]  =  478.2 |  X[6][3]  =  0.0 |  X[6][4]  =  318.8 |  X[6][5]  =  0.0 |
X[7][1]  =  0.0 |  X[7][2]  =  361.8 |  X[7][3]  =  0.0 |  X[7][4]  =  241.2 |  X[7][5]  =  0.0 |
X[8][1]  =  0.0 |  X[8][2]  =  240.6 |  X[8][3]  =  0.0 |  X[8][4]  =  160.4 |  X[8][5]  =  0.0 |
```

Sensitivity Analysis: Objective function with supplier risk + Cost=60000

```
Minimum Cost for satisfying demand =  148223.6009710265

y[1]  =  0.0
y[2]  =  1.0
y[3]  =  0.0
y[4]  =  1.0
y[5]  =  0.0

X[1][1]  =  0.0 |  X[1][2]  =  236.55 |  X[1][3]  =  0.0 |  X[1][4]  =  355.45 |  X[1][5]  =  0.0 |
X[2][1]  =  0.0 |  X[2][2]  =  178.21 |  X[2][3]  =  0.0 |  X[2][4]  =  267.79 |  X[2][5]  =  0.0 |
X[3][1]  =  0.0 |  X[3][2]  =  218.97 |  X[3][3]  =  0.0 |  X[3][4]  =  329.03 |  X[3][5]  =  0.0 |
X[4][1]  =  0.0 |  X[4][2]  =  258.53 |  X[4][3]  =  0.0 |  X[4][4]  =  388.47 |  X[4][5]  =  0.0 |
X[5][1]  =  0.0 |  X[5][2]  =  97.9 |  X[5][3]  =  0.0 |  X[5][4]  =  147.1 |  X[5][5]  =  0.0 |
X[6][1]  =  0.0 |  X[6][2]  =  318.47 |  X[6][3]  =  0.0 |  X[6][4]  =  478.53 |  X[6][5]  =  0.0 |
X[7][1]  =  0.0 |  X[7][2]  =  240.95 |  X[7][3]  =  0.0 |  X[7][4]  =  362.05 |  X[7][5]  =  0.0 |
X[8][1]  =  0.0 |  X[8][2]  =  160.23 |  X[8][3]  =  0.0 |  X[8][4]  =  240.77 |  X[8][5]  =  0.0 |
```

Sensitivity Analysis: Objective function with supplier risk + Cost=90000

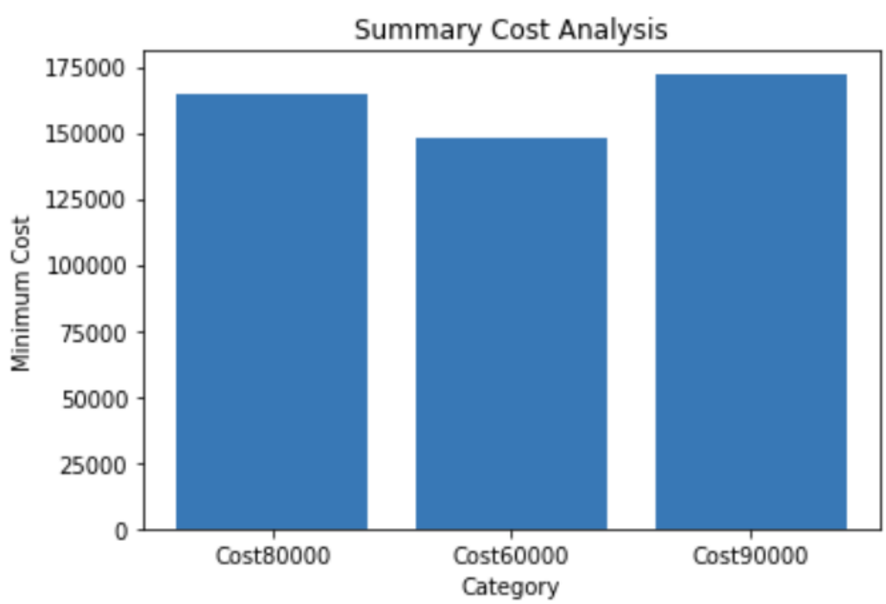
```
Minimum Cost for satisfying demand =  172335.40145653972

y[1]  =  0.0
y[2]  =  1.0
y[3]  =  0.0
y[4]  =  1.0
y[5]  =  0.0

X[1][1]  =  0.0 |  X[1][2]  =  355.2 |  X[1][3]  =  0.0 |  X[1][4]  =  236.8 |  X[1][5]  =  0.0 |
X[2][1]  =  0.0 |  X[2][2]  =  267.6 |  X[2][3]  =  0.0 |  X[2][4]  =  178.4 |  X[2][5]  =  0.0 |
X[3][1]  =  0.0 |  X[3][2]  =  328.8 |  X[3][3]  =  0.0 |  X[3][4]  =  219.2 |  X[3][5]  =  0.0 |
X[4][1]  =  0.0 |  X[4][2]  =  388.2 |  X[4][3]  =  0.0 |  X[4][4]  =  258.8 |  X[4][5]  =  0.0 |
X[5][1]  =  0.0 |  X[5][2]  =  147.0 |  X[5][3]  =  0.0 |  X[5][4]  =  98.0 |  X[5][5]  =  0.0 |
X[6][1]  =  0.0 |  X[6][2]  =  478.2 |  X[6][3]  =  0.0 |  X[6][4]  =  318.8 |  X[6][5]  =  0.0 |
X[7][1]  =  0.0 |  X[7][2]  =  361.8 |  X[7][3]  =  0.0 |  X[7][4]  =  241.2 |  X[7][5]  =  0.0 |
X[8][1]  =  0.0 |  X[8][2]  =  240.6 |  X[8][3]  =  0.0 |  X[8][4]  =  160.4 |  X[8][5]  =  0.0 |
```

Summary Cost Sensitivity Analysis

Summary Cost Sensitivity Analysis

```
[103] Min_cost_summary = [MinCost_WithoutSupplierRiskProbability, MinCost_WithSupplierRisk_CostSensitivity80000, MinCost_WithSupplierRisk_CostSensitivity60000, MinCost_WithSupplierRisk_CostSensitiv
      print (Min_cost_summary)
      print ('MinCost_WithoutSupplierRiskProbability = ', Min_cost_summary[0],
             '\nMinCost_WithSupplierRisk_CostSensitivity80000 = ', Min_cost_summary[1],
             '\nMinCost_WithSupplierRisk_CostSensitivity60000 = ', Min_cost_summary[2],
             '\nMinCost_WithSupplierRisk_CostSensitivity90000 = ', Min_cost_summary[3]
            )
```

```
[1933110.72, 164298.13462803524, 148223.6009710265, 172335.40145653972]
MinCost_WithoutSupplierRiskProbability =  1933110.72
MinCost_WithSupplierRisk_CostSensitivity80000 =  164298.13462803524
MinCost_WithSupplierRisk_CostSensitivity60000 =  148223.6009710265
MinCost_WithSupplierRisk_CostSensitivity90000 =  172335.40145653972
```



```
MinCost_WithoutSupplierRiskProbability =  1933110.72
MinCost_WithSupplierRisk_CostSensitivity80000 =  164298.13462803524
MinCost_WithSupplierRisk_CostSensitivity60000 =  148223.6009710265
MinCost_WithSupplierRisk_CostSensitivity90000 =  172335.40145653972
```

Conclusion

Minimum Cost Category	Minimum Cost Value
Base cost function (without supplier risk probability)	1933110
With Supplier default risk probability with Cost Sensitivity with $80,000	164298
With Supplier default risk probability with Cost Sensitivity with $60,000	148223
With Supplier default risk probability with Cost Sensitivity with $90,000	172335

Although logistics regression was done in the above model, we could have also used the ML model with Gradient Boosting Classifier as that had the best area under the curve (although lower in some of the other metrics of accuracy).