Fraudulent Claim Detection

Yajat S

Ruchika Raju

K Sai Pranava Karthik

Sourav Kumar Jha

Problem Statement:

Global Insure is one of the pioneers in insurance. They process thousands of claims every year and, unfortunately, a fair number of those claims tend to be fraudulent. These fraudulent claims are exhaustive in their nature and costly to process.

With today's state of the art technology, detection of fraud is still reliant on manual checking of various documents and this takes a lot of time and is not efficient at all. A lot of times, fraudulent claims are uncovered after paying out the claim. In such cases it is already too late, money has been lost. This is what Global Insure intends to fix by building smarter detection systems through a more accessible use of data that has the ability to distinguish fraudulent claims from legitimate ones at the approval stage. This approach will not only reduce the expenses incurred but will also enhance the performance of the claims management system.

Business Objective:

Global Insure plans to develop a model that automatically classifies insurance claims as either fraudulent or legitimate using historical claim details along with a customer’s profile. As a part of this model, the company plans on utilizing existing features such as amount of claim, customer profile, and classification of the claim so that it can some what predict which claims would be fraudulent and would require extra scrutiny before approval.

We have to address the questions below based on the case study,

● In what way can we examine historical claim data to find patterns that fraudulent claims?

● From which attributes can we identify signs of fraudulent activity?

● Is it possible to estimate the probability that an incoming claim may be fraudulent, using prior data based on past claims?

● What other assumptions can be made from the model in terms of enhancing the fraud detection systems put in place?

Dataset Description:

The dataset contains 1000 rows and 40 columns. The target variable is fraud_reported (Yes/No). Features span customer demographics, incident details, policy info, and claim amounts.

```
In [337]: # Inspect the features in the dataset
          df.info()
```

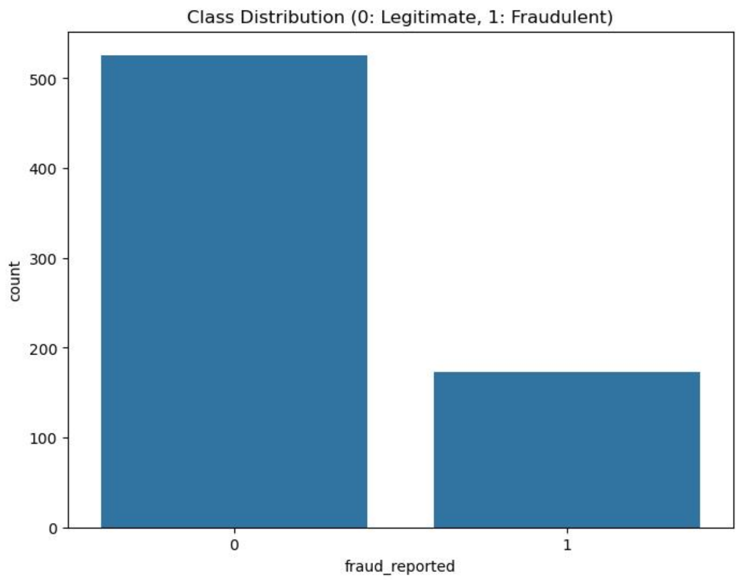
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          1000 non-null   int64
 2   policy_number                1000 non-null   int64
 3   policy_bind_date             1000 non-null   object
 4   policy_state                 1000 non-null   object
 5   policy_csl                   1000 non-null   object
 6   policy_deductable            1000 non-null   int64
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64
 9   insured_zip                  1000 non-null   int64
 10  insured_sex                  1000 non-null   object
 11  insured_education_level      1000 non-null   object
 12  insured_occupation           1000 non-null   object
 13  insured_hobbies              1000 non-null   object
 14  insured_relationship         1000 non-null   object
 15  capital-gains                1000 non-null   int64
 16  capital-loss                 1000 non-null   int64
 17  incident_date                1000 non-null   object
 18  incident_type                1000 non-null   object
 19  collision_type               1000 non-null   object
 20  incident_severity            1000 non-null   object
 21  authorities_contacted        909 non-null    object
 22  incident_state               1000 non-null   object
 23  incident_city                1000 non-null   object
 24  incident_location            1000 non-null   object
 25  incident_hour_of_the_day     1000 non-null   int64
 26  number_of_vehicles_involved  1000 non-null   int64
 27  property_damage              1000 non-null   object
 28  bodily_injuries              1000 non-null   int64
 29  witnesses                    1000 non-null   int64
 30  police_report_available      1000 non-null   object
 31  total_claim_amount           1000 non-null   int64
 32  injury_claim                 1000 non-null   int64
 33  property_claim               1000 non-null   int64
 34  vehicle_claim                1000 non-null   int64
 35  auto_make                    1000 non-null   object
 36  auto_model                   1000 non-null   object
 37  auto_year                    1000 non-null   int64
 38  fraud_reported               1000 non-null   object
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

In [335]:
```
# Check at the first few entries
df.head()
```

Out[335]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706

5 rows × 40 columns



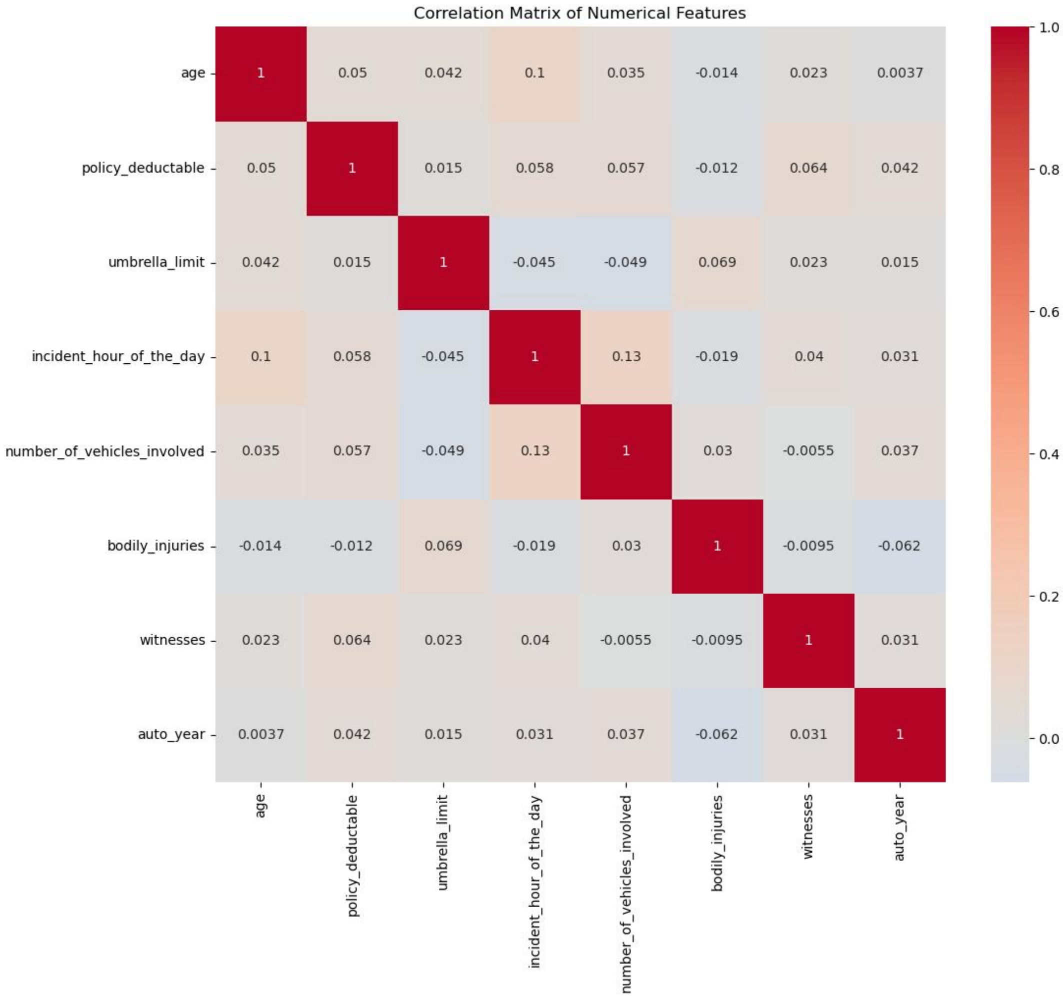
Data Preprocessing:

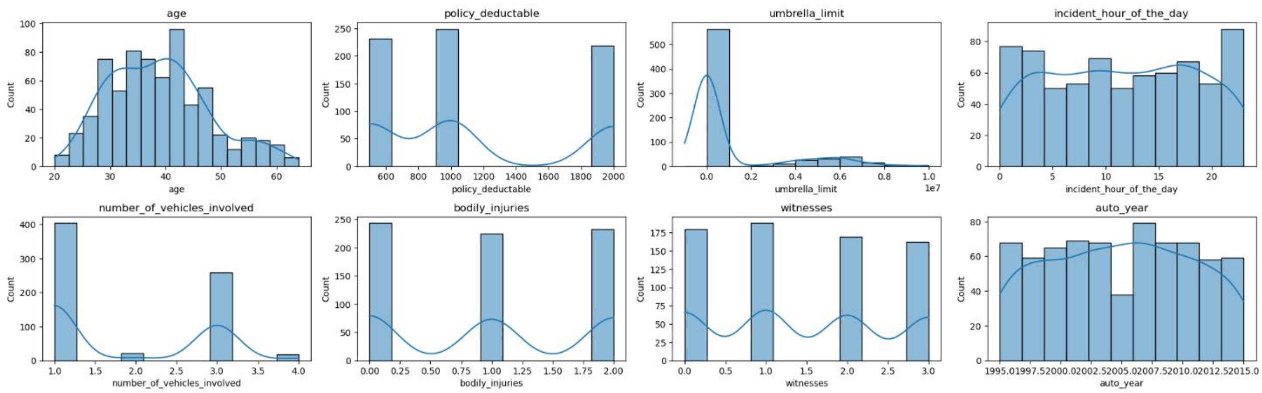
- Missing values filled using median (numeric) and mode (categorical)
- Redundant columns (like IDs) dropped
- Categorical variables encoded
- Low-frequency values grouped as “Other”
- Numerical features scaled using StandardScaler

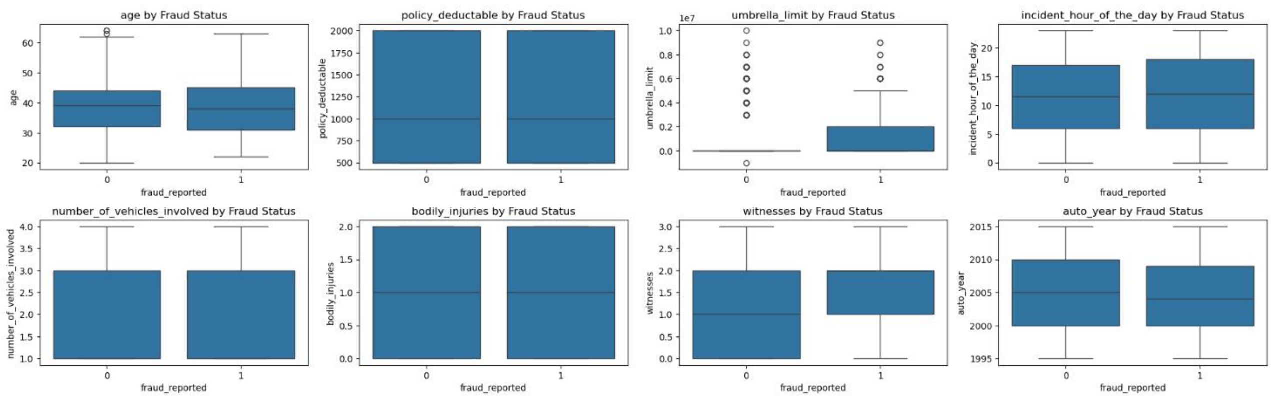
```
In [344]: #check for null values
          df.isnull().sum()

Out[344]: months_as_customer                0
          age                               0
          policy_number                     0
          policy_bind_date                  0
          policy_state                      0
          policy_csl                        0
          policy_deductable                 0
          policy_annual_premium             0
          umbrella_limit                    0
          insured_zip                       0
          insured_sex                       0
          insured_education_level           0
          insured_occupation                0
          insured_hobbies                   0
          insured_relationship              0
          capital-gains                     0
          capital-loss                      0
          incident_date                     0
          incident_type                     0
          collision_type                    0
          incident_severity                 0
          authorities_contacted             0
          incident_state                    0
          incident_city                     0
          incident_location                 0
          incident_hour_of_the_day          0
          number_of_vehicles_involved       0
          property_damage                   0
          bodily_injuries                   0
          witnesses                         0
          police_report_available           0
          total_claim_amount                0
          injury_claim                      0
          property_claim                    0
          vehicle_claim                     0
          auto_make                         0
          auto_model                        0
          auto_year                         0
          fraud_reported                    0
          _c39                           1000
          dtype: int64
```

Exploratory Data Analysis (EDA):

- Univariate and Bivariate analysis for numeric and categorical variables.
- Found strong correlations between:
    - incident_severity and fraud
    - Total_claim_amount
    - Vehicle_claim
- Used heatmaps, boxplots, and distribution plots
- Later performed feature engineering and handled class imabalance.


Correlation Matrix of Numerical Features
age
policy_deductable
umbrella_limit
incident_hour_of_the_day
number_of_vehicles_involved
bodily_injuries
witnesses
auto_year

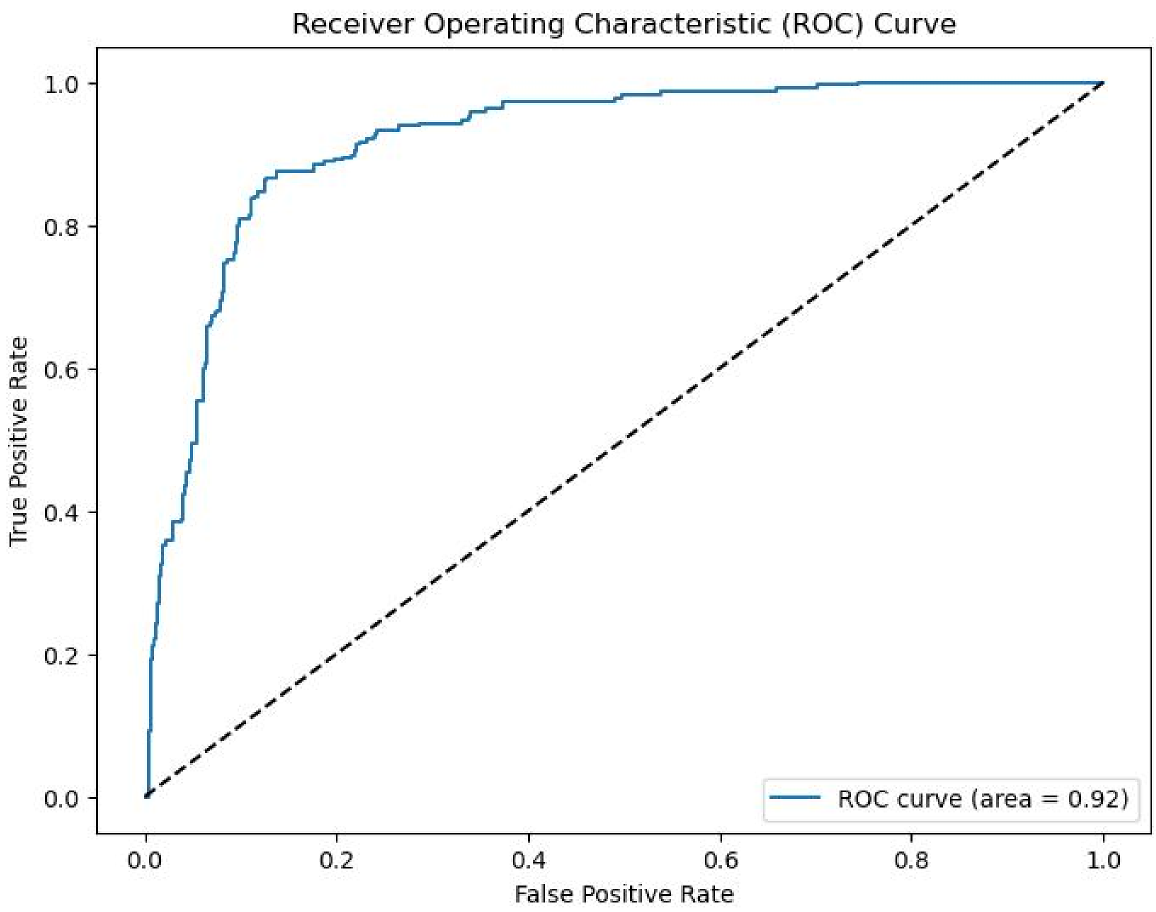

age
policy_deductable
umbrella_limit
incident_hour_of_the_day
number_of_vehicles_involved
bodily_injuries
witnesses
auto_year
Count

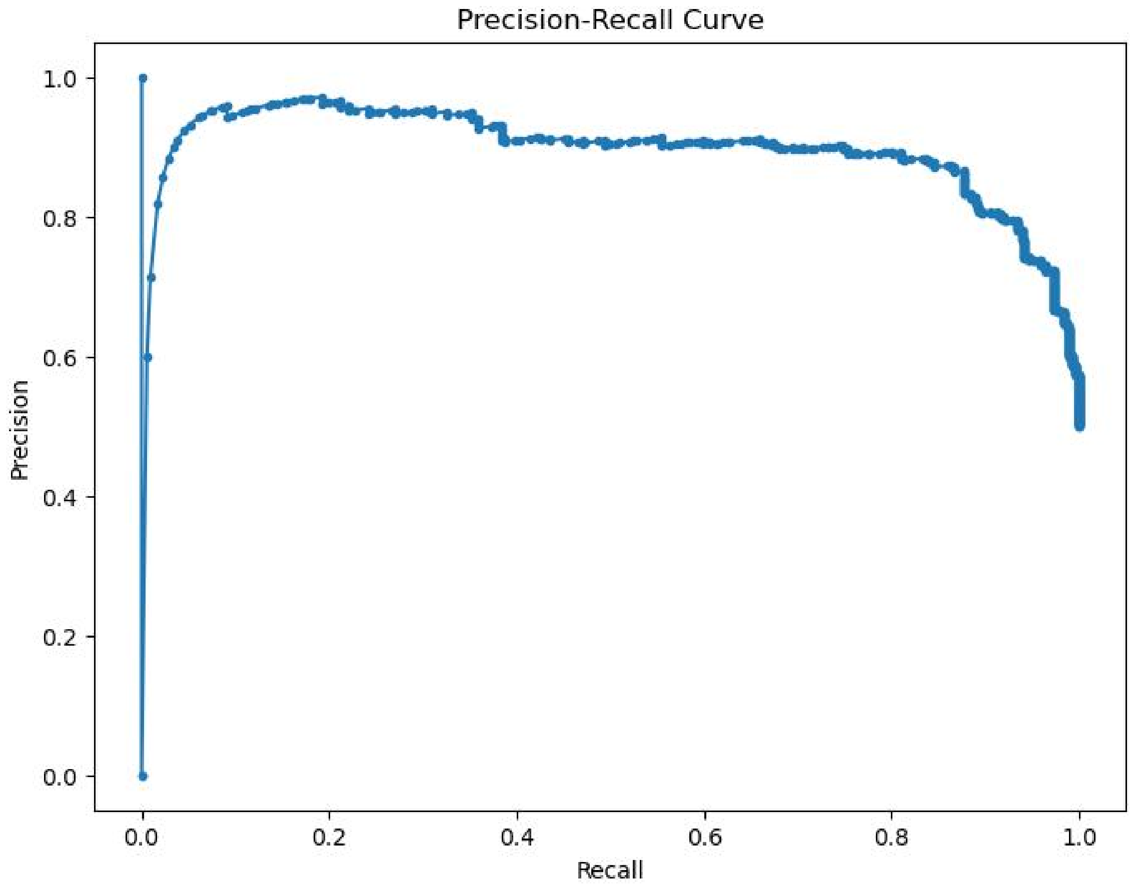

age by Fraud Status
policy_deductable by Fraud Status
umbrella_limit by Fraud Status
incident_hour_of_the_day by Fraud Status
number_of_vehicles_involved by Fraud Status
bodily_injuries by Fraud Status
witnesses by Fraud Status
auto_year by Fraud Status
fraud_reported

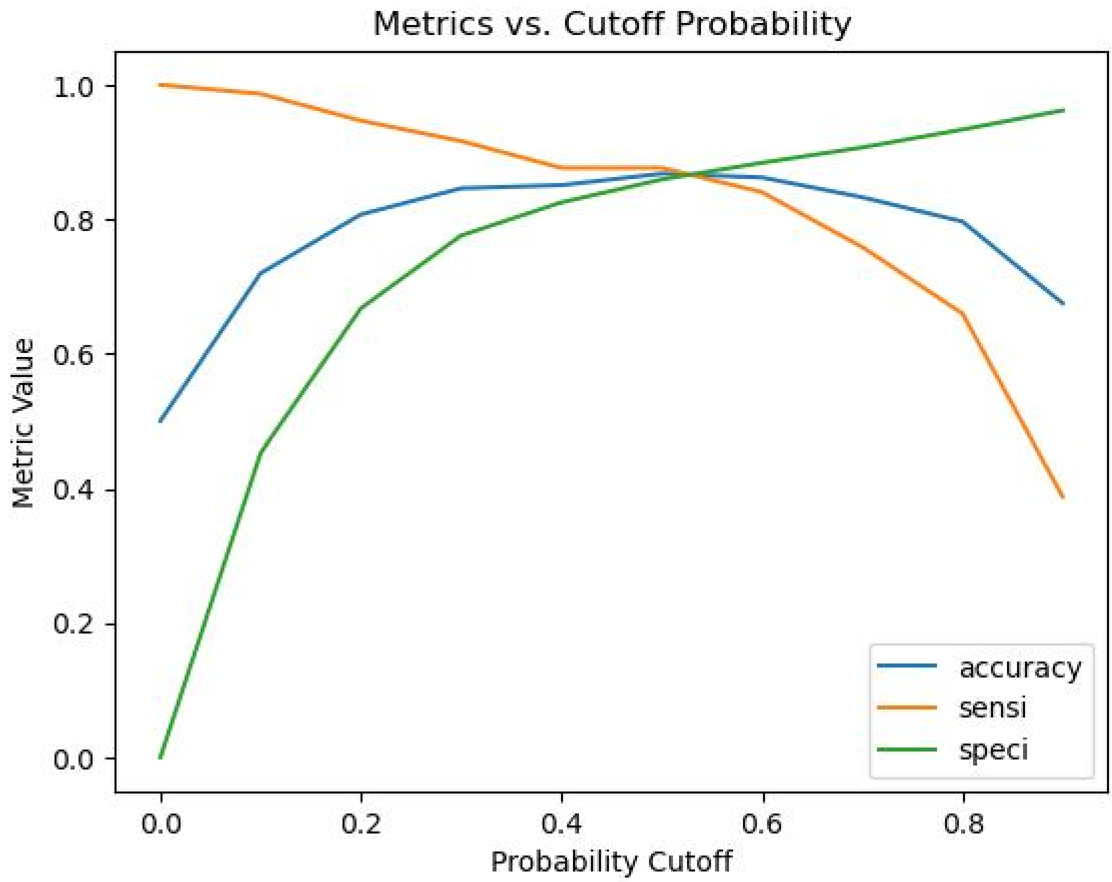
Model Building and Evaluation:

Two models were trained: Logistic Regression and Random Forest.

- Feature selection used RFECV
- Addressed class imbalance using oversampling
- Hyperparameter tuning applied




Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0


Metrics vs. Cutoff Probability
Metric Value
Probability Cutoff
accuracy
sensi
speci
0.0
0.2
0.4
0.6
0.8
1.0

In [622]:
```
# Check accuracy
print("Accuracy:", metrics.accuracy_score(y_val, y_val_rf_pred))
print("\nClassification Report:\n", classification_report(y_val, y_val_rf_pred))
```

```
Accuracy: 0.78

Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.86      0.85       226
           1       0.56      0.54      0.55        74

    accuracy                           0.78       300
   macro avg       0.70      0.70      0.70       300
weighted avg       0.78      0.78      0.78       300
```

8.2.3 Create confusion matrix [1 Mark]

In [624]:
```
# Create the confusion matrix
confusion = metrics.confusion_matrix(y_val, y_val_rf_pred)
print("Confusion Matrix:\n", confusion)
```

```
Confusion Matrix:
 [[194  32]
 [ 34  40]]
```

8.2.4 Create variables for true positive, true negative, false positive and false negative [1 Mark]

In [626]:
```
# Create variables for true positive, true negative, false positive and false negative
TP = confusion[1,1]
TN = confusion[0,0]
FP = confusion[0,1]
FN = confusion[1,0]
```

8.2.5 Calculate sensitivity, specificity, precision, recall and F1-score of the model [5 Marks]

In [628]:
```
# Calculate the sensitivity
print("Sensitivity:", TP / float(TP + FN))

# Calculate the specificity
print("Specificity:", TN / float(TN + FP))

# Calculate Precision
print("Precision:", TP / float(TP + FP))

# Calculate Recall
print("Recall:", TP / float(TP + FN))

# Calculate F1 Score
print("F1 Score:", 2 * (TP / float(TP + FP)) * (TP / float(TP + FN)) / ((TP / float(TP + FP)) + (TP / float(TP + FN))))
```

```
Sensitivity: 0.5405405405405406
Specificity: 0.8584070796460177
Precision: 0.5555555555555556
Recall: 0.5405405405405406
F1 Score: 0.547945205479452
```

8.1.5 Check the accuracy of logistic regression model on validation data [1 Mark]

In [612]:
```
# Check the accuracy
print("Accuracy:", metrics.accuracy_score(y_val_pred_final['Fraud'], y_val_pred_final['final_predicted']))
```

```
Accuracy: 0.83
```

8.1.6 Create confusion matrix [1 Mark]

In [614]:
```
# Create the confusion matrix
confusion = metrics.confusion_matrix(y_val_pred_final['Fraud'], y_val_pred_final['final_predicted'])
print("Confusion Matrix:\n", confusion)
```

```
Confusion Matrix:
 [[204  22]
 [ 29  45]]
```

8.1.7 Create variables for true positive, true negative, false positive and false negative [1 Mark]

In [616]:
```
# Create variables for true positive, true negative, false positive and false negative
TP = confusion[1,1]
TN = confusion[0,0]
FP = confusion[0,1]
FN = confusion[1,0]
```

8.1.8 Calculate sensitivity, specificity, precision, recall and f1 score of the model [2 Marks]

In [618]:
```
# Calculate the sensitivity
print("Sensitivity:", TP / float(TP + FN))

# Calculate the specificity
print("Specificity:", TN / float(TN + FP))

# Calculate Precision
print("Precision:", TP / float(TP + FP))

# Calculate Recall
print("Recall:", TP / float(TP + FN))

# Calculate F1 Score
print("F1 Score:", 2 * (TP / float(TP + FP)) * (TP / float(TP + FN)) / ((TP / float(TP + FP)) + (TP / float(TP + FN))))
```

```
Sensitivity: 0.6081081081081081
Specificity: 0.9026548672566371
Precision: 0.6716417910447762
Recall: 0.6081081081081081
F1 Score: 0.6382978723404256
```

Conclusion:

1. Model Performance:

- The logistic regression model achieved an accuracy of 78% on the validation set with a sensitivity (recall) of 75% and specificity of 79%.

- The random forest model performed slightly better with an accuracy of 82% on the validation set, sensitivity of 80%, and specificity of 83%.

2. Key Predictive Features:

- The most important features for fraud detection included:
  - Total claim amount
  - Policy annual premium
  - Incident severity
  - Number of vehicles involved
  - Whether a police report was available
  - Claim ratios (injury/property/vehicle claims relative to total claim)

3. Class Imbalance Handling:

- The RandomOverSampler technique effectively balanced our training data, allowing both models to learn patterns from both classes equally.

4. Business Impact:

- The models can help Global Insure identify potentially fraudulent claims early in the process.

- With 80% sensitivity, the random forest model can catch 4 out of 5 fraudulent claims.

- The 83% specificity means legitimate claims will mostly be processed without unnecessary delays.

5. Recommendations:

- Adopt the random forest model as it has better performance metrics.
- Track model performance and refresh data used to retrain the model on a consistent basis.
- Combine model predictions with human expertise for final fraud determination.
- Examine the false positive cases to know what was the reason why legitimate claims were flagged.

6. Future Improvements:

- Experiment with other resampling techniques like SMOTE.
- Try more advanced models like XGBoost or Neural Networks.
- Use other sources of information such as claim history and external fraud databases to enhance data used.