In [ ]:
```
#### Logistic Regression
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
from feature_engine.outliers import Winsorizer
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler #, MinMaxScaler

from sklearn.pipeline import Pipeline
import pickle, joblib
```

In [4]:
```
# import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn.model_selection import train_test_split # train and test
```

In [5]:
```
# import pylab as pl
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import classification_report
```

In [39]:
```
### import data
data = pd.read_csv(r"C:\Users\shubham lokare\Downloads\Logistic Regresssion (1)\Logistic Regresssion\claimants.
```

In [40]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   CASENUM   1340 non-null   int64  
 1   ATTORNEY  1340 non-null   int64  
 2   CLMSEX    1328 non-null   float64
 3   CLMINSUR  1299 non-null   float64
 4   SEATBELT  1292 non-null   float64
 5   CLMAGE    1151 non-null   float64
 6   LOSS      1340 non-null   float64
dtypes: float64(5), int64(2)
memory usage: 73.4 KB
```

In [41]:
```
data.head(10)
```

Out[41]:

	CASENUM	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	5	0	0.0	1.0	0.0	50.0	34.940
1	3	1	1.0	0.0	0.0	18.0	0.891
2	66	1	0.0	1.0	0.0	5.0	0.330
3	70	0	0.0	1.0	1.0	31.0	0.037
4	96	1	0.0	1.0	0.0	30.0	0.038
5	97	0	1.0	1.0	0.0	35.0	0.309
6	10	0	0.0	1.0	0.0	9.0	3.538
7	36	0	1.0	1.0	0.0	34.0	4.881
8	51	1	1.0	1.0	0.0	60.0	0.874
9	55	1	0.0	1.0	0.0	NaN	0.350

In [42]:
```
data.describe()
```

Out[42]:

	CASENUM	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
count	1340.000000	1340.000000	1328.000000	1299.000000	1292.000000	1151.000000	1340.000000
mean	11202.001493	0.488806	0.558735	0.907621	0.017028	28.414422	3.806307
std	9512.750796	0.500061	0.496725	0.289671	0.129425	20.304451	10.636903
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4177.000000	0.000000	0.000000	1.000000	0.000000	9.000000	0.400000
50%	8756.500000	0.000000	1.000000	1.000000	0.000000	30.000000	1.069500
75%	15702.500000	1.000000	1.000000	1.000000	0.000000	43.000000	3.781500
max	34153.000000	1.000000	1.000000	1.000000	1.000000	95.000000	173.604000

In [43]:
```
### Remove unwanted columns
```

In [43]:
```
### Remove unwanted columns
data1 = data.drop(['CASENUM'] , axis =1)
```

In [44]:
```
data1
```

Out[44]:

	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	0	0.0	1.0	0.0	50.0	34.940
1	1	1.0	0.0	0.0	18.0	0.891
2	1	0.0	1.0	0.0	5.0	0.330
3	0	0.0	1.0	1.0	31.0	0.037
4	1	0.0	1.0	0.0	30.0	0.038
...	...	...	...	...	...	...
1335	1	0.0	1.0	0.0	NaN	0.576
1336	0	1.0	1.0	0.0	46.0	3.705
1337	1	1.0	1.0	0.0	39.0	0.099
1338	0	1.0	0.0	0.0	8.0	3.177
1339	1	1.0	1.0	0.0	30.0	0.688

1340 rows × 6 columns

In [51]:
```
### Split the data into input and output variable

X = pd.DataFrame(data1.iloc[: ,1:6])
Y = pd.DataFrame(data1.iloc[:,0])
```

In [48]:
```
### input Variable
X
```

Out[48]:

	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	0.0	1.0	0.0	50.0	34.940
1	1.0	0.0	0.0	18.0	0.891
2	0.0	1.0	0.0	5.0	0.330
3	0.0	1.0	1.0	31.0	0.037
4	0.0	1.0	0.0	30.0	0.038
...	...	...	...	...	...
1335	0.0	1.0	0.0	NaN	0.576
1336	1.0	1.0	0.0	46.0	3.705
1337	1.0	1.0	0.0	39.0	0.099
1338	1.0	0.0	0.0	8.0	3.177
1339	1.0	1.0	0.0	30.0	0.688

1340 rows × 5 columns

In [52]:
```
### Target
Y
```

Out[52]:

	ATTORNEY
0	0
1	1
2	1
3	0
4	1
...	...
1335	1
1336	0
1337	1
1338	0
1339	1

1340 rows × 1 columns

In [54]:
```
### check missing values
X.isna().sum()
```

Out[54]:
```
CLMSEX       12
CLMINSUR     41
SEATBELT     48
CLMAGE      189
LOSS          0
dtype: int64
```

In [55]:
```
# Segregating data based on types
numeric_features = X.select_dtypes(exclude = ['object']).columns
numeric_features
```

Out[55]:
```
Index(['CLMSEX', 'CLMINSUR', 'SEATBELT', 'CLMAGE', 'LOSS'], dtype='object')
```

In [56]:
```
### We use impute method to remove missing values
num_pipeline = Pipeline(steps=[('impute', SimpleImputer(strategy = 'mean'))])
```

In [57]:
```
# 1st Imputation Transformer
process = ColumnTransformer([('mode', num_pipeline, numeric_features)])
```

In [58]:
```
print(process)
```

```
ColumnTransformer(transformers=[('mode',
                                 Pipeline(steps=[('impute', SimpleImputer())]),
                                 Index(['CLMSEX', 'CLMINSUR', 'SEATBELT', 'CLMAGE', 'LOSS'], dtype='object'))])
```

In [60]:
```
process1 = process.fit(X)
```

In [61]:
```
## addd into DataFrame

new_data = pd.DataFrame(process1.transform(X) , columns = X.columns)
```

In [63]:
```
new_data
```

Out[63]:

	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	0.0	1.0	0.0	50.000000	34.940
1	1.0	0.0	0.0	18.000000	0.891
2	0.0	1.0	0.0	5.000000	0.330
3	0.0	1.0	1.0	31.000000	0.037
4	0.0	1.0	0.0	30.000000	0.038
...	...	...	...	...	...
1335	0.0	1.0	0.0	28.414422	0.576
1336	1.0	1.0	0.0	46.000000	3.705
1337	1.0	1.0	0.0	39.000000	0.099
1338	1.0	0.0	0.0	8.000000	3.177
1339	1.0	1.0	0.0	30.000000	0.688

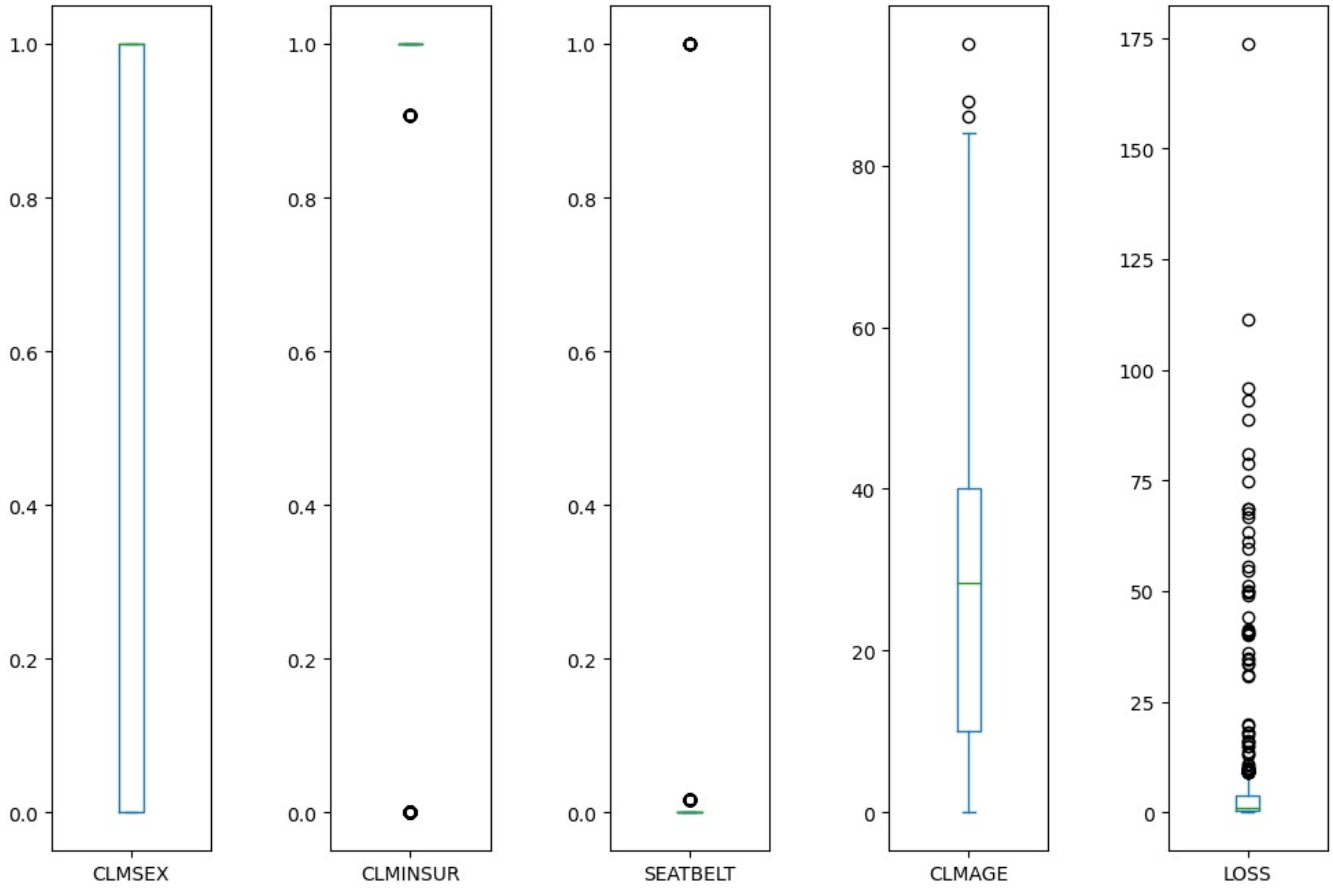
1340 rows × 5 columns

In [64]:
```
### check missing Value
new_data.isna().sum()
```

Out[64]:
```
CLMSEX      0
CLMINSUR    0
SEATBELT    0
CLMAGE      0
LOSS        0
dtype: int64
```

In [65]:
```
#### check outliers
### check outliers we use boxplot

new_data.plot(kind='box' ,subplots = True , figsize =(12 ,8))
plt.subplots_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the aver
plt.show()
```



In [66]:
```
### To remove outliers we use winsorazation method
winsor = Winsorizer(capping_method = 'iqr', # choose  IQR rule boundaries or gaussian for mean and std
                            tail = 'both', # cap left, right or both tails
                            fold = 1.5,
                            variables = ['CLMAGE', 'LOSS'])
```

In [67]:
```
outlier_pipeline = Pipeline(steps = [('winsor', winsor)])
outlier_pipeline
```

Out[67]:

▸ **Pipeline**

▸ Winsorizer

In [71]:
```
process2 =ColumnTransformer(transformers = [('wins',
                                             outlier_pipeline,
                                             numeric_features)],
                                  remainder = 'passthrough')
```

In [72]:
```
print(process2)
```

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('wins',
                                 Pipeline(steps=[('winsor',
                                                  Winsorizer(capping_method='iqr',
                                                             fold=1.5,
                                                             tail='both',
                                                             variables=['CLMAGE',
                                                                        'LOSS']))]),
                                 Index(['CLMSEX', 'CLMINSUR', 'SEATBELT', 'CLMAGE', 'LOSS'], dtype='object'))])
```

In [74]:
```
wins_data = process2.fit(new_data)
```

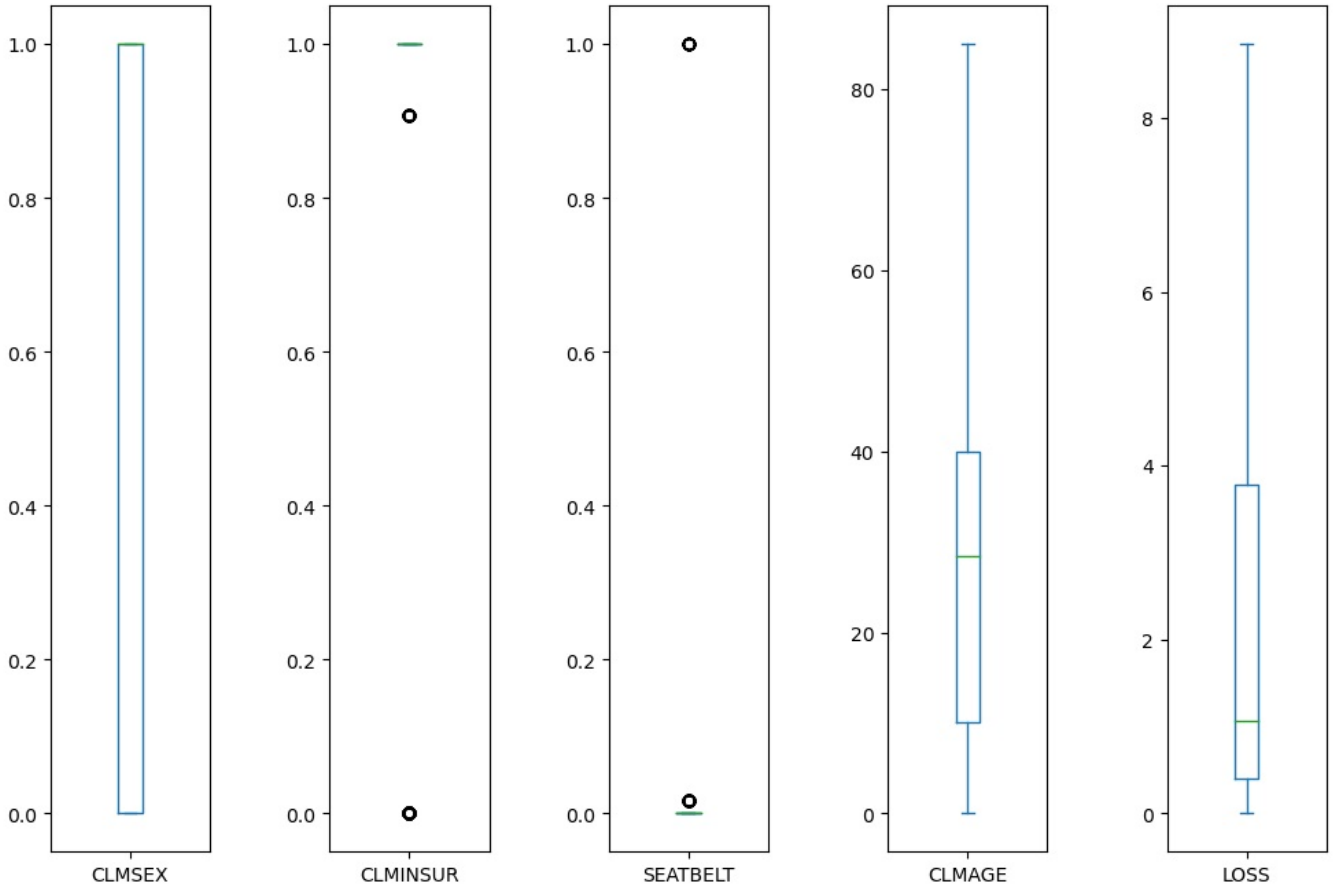
In [75]:
```
### Add indo Dataframe
new_data1 = pd.DataFrame(wins_data.transform(new_data) , columns = new_data.columns)
new_data1
```

Out[75]:

	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	0.0	1.0	0.0	50.000000	8.85375
1	1.0	0.0	0.0	18.000000	0.89100
2	0.0	1.0	0.0	5.000000	0.33000
3	0.0	1.0	1.0	31.000000	0.03700
4	0.0	1.0	0.0	30.000000	0.03800
...	...	...	...	...	...
1335	0.0	1.0	0.0	28.414422	0.57600
1336	1.0	1.0	0.0	46.000000	3.70500
1337	1.0	1.0	0.0	39.000000	0.09900
1338	1.0	0.0	0.0	8.000000	3.17700
1339	1.0	1.0	0.0	30.000000	0.68800

1340 rows × 5 columns

In [76]:
```
### check outliers
new_data1.plot(kind='box' ,subplots = True , figsize =(12 ,8))
plt.subplots_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the aver
plt.show()
```



In [77]:
```
#### then make data into scale
scale_pipeline = Pipeline(steps=[('scale', StandardScaler())])
# scale_pipeline = Pipeline(steps=[('scale', MinMaxScaler())])

preprocessor2 = ColumnTransformer(transformers = [('num',
                                                   scale_pipeline, numeric_features)],
                                  remainder = 'passthrough')
```

In [78]:
```
scale = scale_pipeline.fit(new_data1)
```

In [79]:
```
### add into dataFrame
clean_data = pd.DataFrame(scale.transform(new_data1) , columns = new_data1.columns)
clean_data
```

Out[79]:

	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	-1.130333	0.324027	-0.134039	1.150174	2.523487
1	0.892689	-3.183568	-0.134039	-0.554101	-0.575439
2	-1.130333	0.324027	-0.134039	-1.246463	-0.793767
3	-1.130333	0.324027	7.737696	0.138261	-0.907796
4	-1.130333	0.324027	-0.134039	0.085002	-0.907407
...	...	...	...	...	...
1335	-1.130333	0.324027	-0.134039	0.000556	-0.698030
1336	0.892689	0.324027	-0.134039	0.937140	0.519708
1337	0.892689	0.324027	-0.134039	0.564329	-0.883667
1338	0.892689	-3.183568	-0.134039	-1.086687	0.314222
1339	0.892689	0.324027	-0.134039	0.085002	-0.654442

1340 rows × 5 columns

In [80]:
```
### Statsmodel  of logistic regression
# Building the model and fitting the data

model = sm.Logit(Y,clean_data).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.591843
         Iterations 5
```

In [81]:
```
model.summary()
```

Out[81]:

Logit Regression Results

Dep. Variable:	ATTORNEY	**No. Observations:**	1340
Model:	Logit	**Df Residuals:**	1335
Method:	MLE	**Df Model:**	4
Date:	Sat, 01 Jun 2024	**Pseudo R-squ.:**	0.1458
Time:	15:30:44	**Log-Likelihood:**	-793.07
converged:	True	**LL-Null:**	-928.48
Covariance Type:	nonrobust	**LLR p-value:**	2.120e-57

	coef	std err	z	P>|z|	[0.025	0.975]
CLMSEX	0.1755	0.061	2.889	0.004	0.056	0.295
CLMINSUR	0.1553	0.060	2.569	0.010	0.037	0.274
SEATBELT	-0.0875	0.065	-1.340	0.180	-0.215	0.040
CLMAGE	0.1446	0.062	2.339	0.019	0.023	0.266
LOSS	-1.0363	0.075	-13.782	0.000	-1.184	-0.889

In [82]:
```
# Prediction
pred = model.predict(clean_data)
```

In [84]:
```
# Probabilities
pred
```

Out[84]:
```
0       0.070109
1       0.547431
2       0.623855
3       0.533826
4       0.693444
          ...
1335    0.642696
1336    0.454052
1337    0.771367
1338    0.308176
1339    0.712833
Length: 1340, dtype: float64
```

In [94]:
```
## # ROC Curve to identify the appropriate cutoff value to make output informat of 0 and 1
fpr, tpr, thresholds = roc_curve(Y.ATTORNEY, pred)
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
optimal_threshold
```

Out[94]:
```
0.5692935412111958
```

In [95]:
```
auc = metrics.auc(fpr,tpr)
print("Area under is : %f" % auc)
```

```
Area under is : 0.762515
```

In [96]:
```
# Filling all the cells with zeroes
clean_data['pred'] = np.zeros(1340)
```

In [97]:
```
### if pred value is grather then optimal_thershold value then it is Ture
clean_data.loc[pred > optimal_threshold ,"pred"] = 1
```

In [100…
```
# Confusion Matrix
confusion_matrix(clean_data['pred'] ,Y.ATTORNEY)
```

Out[100]:
```
array([[462, 160],
       [223, 495]], dtype=int64)
```

In [102…
```
# Accuracy score of the model
print("test Accuracy :" , accuracy_score(clean_data['pred'] ,Y.ATTORNEY))
```

```
test Accuracy : 0.7141791044776119
```

In [103…
```
### Classification report
classification = classification_report(clean_data["pred"], Y)
print(classification)
```

```
              precision    recall  f1-score   support

         0.0       0.67      0.74      0.71       622
         1.0       0.76      0.69      0.72       718

    accuracy                           0.71      1340
   macro avg       0.72      0.72      0.71      1340
weighted avg       0.72      0.71      0.71      1340
```

In [105…
```
### # Model evaluation - Data Split
x_train, x_test, y_train, y_test = train_test_split (clean_data.iloc[:, :5], Y,
                                                     test_size = 0.2,
                                                     random_state = 0)
```

In [106…
```
# Fitting Logistic Regression to the training set
logisticmodel = sm.Logit(y_train, x_train).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.584701
         Iterations 6
```

In [107…
```
# Evaluate on train data
y_pred_train = logisticmodel.predict(x_train)
y_pred_train
```

Out[107]:
```
362     0.468335
483     0.642628
866     0.757335
625     0.111296
194     0.736422
          ...
763     0.738082
835     0.757335
1216    0.726410
559     0.769961
684     0.162178
Length: 1072, dtype: float64
```

In [108…
```
# Metrics
# Filling all the cells with zeroes
y_train["pred"] = np.zeros(1072)
```

In [109…
```
# taking threshold value and above the prob value will be treated as correct value
y_train.loc[pred > optimal_threshold, "pred"] = 1
```

In [110…
```
auc = metrics.roc_auc_score(y_train["ATTORNEY"], y_pred_train)
print("Area under the ROC curve : %f" % auc)
```

```
Area under the ROC curve : 0.769350
```

In [111…
```
classification_train = classification_report(y_train["pred"], y_train["ATTORNEY"])
print(classification_train)
```

```
              precision    recall  f1-score   support

         0.0       0.69      0.75      0.72       500
         1.0       0.76      0.70      0.73       572

    accuracy                           0.72      1072
   macro avg       0.72      0.72      0.72      1072
weighted avg       0.73      0.72      0.72      1072
```

In [113…
```
# confusion matrix
```

```
confusion_matrix(y_train["pred"], y_train["ATTORNEY"])
```

Out[113]:
```
array([[373, 127],
       [170, 402]], dtype=int64)
```

In [114…
```
# Accuracy score of the model
print('Train accuracy = ', accuracy_score(y_train["pred"], y_train["ATTORNEY"]))
```

```
Train accuracy =  0.7229477611940298
```

In [115…
```
# Validate on Test data
y_pred_test = logisticmodel.predict(x_test)
y_pred_test
```

Out[115]:
```
574     0.220831
661     0.086551
458     0.079589
1023    0.396612
958     0.335217
          ...
1111    0.621442
1074    0.534153
744     0.311883
731     0.297463
317     0.745047
Length: 268, dtype: float64
```

In [116…
```
# Filling all the cells with zeroes
y_test["y_pred_test"] = np.zeros(268)

# Capturing the prediction binary values
y_test.loc[y_pred_test > optimal_threshold, "y_pred_test"] = 1
```

In [117…
```
# confusion matrix
confusion_matrix(y_test["y_pred_test"], y_test["ATTORNEY"])
```

Out[117]:
```
array([[88, 32],
       [54, 94]], dtype=int64)
```

In [118…
```
# Accuracy score of the model
print('Test accuracy = ', accuracy_score(y_test["y_pred_test"], y_test["ATTORNEY"]))
```

```
Test accuracy =  0.6791044776119403
```

In [119…
```
# classification report
classification1 = classification_report(y_test["y_pred_test"], y_test["ATTORNEY"])
print(classification1)
```

```
              precision    recall  f1-score   support

         0.0       0.62      0.73      0.67       120
         1.0       0.75      0.64      0.69       148

    accuracy                           0.68       268
   macro avg       0.68      0.68      0.68       268
weighted avg       0.69      0.68      0.68       268
```

In [120…
```
### final output
print(clean_data)
```

```
        CLMSEX  CLMINSUR  SEATBELT    CLMAGE      LOSS  pred
0    -1.130333  0.324027 -0.134039  1.150174  2.523487   0.0
1     0.892689 -3.183568 -0.134039 -0.554101 -0.575439   0.0
2    -1.130333  0.324027 -0.134039 -1.246463 -0.793767   1.0
3    -1.130333  0.324027  7.737696  0.138261 -0.907796   0.0
4    -1.130333  0.324027 -0.134039  0.085002 -0.907407   1.0
...        ...       ...       ...       ...       ...   ...
1335 -1.130333  0.324027 -0.134039  0.000556 -0.698030   1.0
1336  0.892689  0.324027 -0.134039  0.937140  0.519708   0.0
1337  0.892689  0.324027 -0.134039  0.564329 -0.883667   1.0
1338  0.892689 -3.183568 -0.134039 -1.086687  0.314222   0.0
1339  0.892689  0.324027 -0.134039  0.085002 -0.654442   1.0

[1340 rows x 6 columns]
```

In [ ]:
```
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js