Insurance Claims- Fraud Detection

Fraud occurs when someone knowingly lies to obtain a benefit or advantage to which they are not otherwise entitled or someone knowingly denies a benefit that is due and to which someone is entitled. According to the law, the crime of insurance fraud can be prosecuted when:

- The suspect had the intent to defraud. Insurance fraud is a "specific" intent crime. This means a prosecutor must prove that the person involved knowingly committed an act to defraud.
- An act is completed. Simply making a misrepresentation (written or oral) to an insurer with knowledge that is untrue is sufficient.
- The act and intent must come together. One without the other is not a crime.
- Actual monetary loss is not necessary as long as the suspect has committed an act and had the intent to commit the crime.

Types of Insurance Fraud:

- Automobile.
- Medical.
- Property.
- Healthcare.
- Workers' Compensation and others.

The insurance industry consists of more than 7,000 companies that collect over $1 trillion in premiums each year. The massive size of the industry contributes significantly to the cost of insurance fraud by providing more opportunities and bigger incentives for committing illegal activities.

The total cost of insurance fraud (non-health insurance) is estimated to be more than $40 billion per year. That means Insurance Fraud costs the average U.S. family between $400 and $700 per year in the form of increased premiums.

Problem Definition:

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem. Fraud in insurance is done by intentional deception or misrepresentation for gaining shabby benefit in the form of showing false expenditures and claim. Data mining tools and techniques can be used to detect fraud in large sets of insurance claim data.

Dataset:

```
insu = pd.read_csv('https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fraud.csv
insu.head()
```

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

5 rows × 40 columns

Data Analysis:

Information About the Dataset.

```
insu.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               822 non-null    object
 20  incident_severity            1000 non-null   object
 21  authorities_contacted        1000 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              640 non-null    object
 28  bodily_injuries              1000 non-null   int64
 29  witnesses                    1000 non-null   int64
 30  police_report_available      657 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
```

- RangeIndex: 0 to 999
- Total columns: 40
- dtypes: float64(2), int64(17), object(21)

Description of the Dataset:

```
insu.describe().T
```

	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
_c39	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Short description of our dataset

- Counts
- Mean
- Standard deviation
- Minimum
- 1st quantile
- 2nd quantile
- 3rd quantile
- Maximum value of each column.

Null Values:

```
insu.isna().sum()
```

```
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                178
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               360
bodily_injuries                 0
witnesses                       0
police_report_available       343
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
```

There are null values present in our dataset.

Visualization:

- Scatterplot between policy_annual_premium & auto_make

```
scatter(x = 'policy_annual_premium',y = 'auto_make', data = insu)
```

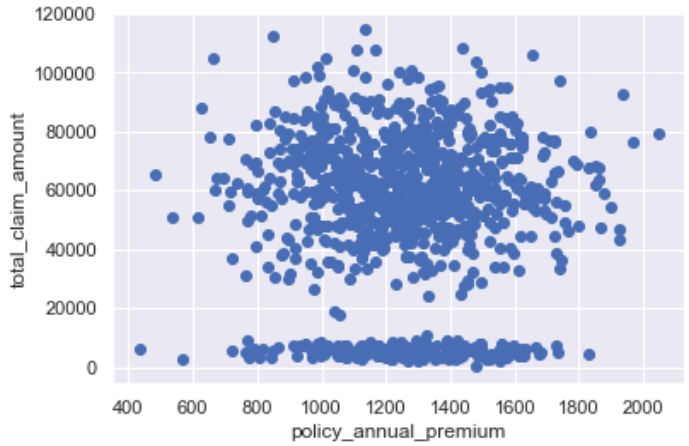
Accura, Dodge, Honda, Jeep, Mercedes is having the highest policy annual premium among all the auto makers.

Nissan, Toyota are the lowest in policy annual premium.

- Scatterplot between policy_annual_premium & total_claim_amount

```
scatter(x = 'policy_annual_premium',y = 'total_claim_amount', data = insu)
```

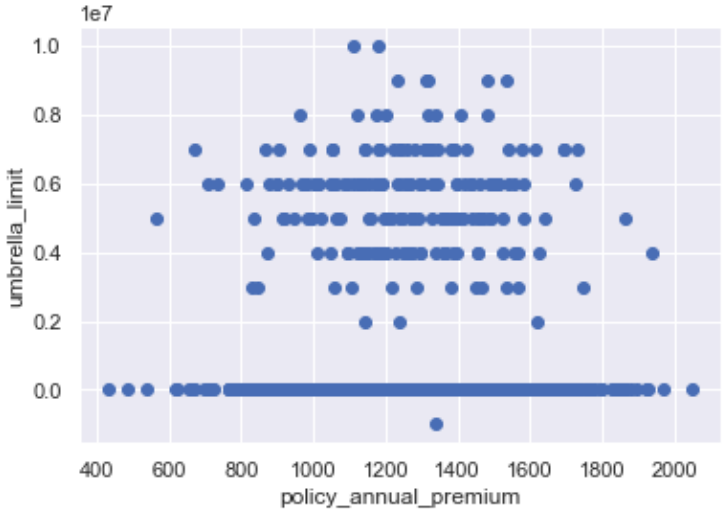
Scatterpot between policy_annual_premium and total_claim_amount



- Scatterplot between policy_annual_premium and umbrella_limit

```
scatter(x = 'policy_annual_premium',y = 'umbrella_limit', data = insu)
```

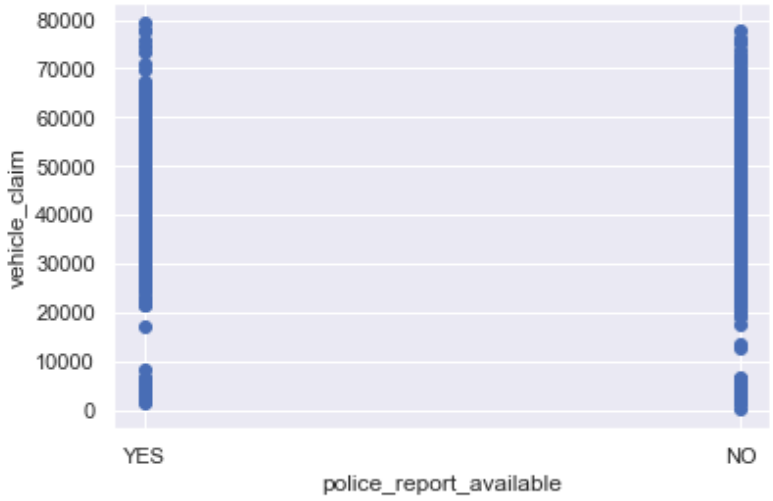
Scatterpot between policy_annual_premium and umbrella_limit



- Scatterpot between police_report_available and vehicle_claim

```
scatter(x = 'police_report_available',y = 'vehicle_claim', data = insu)
```

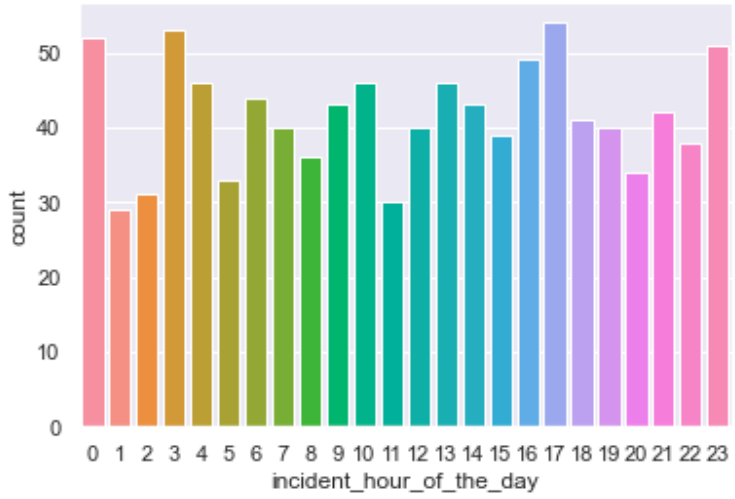
Scatterpot between police_report_available and vehicle_claim



- Countplot of incident_hour_of_the_day

```
sns.countplot(insu['incident_hour_of_the_day'])
```

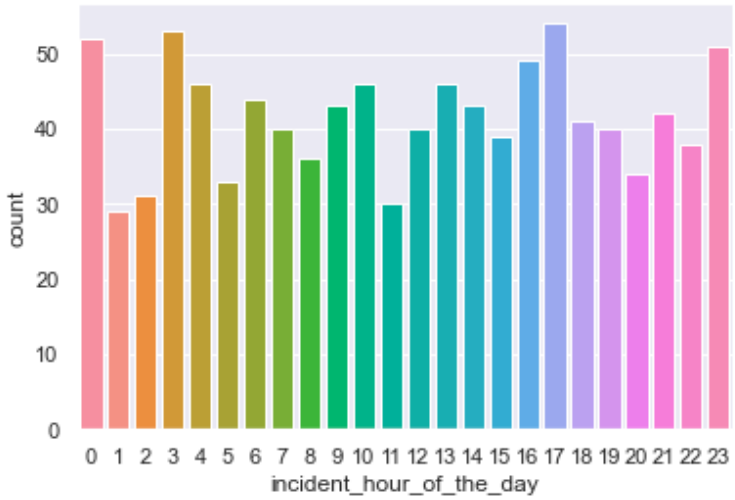
```
<AxesSubplot:xlabel='incident_hour_of_the_day', ylabel='count'>
```



- Countplot of number_of_vehicles_involved

```
sns.countplot(insu['incident_hour_of_the_day'])
```

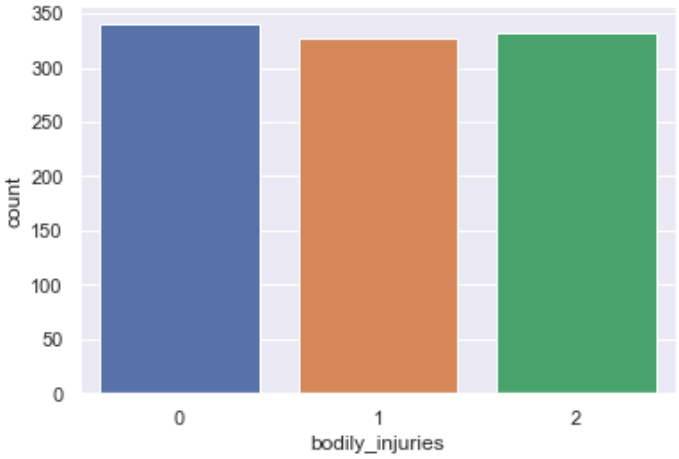
```
<AxesSubplot:xlabel='incident_hour_of_the_day', ylabel='count'>
```



- Countplot of bodily_injuries

```
sns.countplot(insu['bodily_injuries'])
```

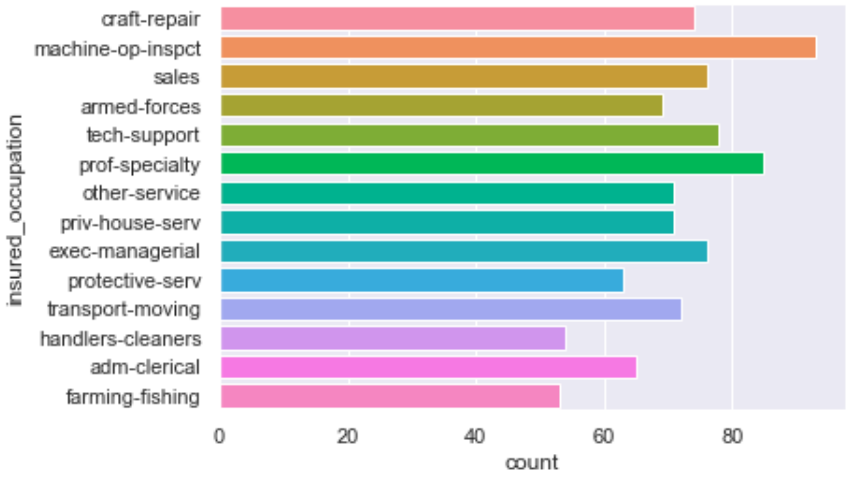
```
<AxesSubplot:xlabel='bodily_injuries', ylabel='count'>
```



- Countplot of insured_occupation

```
sns.countplot(y = insu['insured_occupation'])
```

```
<AxesSubplot:xlabel='count', ylabel='insured_occupation'>
```



EDA Concluding Remarks:

Data Cleaning

- Filling Null Values & Dropping Column with Complete Null Values.

```
insu['collision_type'] = insu['collision_type'].fillna(insu['collision_type'].mode()[0])
insu['property_damage'] = insu['property_damage'].fillna(insu['property_damage'].mode()[0])
insu['police_report_available'] = insu['police_report_available'].fillna(insu['police_report_available'].mode()[0])
insu.drop('_c39',axis = 1,inplace = True)
```

```
insu.isna().sum()
```

```
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
dtype: int64
```

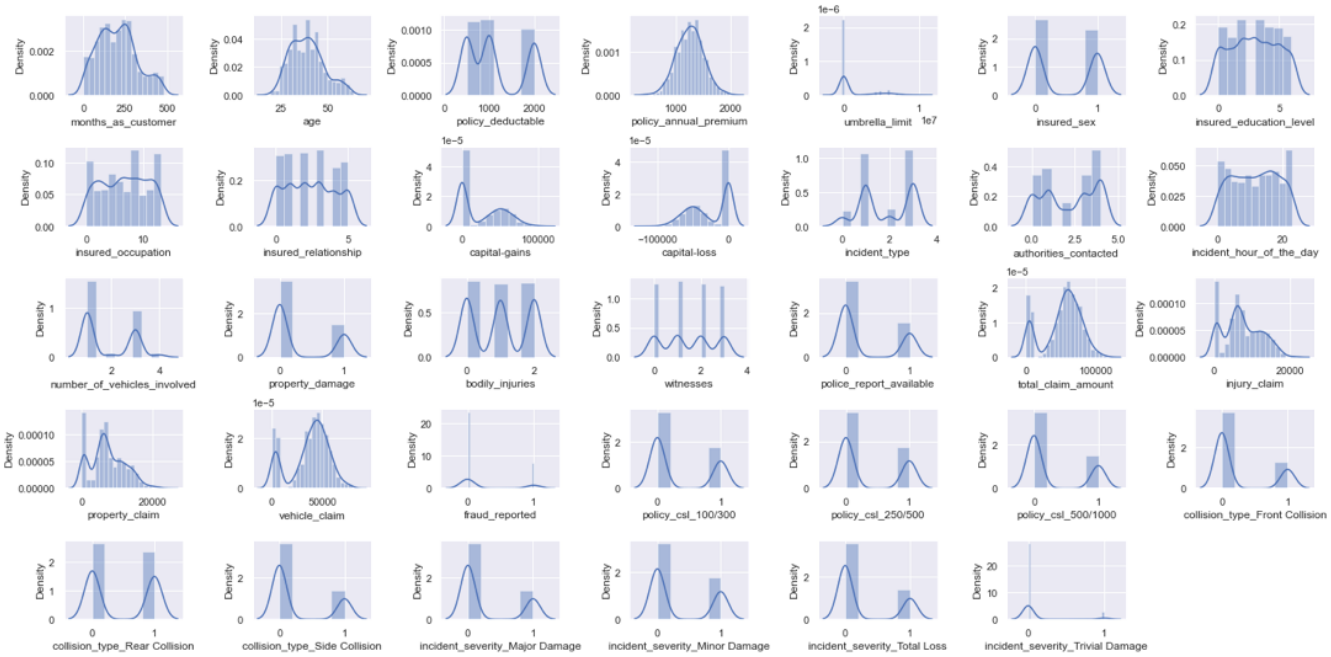
- Dropping Unwanted Columns.

```
insu.drop(columns = ['policy_number','policy_bind_date','policy_state','insured_zip','incident_location','incident_date','incident_state',
```

```
insu.shape
```

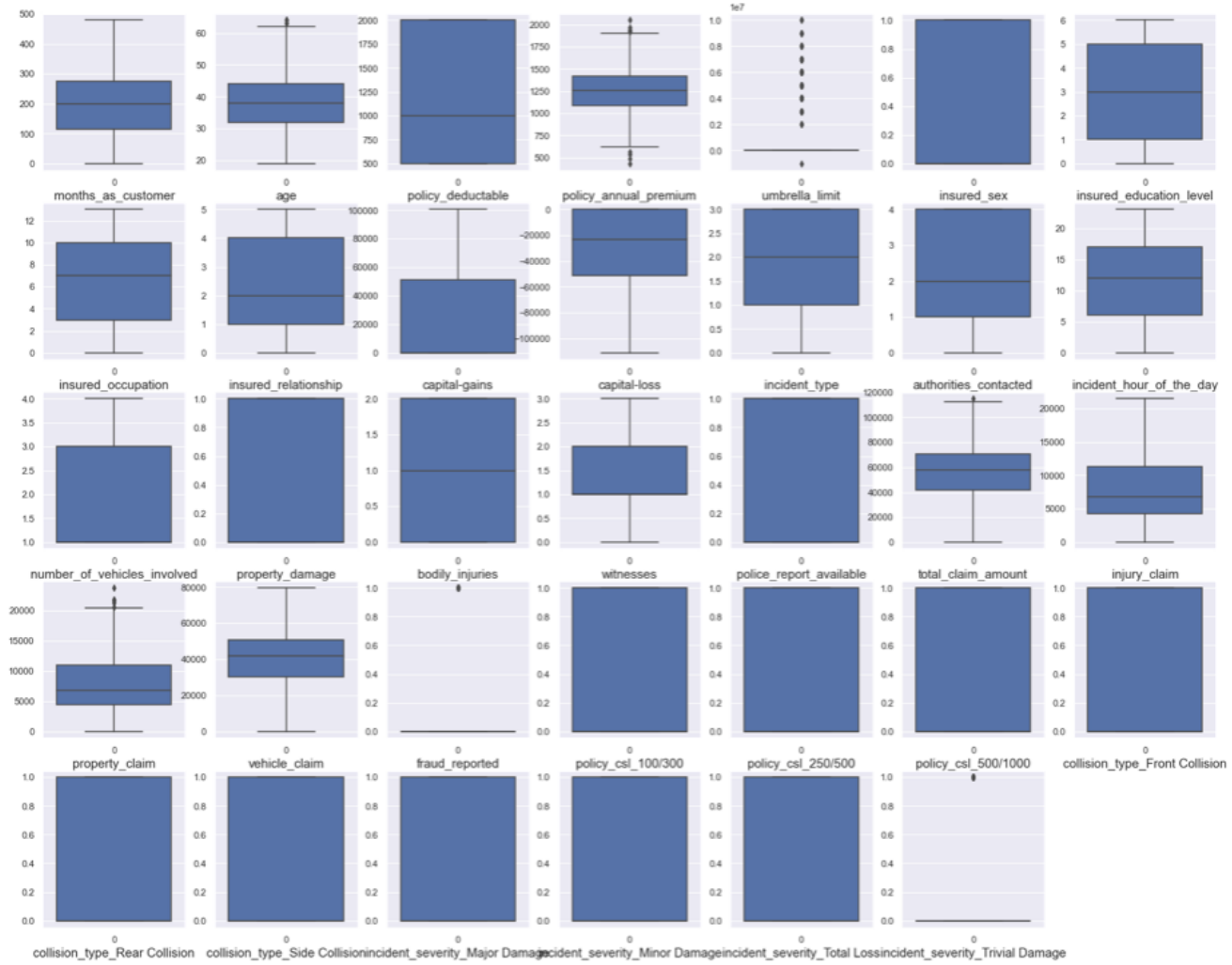
```
(1000, 27)
```

- Normal Distribution



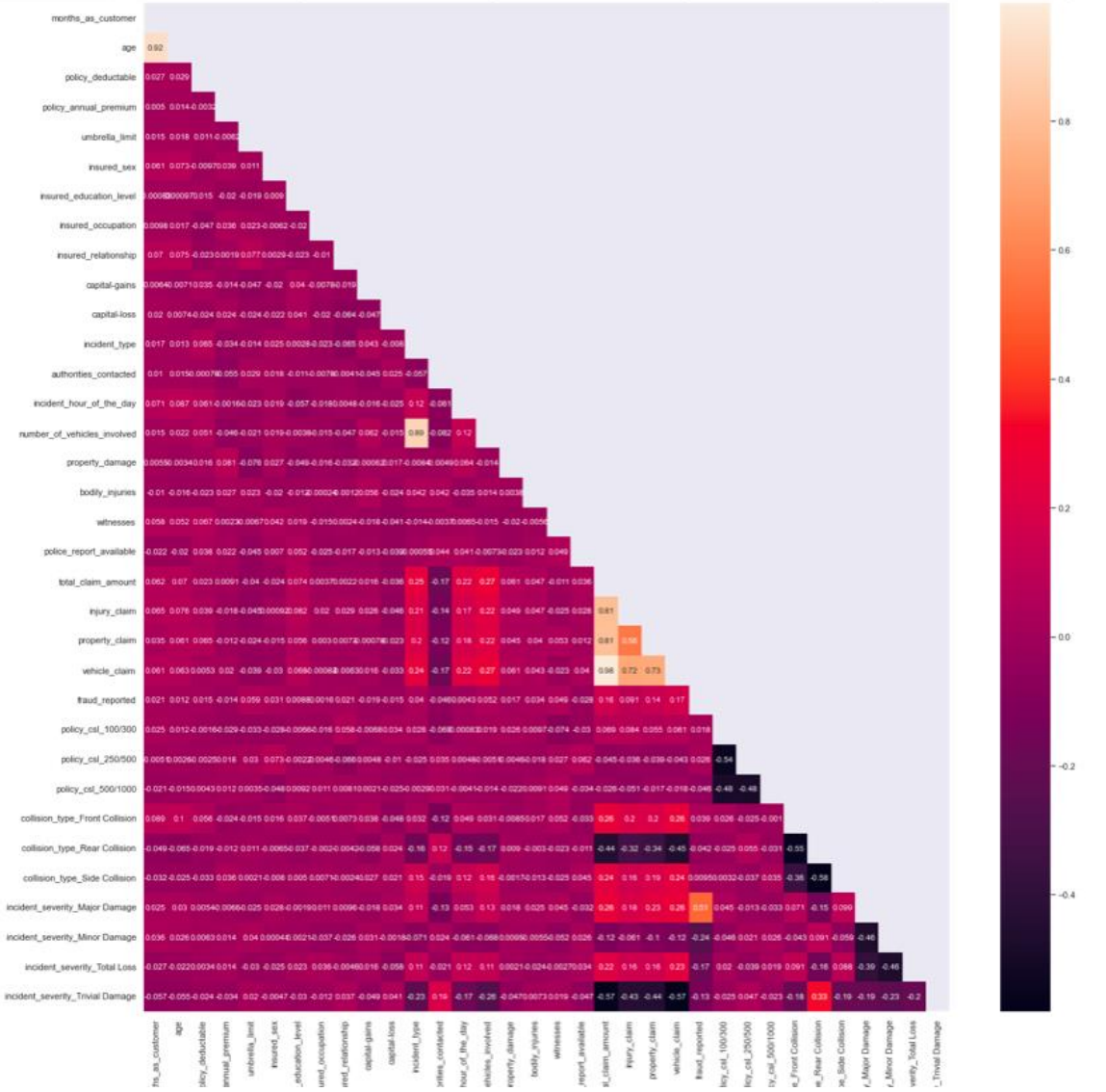
- Dataset is close to Normal Distribution

- Outliers



- Only few outliers are present in our dataset, we can scale them further.

- Correlation



- Found that:
  1. age and months_as_customer are correlated with each other.
  2. total_claim_amount and vehicle_claim are correlated columns.

We are taking the correlation whoes correlation is greater than 90%, So dropping any of the columns which are correlated.

Pre-processing Pipeline

Encoding

- Object columns

```
en = ['insured_sex','policy_csl','insured_education_level','insured_occupation','insured_relationship',
      'incident_type','collision_type','incident_severity','authorities_contacted','property_damage',
      'police_report_available','fraud_reported']
```

- Unique values of every object column

```
 insured_sex
['MALE' 'FEMALE']
 ----------------------------------------

 policy_csl
['250/500' '100/300' '500/1000']
 ----------------------------------------

 insured_education_level
['MD' 'PhD' 'Associate' 'Masters' 'High School' 'College' 'JD']
 ----------------------------------------

 insured_occupation
['craft-repair' 'machine-op-inspct' 'sales' 'armed-forces' 'tech-support'
 'prof-specialty' 'other-service' 'priv-house-serv' 'exec-managerial'
 'protective-serv' 'transport-moving' 'handlers-cleaners' 'adm-clerical'
 'farming-fishing']
 ----------------------------------------

 insured_relationship
['husband' 'other-relative' 'own-child' 'unmarried' 'wife' 'not-in-family']
 ----------------------------------------

 incident_type
['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']
 ----------------------------------------

 collision_type
['Side Collision' 'Rear Collision' 'Front Collision']
 ----------------------------------------

 incident_severity
['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']
 ----------------------------------------

 authorities_contacted
['Police' 'None' 'Fire' 'Other' 'Ambulance']
 ----------------------------------------

 property_damage
['YES' 'NO']
 ----------------------------------------

 police_report_available
['YES' 'NO']
 ----------------------------------------

 fraud_reported
['Y' 'N']
 ----------------------------------------
```

➢ Applying Dummies to column

```
gd = pd.get_dummies(insu[['policy_csl','collision_type','incident_severity']])
insu = pd.concat([insu,gd],axis = 1)
insu.drop(columns = ['policy_csl','collision_type','incident_severity'],axis = 1,inplace = True)
insu.shape
```

```
(1000, 34)
```

➢ Applying LabelEncoder

- LabelEncoder.

```
lb = LabelEncoder()
insu['insured_sex'] = lb.fit_transform(insu['insured_sex'])
insu['insured_education_level'] = lb.fit_transform(insu['insured_education_level'])
insu['insured_occupation'] = lb.fit_transform(insu['insured_occupation'])
insu['insured_relationship'] = lb.fit_transform(insu['insured_relationship'])
insu['authorities_contacted'] = lb.fit_transform(insu['authorities_contacted'])
insu['property_damage'] = lb.fit_transform(insu['property_damage'])
insu['police_report_available'] = lb.fit_transform(insu['police_report_available'])
insu['fraud_reported'] = lb.fit_transform(insu['fraud_reported'])
```

➢ Replacing Values

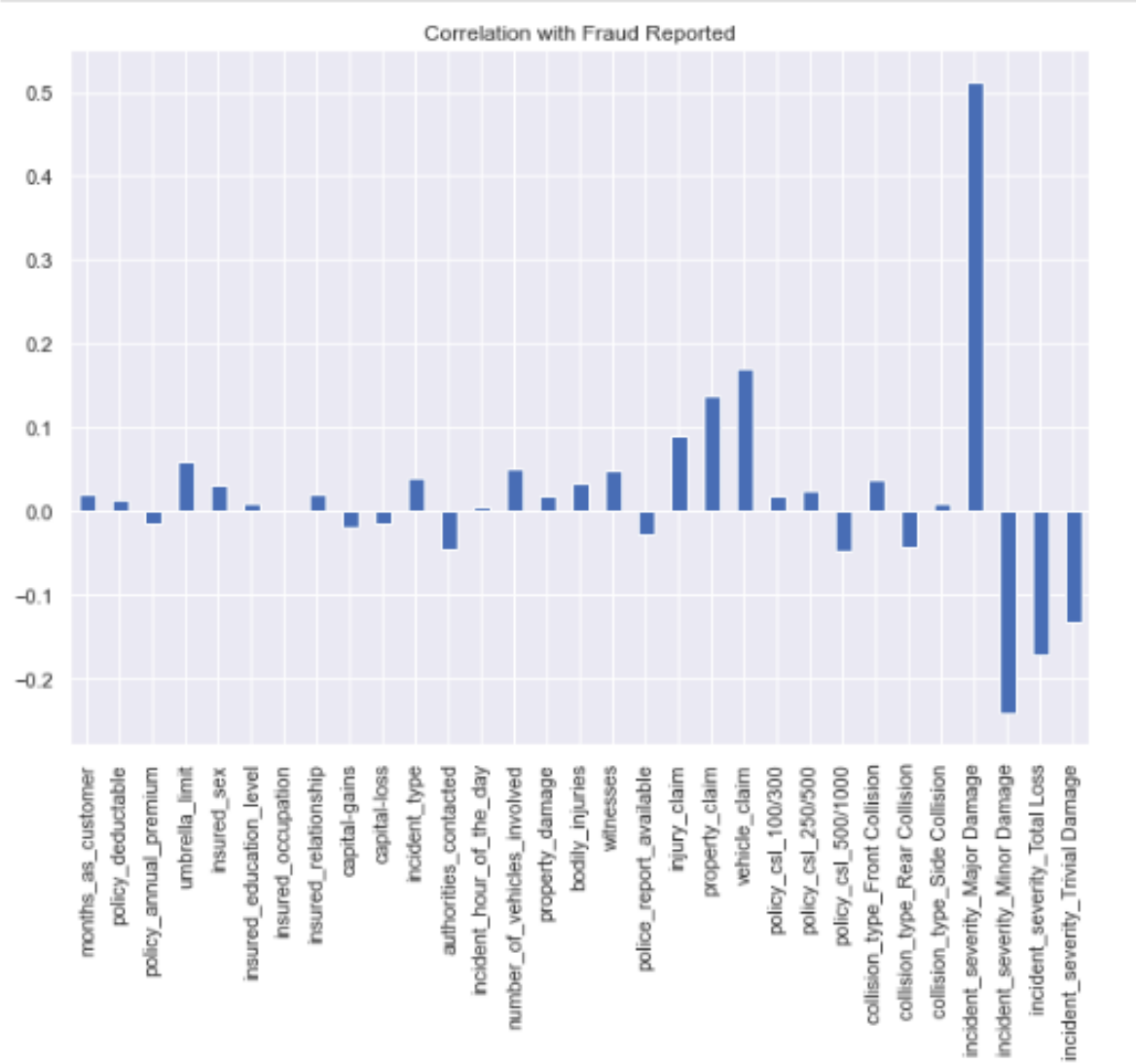
- LabelEncoder.

```
lb = LabelEncoder()
insu['insured_sex'] = lb.fit_transform(insu['insured_sex'])
insu['insured_education_level'] = lb.fit_transform(insu['insured_education_level'])
insu['insured_occupation'] = lb.fit_transform(insu['insured_occupation'])
insu['insured_relationship'] = lb.fit_transform(insu['insured_relationship'])
insu['authorities_contacted'] = lb.fit_transform(insu['authorities_contacted'])
insu['property_damage'] = lb.fit_transform(insu['property_damage'])
insu['police_report_available'] = lb.fit_transform(insu['police_report_available'])
insu['fraud_reported'] = lb.fit_transform(insu['fraud_reported'])
```

- Feature Selection
  - Top 30 Columns (Feature Name & Score)

	Feature Name	Scores
27	incident_severity_Major Damage	355.765552
28	incident_severity_Minor Damage	60.845234
29	incident_severity_Total Loss	30.151126
20	vehicle_claim	29.718214
19	property_claim	19.327729
30	incident_severity_Trivial Damage	17.561740
18	injury_claim	8.328776
3	umbrella_limit	3.441452
13	number_of_vehicles_involved	2.689100
16	witnesses	2.451026
23	policy_csl_500/1000	2.120430
11	authorities_contacted	2.098018
25	collision_type_Rear Collision	1.783068
10	incident_type	1.628536
24	collision_type_Front Collision	1.495646
15	bodily_injuries	1.146656
4	insured_sex	0.952144
17	police_report_available	0.770117
22	policy_csl_250/500	0.662855
7	insured_relationship	0.442135
0	months_as_customer	0.421370
8	capital-gains	0.366990
21	policy_csl_100/300	0.340613
14	property_damage	0.295389
9	capital-loss	0.220519
1	policy_deductable	0.219163
2	policy_annual_premium	0.209284
26	collision_type_Side Collision	0.089739
5	insured_education_level	0.077435
12	incident_hour_of_the_day	0.018590

- Graph of correlation with Target Column.



1. Taking top 16 columns according to the score and graph
2. Also taking columns whose score is greater than 1.

– Top 16 Column & Final Dataset

```
xbest = x[['incident_severity_Major Damage','incident_severity_Minor Damage','incident_severity_Total Loss',
           'vehicle_claim','property_claim','incident_severity_Trivial Damage','injury_claim','umbrella_limit',
           'number_of_vehicles_involved','witnesses','policy_csl_500/1000','authorities_contacted',
           'collision_type_Rear Collision','incident_type','collision_type_Front Collision','bodily_injuries']]
```

```
xbest
```

	incident_severity_Major Damage	incident_severity_Minor Damage	incident_severity_Total Loss	vehicle_claim	property_claim	incident_severity_Trivial Damage	injury_claim	umbrella_limit	nu
0	1	0	0	52080	13020	0	6510	0	
1	0	1	0	3510	780	0	780	5000000	
2	0	1	0	23100	3850	0	7700	5000000	
3	1	0	0	50720	6340	0	6340	6000000	
4	0	1	0	4550	650	0	1300	6000000	
...	...	...	...	...	...	...	...	...	
995	0	1	0	61040	8720	0	17440	0	
996	1	0	0	72320	18080	0	18080	0	
997	0	1	0	52500	7500	0	7500	3000000	
998	1	0	0	36540	5220	0	5220	5000000	
999	0	1	0	3680	920	0	460	0	

1000 rows × 16 columns

➢ Scaling the Features (StandardScaler)

```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(xbest)
```

➢ Train Test Split

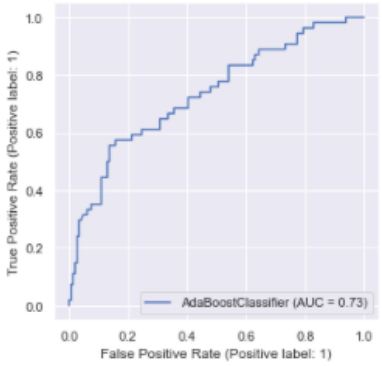
```
x_train,x_test,y_train,y_test = train_test_split(x_scaled, y, test_size = 0.20, random_state = 90)
```

Building Machine Learning Models

- AdaBoost Classifier

```
ada.fit(x_train,y_train)
score(ada, x_train,x_test,y_train,y_test,train = True)
score(ada, x_train,x_test,y_train,y_test,train = False)
```

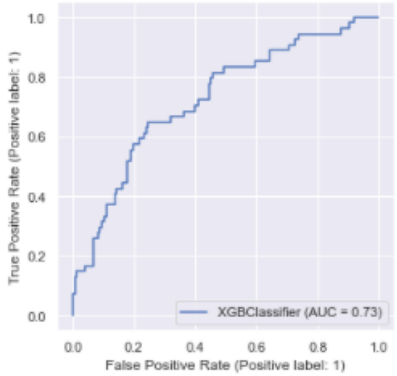
```
 ----- Train Result -----

Accuracy Score: 0.83

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       607
           1       0.68      0.56      0.62       193

    accuracy                           0.83       800
   macro avg       0.77      0.74      0.75       800
weighted avg       0.82      0.83      0.82       800


 ----- Confusion matrix -----
 [[555  52]
 [ 84 109]]

 ----- Test Result -----

Accuracy Score: 0.745

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       146
           1       0.54      0.35      0.43        54

    accuracy                           0.74       200
   macro avg       0.67      0.62      0.63       200
weighted avg       0.72      0.74      0.73       200


 ----- Confusion matrix -----
 [[130  16]
 [ 35  19]]

 ----- Roc Curve -----
```



- XGBoost Classifier.

```
xgb.fit(x_train,y_train)
score(xgb, x_train,x_test,y_train,y_test,train = True)
score(xgb, x_train,x_test,y_train,y_test,train = False)
```

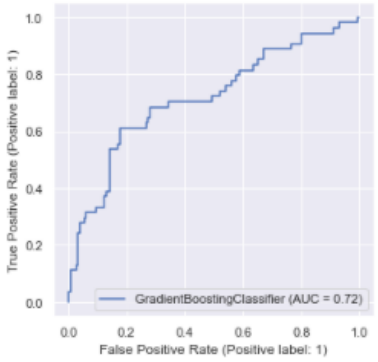
```
 ----- Train Result -----

Accuracy Score: 1.0

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       607
           1       1.00      1.00      1.00       193

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800


 ----- Confusion matrix -----
 [[607   0]
 [  0 193]]

 ----- Test Result -----

Accuracy Score: 0.74

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.78      0.90      0.84       146
           1       0.53      0.30      0.38        54

    accuracy                           0.74       200
   macro avg       0.65      0.60      0.61       200
weighted avg       0.71      0.74      0.71       200


 ----- Confusion matrix -----
 [[132  14]
 [ 38  16]]

 ----- Roc Curve -----
```



- GradientBoosting Classifier.

```
gb.fit(x_train,y_train)
score(gb, x_train,x_test,y_train,y_test,train = True)
score(gb, x_train,x_test,y_train,y_test,train = False)
```

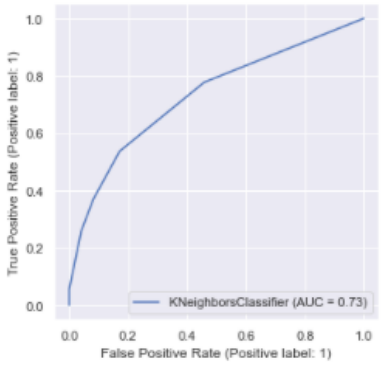
```
 ----- Train Result -----

Accuracy Score: 0.91875

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       607
           1       0.89      0.76      0.82       193

    accuracy                           0.92       800
   macro avg       0.91      0.87      0.88       800
weighted avg       0.92      0.92      0.92       800


 ----- Confusion matrix -----
 [[588  19]
 [ 46 147]]

 ----- Test Result -----

Accuracy Score: 0.735

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       146
           1       0.51      0.37      0.43        54

    accuracy                           0.73       200
   macro avg       0.65      0.62      0.63       200
weighted avg       0.71      0.73      0.72       200


 ----- Confusion matrix -----
 [[127  19]
 [ 34  20]]

 ----- Roc Curve -----
```



- KNeighbors Classifier.

```
knn.fit(x_train,y_train)
score(knn, x_train,x_test,y_train,y_test,train = True)
score(knn, x_train,x_test,y_train,y_test,train = False)
```

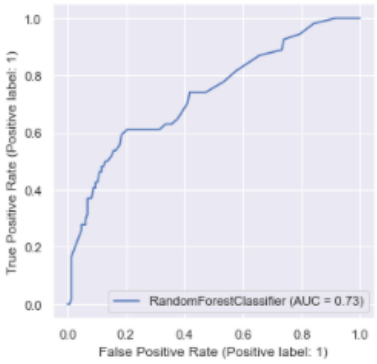
```
 ----- Train Result -----

Accuracy Score: 0.84875

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       607
           1       0.72      0.60      0.66       193

    accuracy                           0.85       800
   macro avg       0.80      0.76      0.78       800
weighted avg       0.84      0.85      0.84       800


 ----- Confusion matrix -----
 [[563  44]
 [ 77 116]]

 ----- Test Result -----

Accuracy Score: 0.77

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.80      0.92      0.85       146
           1       0.62      0.37      0.47        54

    accuracy                           0.77       200
   macro avg       0.71      0.64      0.66       200
weighted avg       0.75      0.77      0.75       200


 ----- Confusion matrix -----
 [[134  12]
 [ 34  20]]

 ----- Roc Curve -----
```



- RandomForest Classifier.

```
rf.fit(x_train,y_train)
score(rf, x_train,x_test,y_train,y_test,train = True)
score(rf, x_train,x_test,y_train,y_test,train = False)
```

```
 ----- Train Result -----

Accuracy Score: 1.0

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       607
           1       1.00      1.00      1.00       193

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800


 ----- Confusion matrix -----
 [[607   0]
 [  0 193]]

 ----- Test Result -----

Accuracy Score: 0.775

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       146
           1       0.63      0.41      0.49        54

    accuracy                           0.78       200
   macro avg       0.72      0.66      0.67       200
weighted avg       0.76      0.78      0.76       200


 ----- Confusion matrix -----
 [[133  13]
 [ 32  22]]

 ----- Roc Curve -----
```



Concluding Remarks

- RandomForest Classifier is giving the best score comparatively and all the metrics and curves are in favor of RandomForest Classifier.

- Original vs Predicited

```
a_rfc = np.array(y_test)
predicted_rfc = np.array(rf.predict(x_test))
df_rfc = pd.DataFrame({'Original':a_rfc,'Predicted':predicted_rfc})
df_rfc
```

	Original	Predicted
0	0	0
1	0	0
2	0	0
3	1	0
4	0	0
...	...	...
195	0	0
196	0	1
197	0	0
198	0	0
199	0	0

200 rows × 2 columns

➢ Cross-Validation

```
k_f = KFold(n_splits = 5,shuffle = True)
k_f
```

```
KFold(n_splits=5, random_state=None, shuffle=True)
```

```
for train,test in k_f.split(insu):
    print('Train:',train,'\ntest:',test)
```

```
Train: [  2   3   4   5   6   7   8   9  10  12  13  14  15  16  18  19  22  23
  24  25  26  27  28  29  30  31  32  34  35  36  37  38  39  40  41  42
  44  46  47  48  50  51  53  56  57  58  59  60  61  62  63  64  65  66
  67  68  69  70  71  72  73  76  78  80  83  84  85  86  87  88  90  91
  92  93  94  95  96  97  98  99 100 101 102 106 107 108 110 111 112 113
 114 115 117 118 120 121 122 124 125 126 129 130 131 132 135 136 137 138
 139 140 141 142 143 144 145 146 147 150 151 152 153 155 156 157 158 159
 160 161 162 163 164 165 166 168 170 171 173 174 175 177 179 180 181 182
 183 184 185 186 187 188 189 190 191 192 193 194 195 196 198 199 201 202
 203 204 206 208 209 211 212 213 214 215 216 217 218 219 220 221 222 223
 224 225 227 228 231 232 233 235 236 237 238 239 240 241 243 244 245 246
 249 250 251 252 253 254 255 256 257 258 259 261 263 264 265 266 269 270
 271 274 275 276 277 278 280 281 282 283 284 285 286 287 288 289 290 293
 294 295 296 297 298 299 300 301 302 303 306 307 308 309 310 313 315 316
 317 320 321 323 325 326 328 330 331 332 333 334 335 338 339 341 343 347
 349 350 351 352 353 355 356 357 358 360 362 363 364 365 366 367 369 370
 371 372 373 374 376 377 378 379 380 381 382 385 386 389 390 391 393 394
 395 396 397 398 399 402 403 405 406 407 408 409 410 411 412 413 414 417
 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 436
 438 439 440 441 442 443 445 447 448 449 450 451 452 453 455 456 457 459
 460 461 462 464 465 466 467 468 469 470 473 476 477 478 479 480 481 483
 484 485 486 487 488 489 490 491 493 494 496 497 498 499 500 502 503 504
 505 506 507 508 509 511 512 514 515 516 517 518 519 520 521 522 523 524
 525 526 527 529 530 531 532 533 535 536 537 539 540 541 542 543 544 545
 546 547 548 549 552 554 555 556 557 558 560 562 563 564 565 566 567 569
 570 571 572 573 574 576 577 578 579 580 581 582 583 585 587 588 589 591
 592 593 594 595 598 599 601 603 604 605 606 607 608 609 610 612 613 614
 615 616 617 618 619 620 622 623 624 625 626 627 628 629 630 631 632 633
 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652
 653 655 656 657 658 659 660 661 662 663 664 665 669 670 672 673 674 676
 678 679 680 681 683 684 685 686 687 688 689 690 691 692 693 694 695 696
 697 698 699 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715
 716 717 718 721 722 724 725 726 729 730 731 732 733 734 736 737 738 739
 740 741 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758
 759 761 762 763 764 767 768 769 770 773 775 776 777 778 779 780 782 784
 785 786 788 789 790 791 792 796 798 799 801 802 804 805 806 807 808 809
 811 812 813 814 815 816 818 819 820 822 823 824 828 829 831 832 835 836
 837 838 839 840 841 842 843 844 846 848 849 850 851 852 854 855 856 857
 858 859 860 861 862 863 865 866 869 870 871 872 873 874 875 876 877 878
 881 882 883 884 885 886 887 888 889 890 891 892 893 894 896 897 898 899
 900 901 902 903 904 906 907 909 910 912 913 914 915 916 917 918 920 921
 922 923 924 925 927 930 931 932 934 935 936 937 938 939 940 941 942 943
 945 946 947 948 949 950 951 953 954 955 956 957 958 959 961 963 965 966
 970 971 972 973 974 976 977 978 979 980 981 982 983 984 985 986 989 990
 991 992 993 994 995 996 997 998]
test: [  0   1  11  17  20  21  33  43  45  49  52  54  55  74  75  77  79  81
  82  89 103 104 105 109 116 119 123 127 128 133 134 148 149 154 167 169
 172 176 178 197 200 205 207 210 226 229 230 234 242 247 248 260 262 267
 268 272 273 279 291 292 304 305 311 312 314 318 319 322 324 327 329 336
 337 340 342 344 345 346 348 354 359 361 368 375 383 384 387 388 392 400
```

➢ Cross Validation Score

```
cross_val_score(rf,x_scaled,y,cv = 5).mean()
```

```
0.78
```

- cross validation score and model's accuracy score is almost equal, so we can consider that our model isn't overfitting/underfitting.

- Hyperparameter Tuning

```
param = {'n_estimators':range(0,100,10),
         'ccp_alpha':[0.0,0.2,0.4,0.5,0.7,0.8,1.0],
         'criterion':["gini", "entropy"],
         'max_features':["sqrt", "log2", None],
         }
```

```
grid = GridSearchCV(rf,param_grid = param)
grid.fit(x_train,y_train)
print('Best Params = ',grid.best_params_)
```

```
Best Params =  {'ccp_alpha': 0.0, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators': 70}
```

```
rf_hyp = RandomForestClassifier(ccp_alpha = 0.0, criterion = 'gini', max_features = 'sqrt', n_estimators = 70)
```

```
rf_hyp.fit(x_train,y_train)
score(rf_hyp, x_train,x_test,y_train,y_test,train = True)
score(rf_hyp, x_train,x_test,y_train,y_test,train = False)
```

```
 ----- Train Result -----

Accuracy Score: 1.0

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       607
           1       1.00      1.00      1.00       193

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800


 ----- Confusion matrix -----
 [[607   0]
 [  0 193]]

 ----- Test Result -----

Accuracy Score: 0.76

 ----- Classification Report -----
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       146
           1       0.58      0.39      0.47        54

    accuracy                           0.76       200
   macro avg       0.69      0.64      0.66       200
weighted avg       0.74      0.76      0.74       200


 ----- Confusion matrix -----
 [[131  15]
 [ 33  21]]
```

  - Post tuning result are not better, so taking default parameters only as the best parameters.

- **Library Used:**
  - Pandas
  - Numpy
  - Seaborn
  - Matplotlib
  - LabelEncoder
  - StandardScaler
  - SelectKBest
  - f_classif
  - Train Test Split
  - GridSearchCV
  - metrics
  - AdaBoostClassifier
  - GradientBoostingClassifier
  - RandomForestClassifier
  - KNeighborsClassifier
  - XGBClassifier
  - KFold
  - cross_val_score

- **Project Link:** **https://github.com/Miteshverma9/Project/blob/main/Projects/Insurance%20Claims-%20Fraud%20Detection.ipynb**