Insurance Claims
Fraud Detection

NOVEMBER 2

Glenn Nigel Ebenezer
Batch No 1844

Contents

Insurance Claims- Fraud Detection

Problem Statement:

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.

In this project, you are provided a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

In this example, you will be working with some auto insurance data to demonstrate how you can create a predictive model that predicts if an insurance claim is fraudulent or not.

> ***"Solution: A Classic Classification model to be used to detect whether the claim is Fraudulant or Genuine"***

Link to the Dataset provided:

Github Link to the Jupyter Notebook

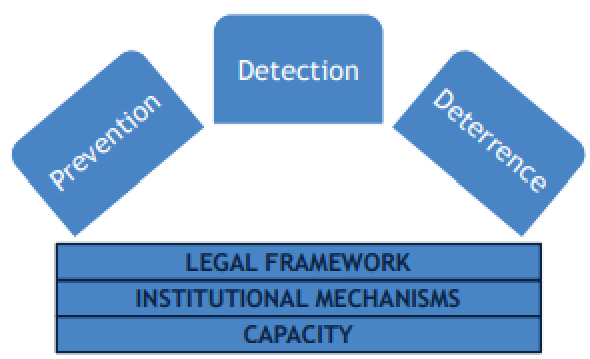
I Problem Definition

One of the most challenging problems in the niche of insurance is fraudulent claims, and the ability to detect whether the claim is a fraud or genuine case, over the years, was and is still a herculean task. Although, in these modern times, with the advent of Machine learning (ML) in Datascience, we have the capability to detect these frauds with the help of ML by studying trends and data analysis.

The Solution for the problem statement requires us to figure out the best and most efficient way to build a model using ML to automatically, study the data and give us a fraud alert whenever it is detected. With the number of insurance policies and claims that are out there, it is very important that a fraudulent claim be flagged in time.

As we know, it is merely impossible to achieve a 100% accurate model that is efficient and not cost the insurance company all its profits. We can at least ensure we build a model that can red flag any suspicious case, even if it's a false positive, the number of cases for manual scrutiny would drastically reduce.

In India alone, the Insurance Regulatory and Development Authority (IRDA), stated on record that Insurance companies lose over USD 6.25 billion to frauds which results in higher premiums for genuine consumers. And hence devised an "Anti-Fraud Conceptual Framework" which rests on 3 Pillars, of which Detection forms the main Cornerstone. It is only with the right, and timely detection that can anything else, like prevention or deterrence, or the other institutional and legal mechanisms come into play.



Insurance companies that haven't implemented high-tech solutions yet should hurry up. Artificial intelligence (AI) and predictive analytics are shaping the future of the entire industry, giving significant competitive advantages to organizations that are already making use of these technologies.

In this article, we'll dive into the newest insurance fraud prevention methods, uncover the benefits of predictive analytics, and feature the three best insurance claims fraud detection software tools.

Top 3 Insurance Claims Fraud Detection Software

- FRISS Fraud Detection
- SAS Detection and Investigation for Insurance
- SEON

Although these are more detailed and elaborate programs out there, can be used when the stakes are high, but a far less expensive with decent accuracy maybe built and used when the stakes are rather low. This is what we will consider doing today using simple Machine Learning strategies in Python.

In this prediction model, we will be using basically 4 simple classification models:

- Decision Tree Classifier
- Linear Regression
- K Nearest Classifier
- Random Forest Classifier

We will also consider Hyperparameter tuning, to increase the efficiency and otherwise overall effectivess by knowing the actual scores incase the data/model led the result to over or under fitting.

It is a given that, as much as we try and get the best success rates, it all boils down to the data collection and manipulation. As Data Scientists we can do very little when it comes to the data that is given to us, until and unless we are scraping it, of course. But Data Manipulation and processing is the main factor attributing to success. And along with a skillset in python, it will be really helpful if we have knowledge on the topic we are working on, also known as –Domain Knowledge.

One of the reasons for having picked this project to draft this detailed report is because of the Domain knowledge I have acquired, having cleared the IRDA Entrance exam multiple times.

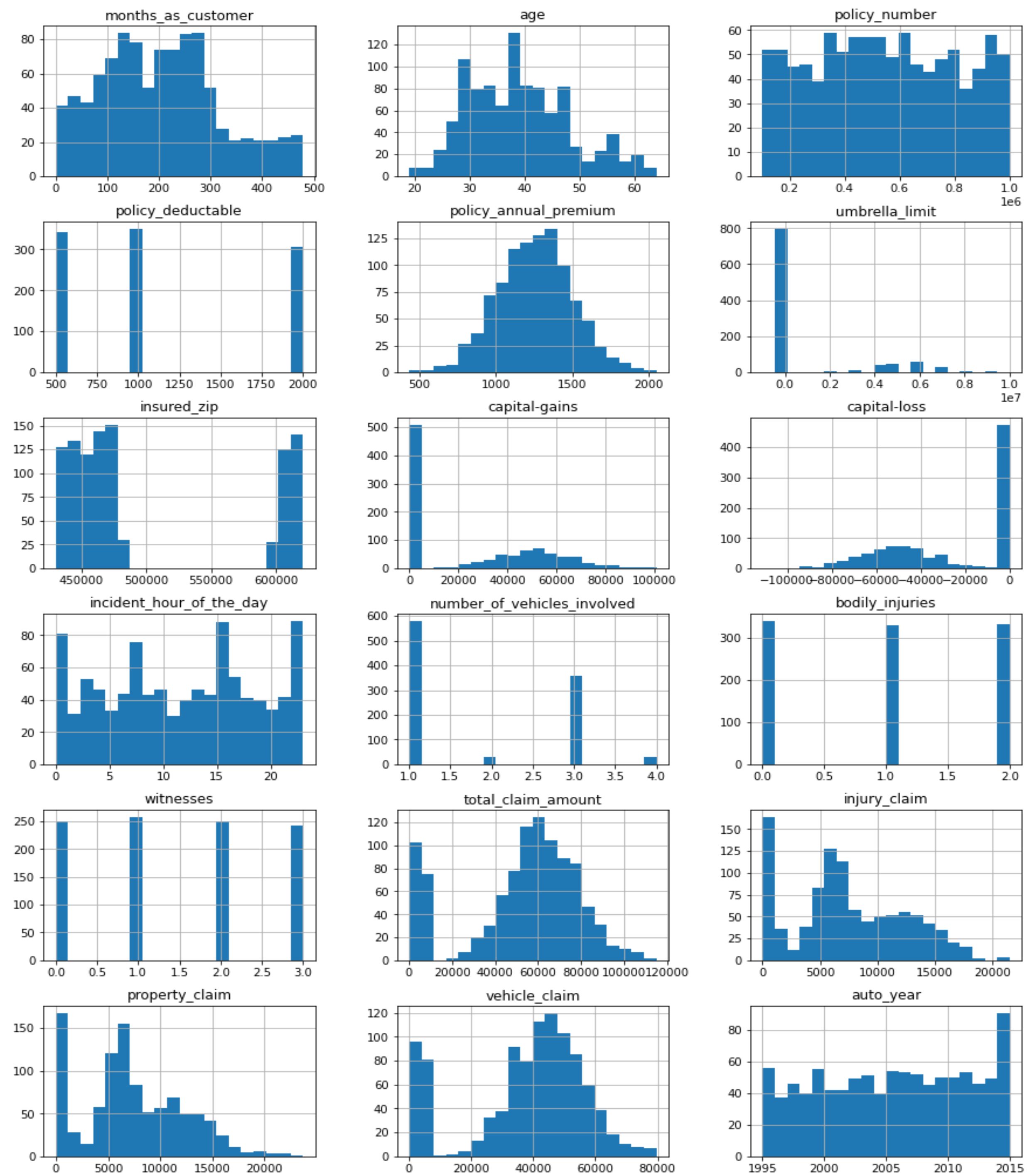
II Data Analysis

- The Dataset was imported into the Jupyter Notebook as a CSV file, and on initial inspection, we saw that there were 40 columns and 1000 records. The Columns were a mix of Float, Int and object datatypes and there were no missing or null values.
- On studying the standard deviation from the mean in the columns with int and float dtype, it was clear that there were a few outliers as the standard deviation was disproportionate. Added to this the increments between the quartiles also were uneven in some columns. This anyway was checked and attended to at a later stage.
- There was one column that was completely empty, also a nominal data columns like “policy numbers” that can be removed.
- Comparing the categorical and Numerical Data columns, it was clear that python had on the basis of their Dtype had predicted if it was an object or non-object, which in this case would not work, as there were columns with limited numerical values, that were actually categorical columns. This too needed to be addressed.
- The 'insured_zip’ column, although not nominal, had almost all unique values, and may not contribute well towards the model building, maybe removed after confirmation via EDA.
- Looking at the Dataset and the columns and column data, along with the number of unique values in each column, we were able to estimate a threshold at which we can categorize a column as categorical or not based on whether it contains lesser than the threshold of 40 unique values.
- Finally, although there were no null or missing values, there were a lot of ”?” and “0” in the columns that needed to be attended to as well.

III Exploratory Data Analysis(EDA)

-using Visualisation Techniques

- We first plot a Histogram plot of all columns



- We then began the EDA be separating the Categorical and Numerical data columns, based on the number of unique values.

```
In [12]:
#User Defined Function to Seperate Catagorical and Numerical data Columns based on number of unique values.
def num_cat(df):
    num=[]
    cat=[]
    count=df.nunique()
    for i in df.columns:
        if count[i]>40:
            num.append(i)
        else:
            cat.append(i)
    return(num,cat)
```

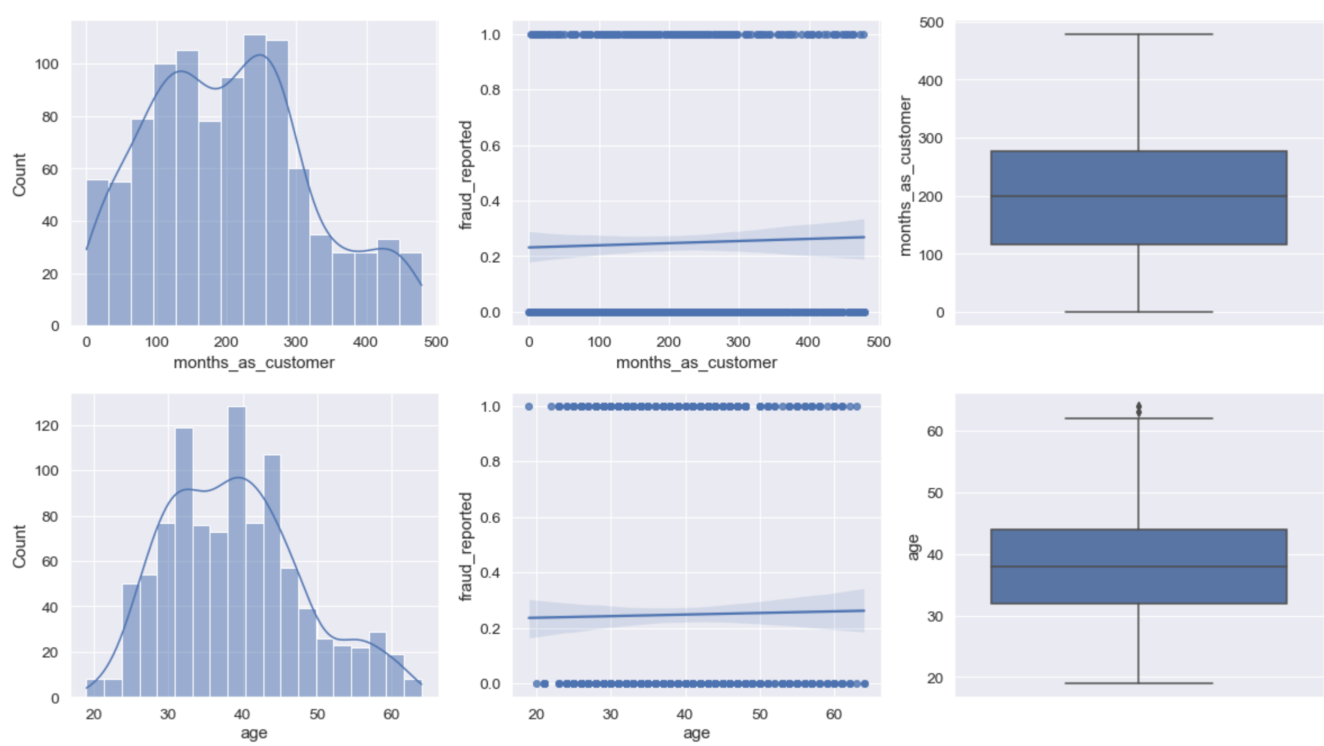
```
In [13]:
numer_cols=num_cat(data)[0]
cat_cols=num_cat(data)[1]
print(f'Numerical data columns: \n',num_cat(data)[0])
print(f'\n\n Categorical data columns: \n',num_cat(data)[1])
```

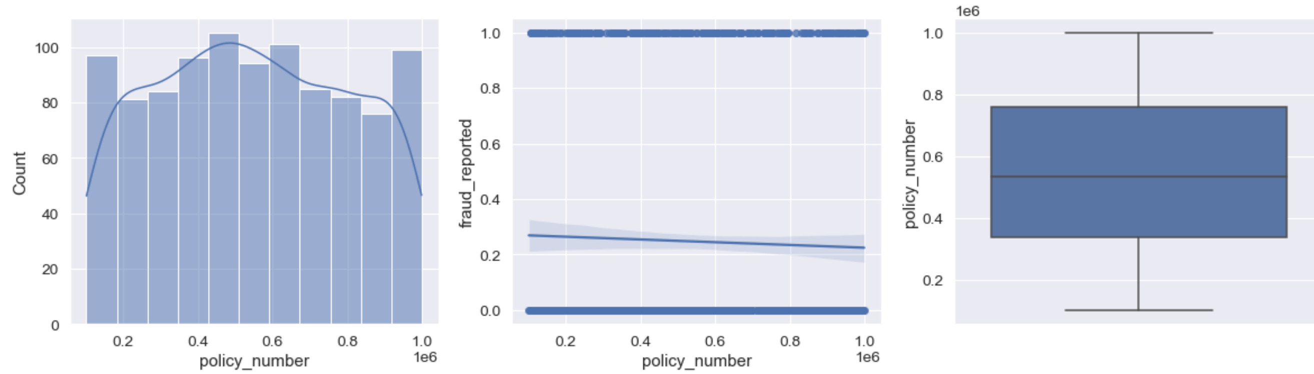
```
Numerical data columns:
 ['months_as_customer', 'age', 'policy_number', 'policy_bind_date', 'policy_annual_premium', 'insured_zip', 'capital-gains', 'c
apital-loss', 'incident_date', 'incident_location', 'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_claim']


 Categorical data columns:
 ['policy_state', 'policy_csl', 'policy_deductable', 'umbrella_limit', 'insured_sex', 'insured_education_level', 'insured_occup
ation', 'insured_hobbies', 'insured_relationship', 'incident_type', 'collision_type', 'incident_severity', 'authorities_contact
ed', 'incident_state', 'incident_city', 'incident_hour_of_the_day', 'number_of_vehicles_involved', 'property_damage', 'bodily_i
njuries', 'witnesses', 'police_report_available', 'auto_make', 'auto_model', 'auto_year', 'fraud_reported']
```

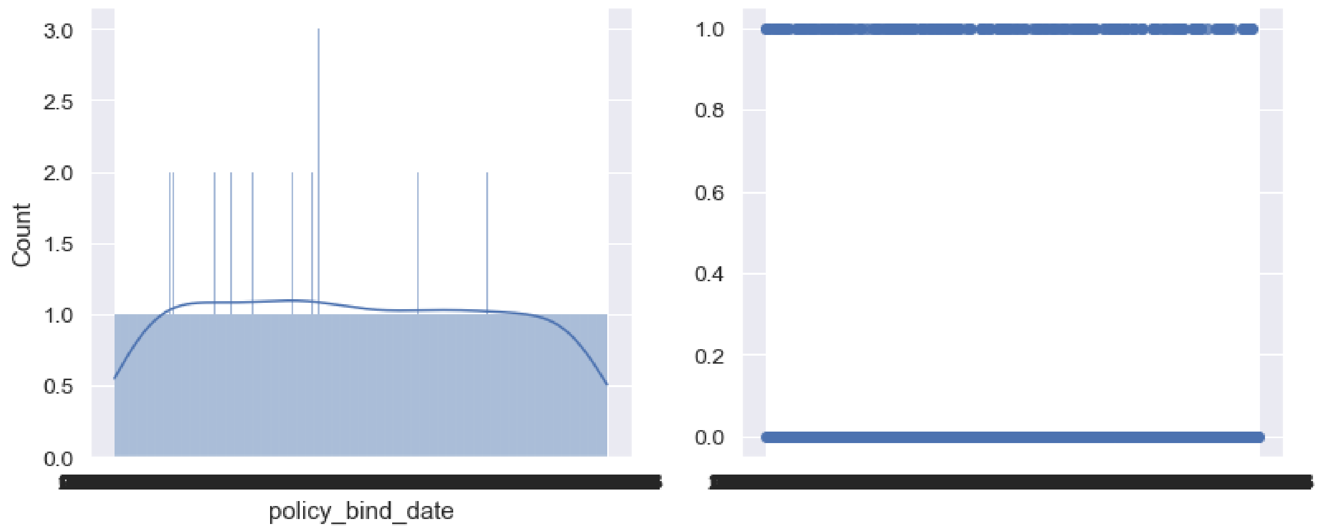
- Based on this we plotted a Regplot, Boxplot and Histplot of all the Numerical data Columns using another user defined function.

```
In [24]:
#User ddefined function for plotting Numerical Data Columns
for i in data[numer_cols].columns:
    plt.figure(figsize=(20,5))
    plt.subplot(1,3,1)
    sns.histplot(x=i,data=data,kde=True)
    plt.subplot(1,3,2)
    sns.regplot(x=i,y=data.fraud_reported,data=data)
    plt.subplot(1,3,3)
    sns.boxplot(y=i,data=data)
    plt.show()
```

Results shows below







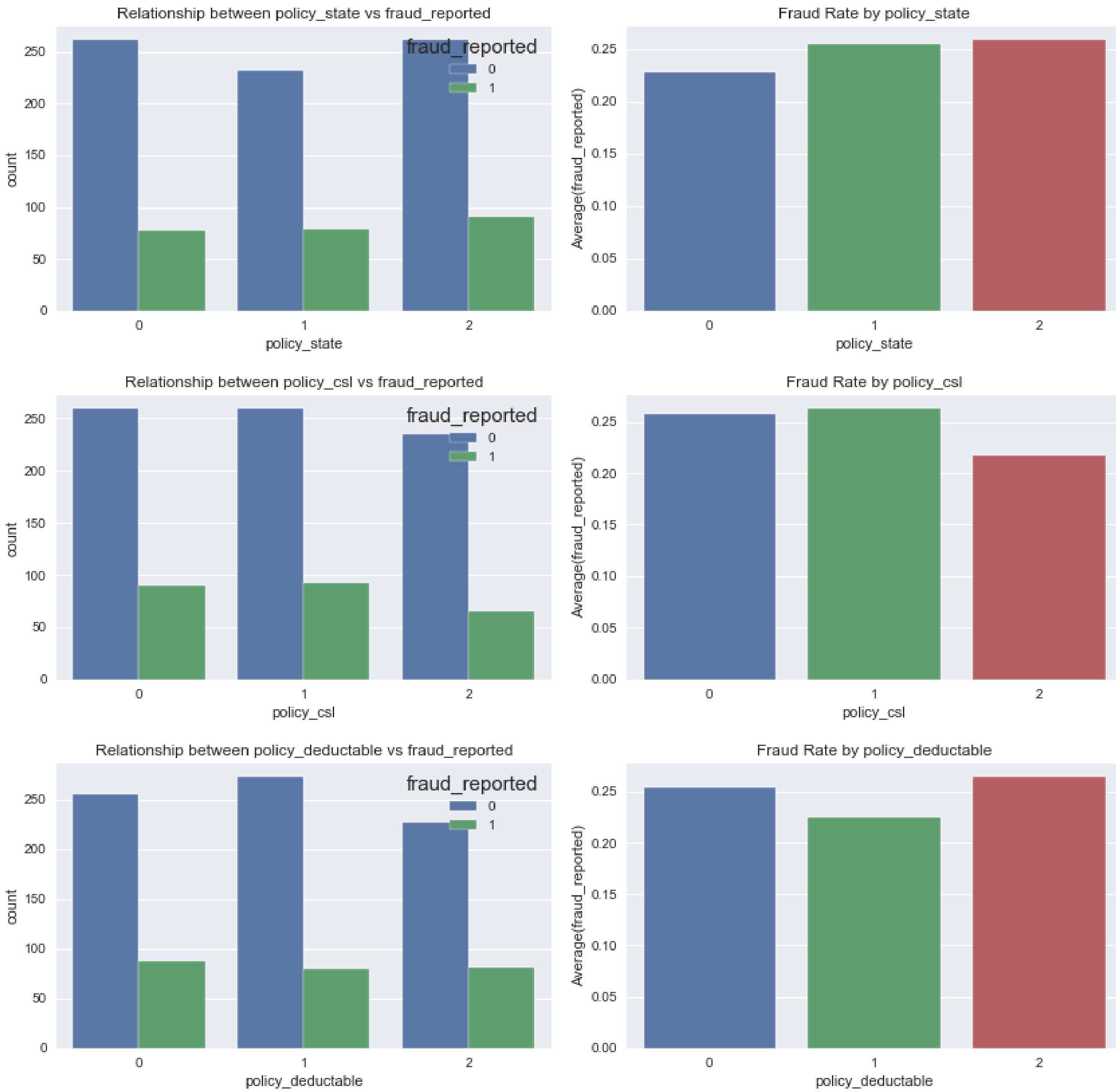
from the above it is clear that it is clear that none of the numerical columns, except 'vehicle_claim', 'property_claim','injust claim','total_claim_amount' have any relationship with the target.

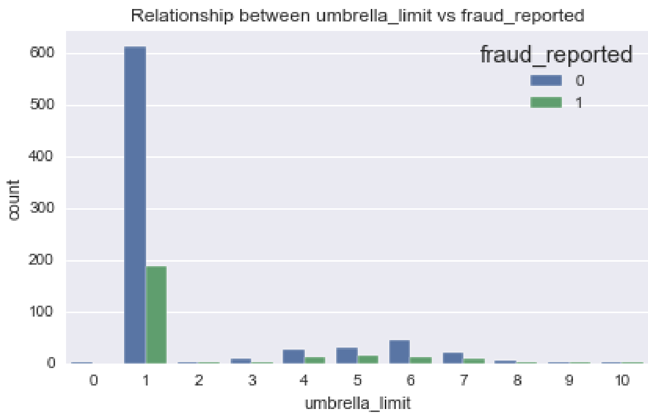
- Next we will plot the Categorical data columns using Countplot to understand the relationship between the feature and label and barplot to understand the attributes in the feature.

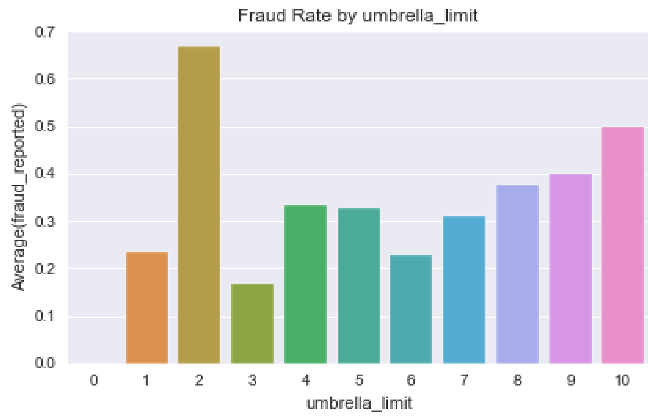
In [148]:
```
#Lets make a function to visualize categorical data
def Discrete_plots(dfrme, feature_c,invert_axis = False, label = "fraud_reported"):
    fig, ax = plt.subplots(ncols= 2, figsize = (12,4))
    if invert_axis == False:
        sns.barplot(x = feature_c, y = label ,data=dfrme,ci=None)
    else:
        sns.barplot(y = feature_c, x = label ,data=dfrme,ci=None)

    if invert_axis == False:
        sns.countplot(x = feature_c, data=dfrme,hue="fraud_reported",ax=ax[0])
    else:
        sns.countplot(y = feature_c, data=dfrme,hue="fraud_reported",ax=ax[0])

    ax[0].set_title("Relationship between " + feature_c + " vs " + label)
    ax[1].set_title("Fraud Rate by {}".format(feature_c))
    ax[1].set_ylabel("Average(fraud_reported)")
    plt.tight_layout()

    plt.show()
```

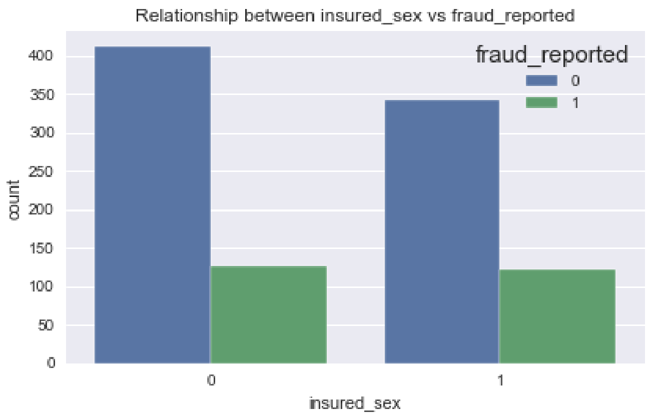
In [149]:
```
for i in data[cat_cols].columns:
    Discrete_plots(data[cat_cols],feature_c=i)
```

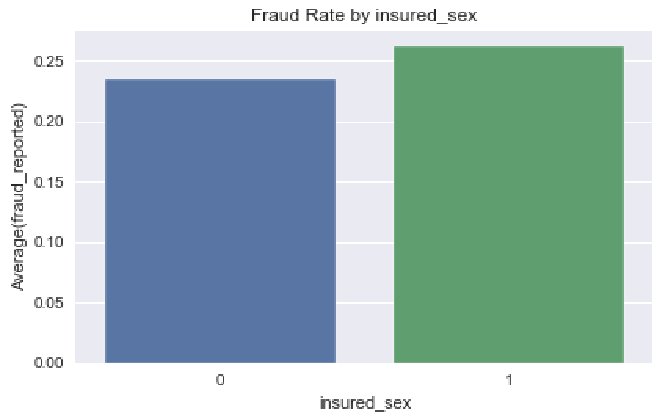
Below are the generated plots:

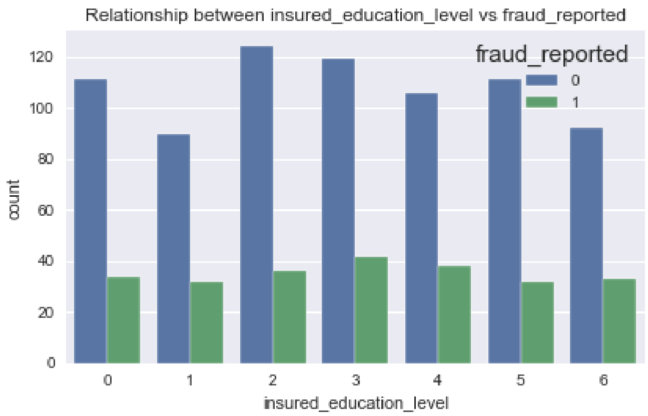


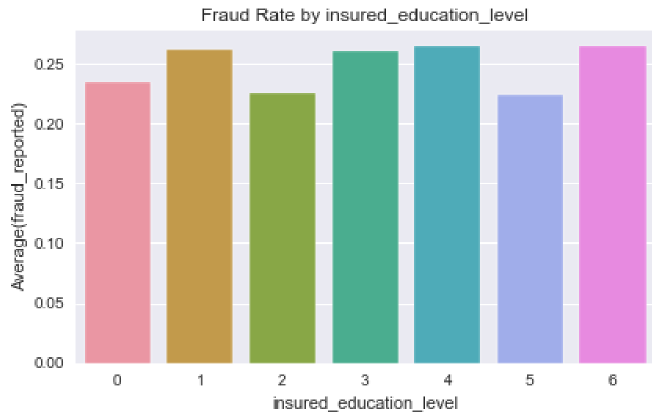

Relationship between umbrella_limit vs fraud_reported
fraud_reported
0
1
count
umbrella_limit

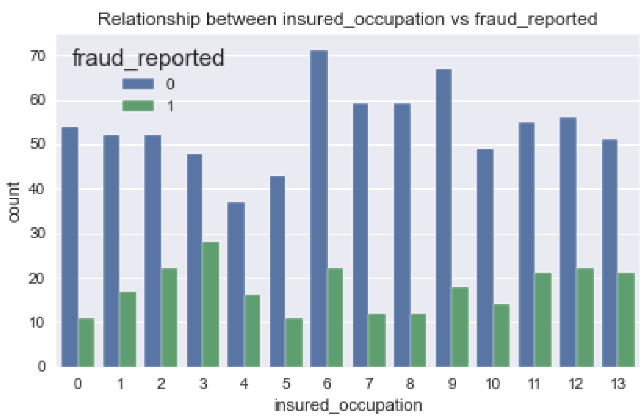

Fraud Rate by umbrella_limit
Average(fraud_reported)
umbrella_limit

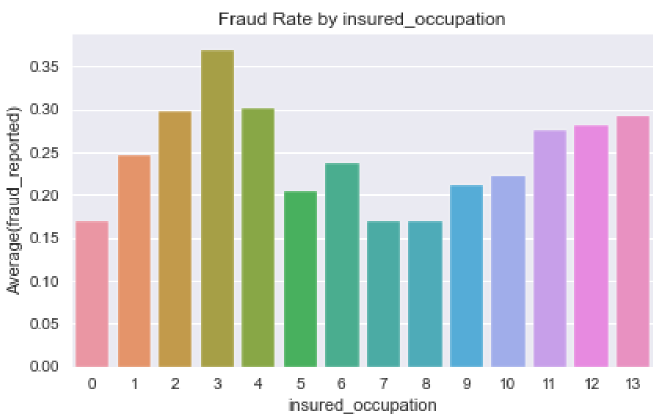

Relationship between insured_sex vs fraud_reported
fraud_reported
0
1
count
insured_sex


Fraud Rate by insured_sex
Average(fraud_reported)
insured_sex

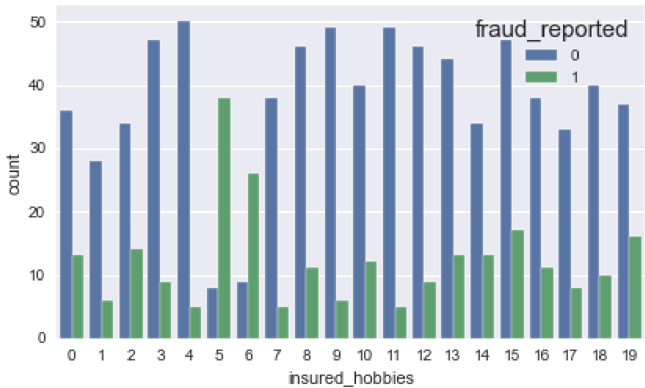

Relationship between insured_education_level vs fraud_reported
fraud_reported
0
1
count
insured_education_level


Fraud Rate by insured_education_level
Average(fraud_reported)
insured_education_level

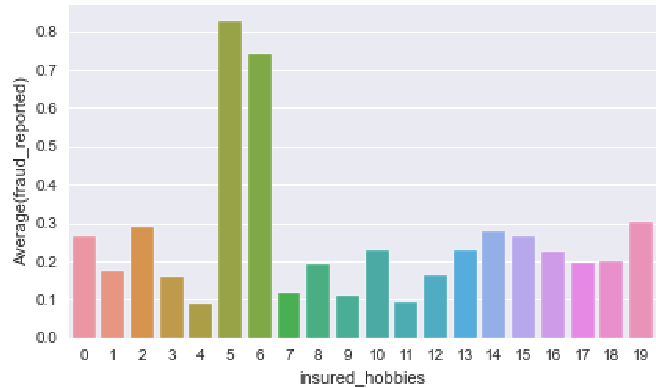

Relationship between insured_occupation vs fraud_reported
fraud_reported
0
1
count
insured_occupation


Fraud Rate by insured_occupation
Average(fraud_reported)
insured_occupation

Relationship between insured_hobbies vs fraud_reported



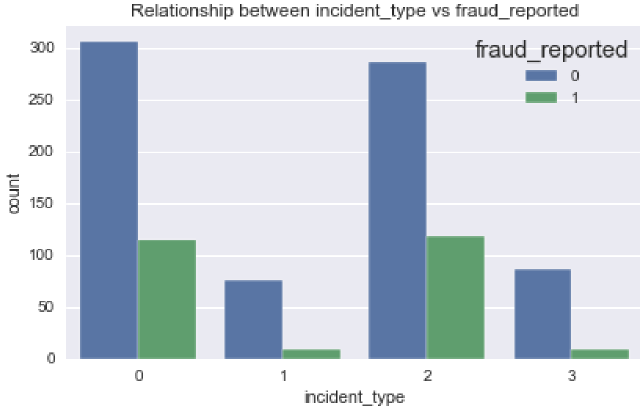
Fraud Rate by insured_hobbies



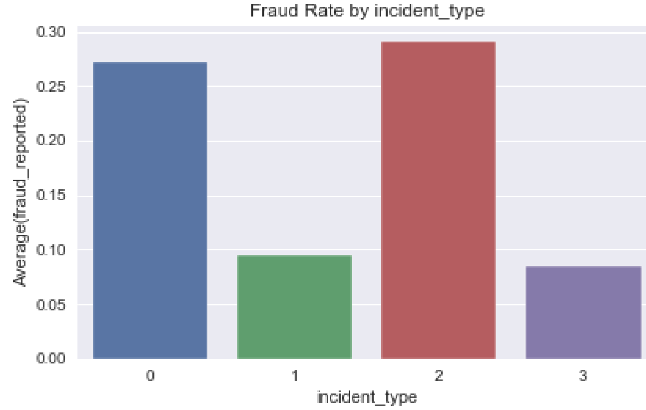
Relationship between insured_relationship vs fraud_reported

Fraud Rate by insured_relationship

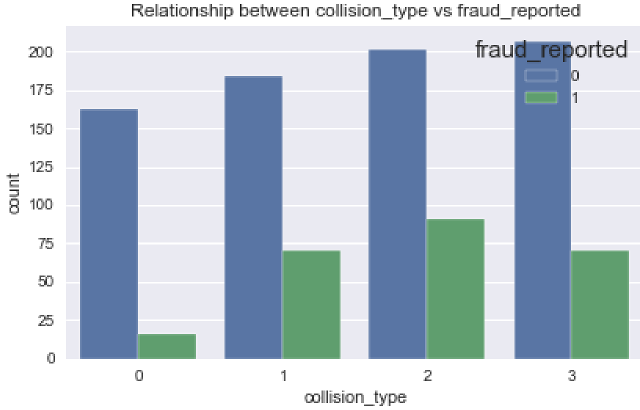
Relationship between incident_type vs fraud_reported



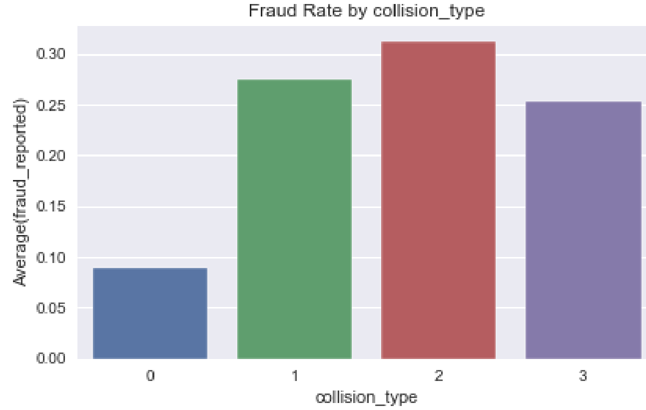
Fraud Rate by incident_type



Relationship between collision_type vs fraud_reported

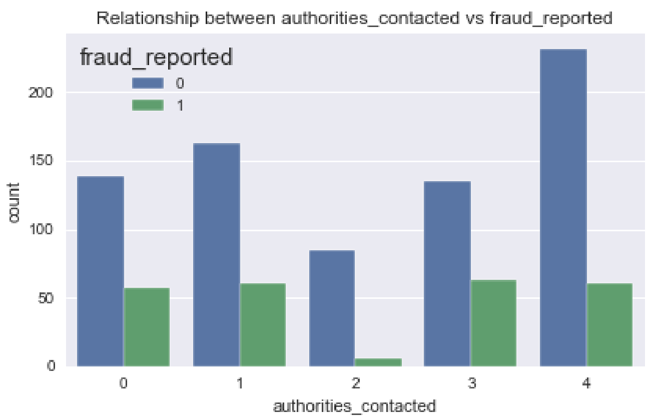


Fraud Rate by collision_type

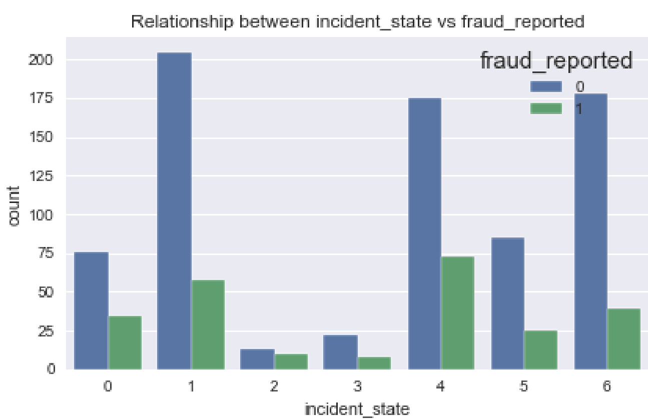



Relationship between incident_severity vs fraud_reported
fraud_reported
0
1
count
incident_severity


Fraud Rate by incident_severity
Average(fraud_reported)
incident_severity


Relationship between authorities_contacted vs fraud_reported
fraud_reported
0
1
count
authorities_contacted

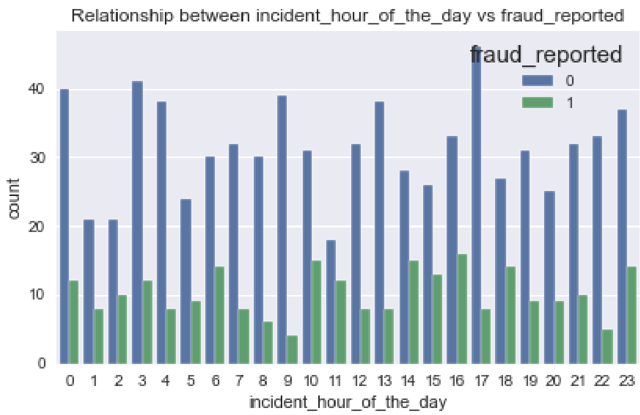

Fraud Rate by authorities_contacted
Average(fraud_reported)
authorities_contacted

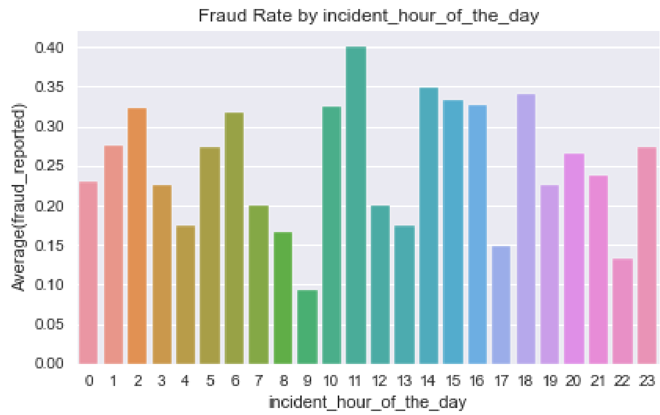

Relationship between incident_state vs fraud_reported
fraud_reported
0
1
count
incident_state

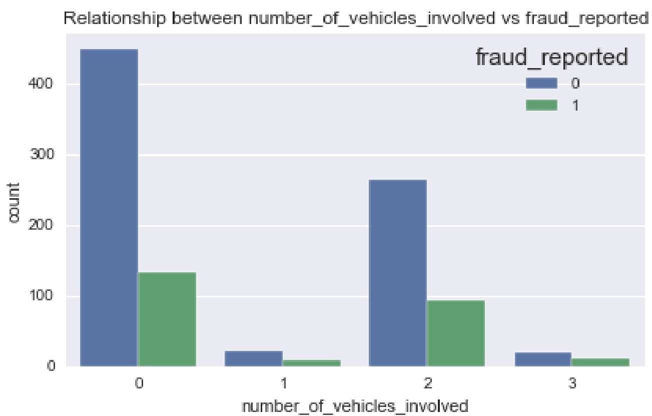

Fraud Rate by incident_state
Average(fraud_reported)
incident_state

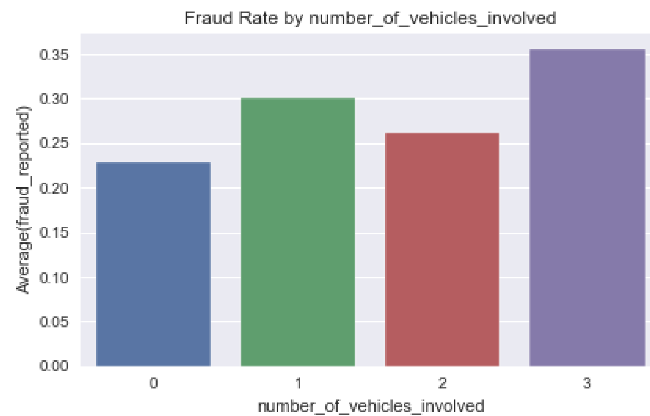

Relationship between incident_city vs fraud_reported
fraud_reported
0
1
count
incident_city


Fraud Rate by incident_city
Average(fraud_reported)
incident_city


Relationship between incident_hour_of_the_day vs fraud_reported
fraud_reported
0
1
count
incident_hour_of_the_day


Fraud Rate by incident_hour_of_the_day
Average(fraud_reported)
incident_hour_of_the_day

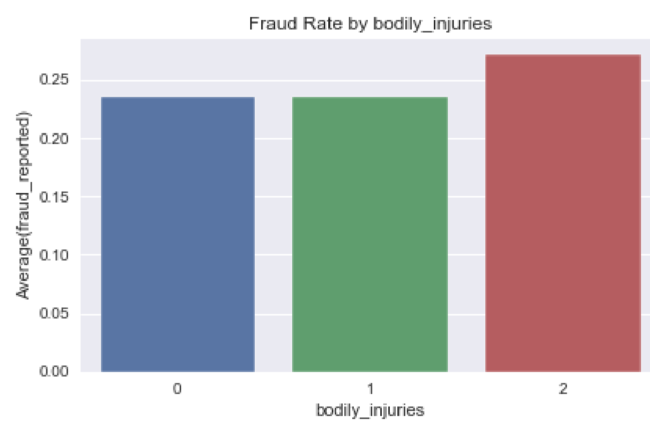

Relationship between number_of_vehicles_involved vs fraud_reported
fraud_reported
0
1
count
number_of_vehicles_involved


Fraud Rate by number_of_vehicles_involved
Average(fraud_reported)
number_of_vehicles_involved


Relationship between property_damage vs fraud_reported
fraud_reported
0
1
count
property_damage


Fraud Rate by property_damage
Average(fraud_reported)
property_damage


Relationship between bodily_injuries vs fraud_reported
fraud_reported
0
1
count
bodily_injuries

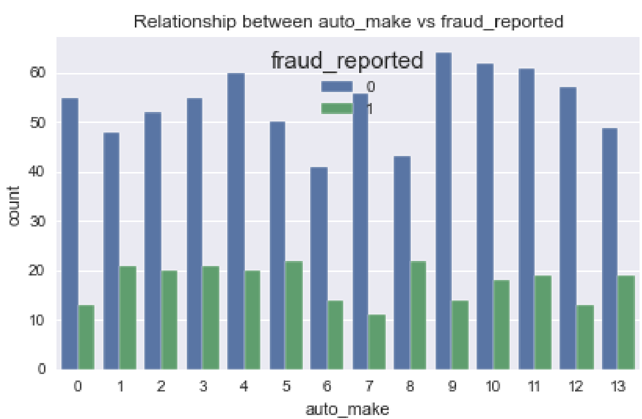

Fraud Rate by bodily_injuries
Average(fraud_reported)
bodily_injuries


Relationship between witnesses vs fraud_reported
fraud_reported
0
1
count
witnesses

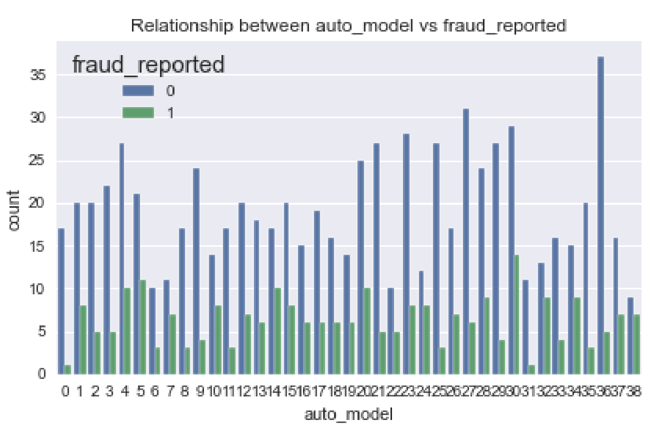

Fraud Rate by witnesses
Average(fraud_reported)
witnesses

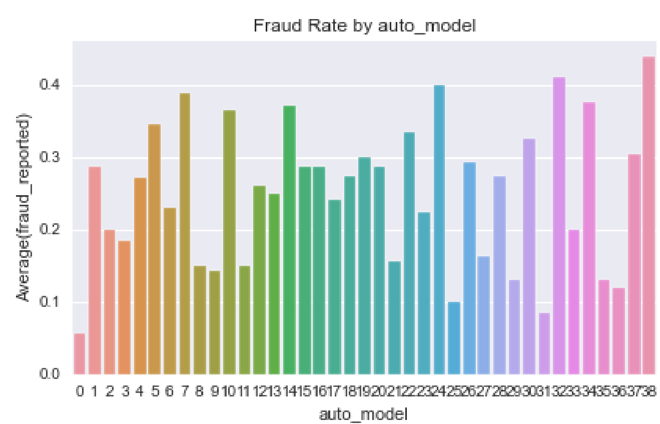

Relationship between police_report_available vs fraud_reported
fraud_reported
0
1
count
police_report_available

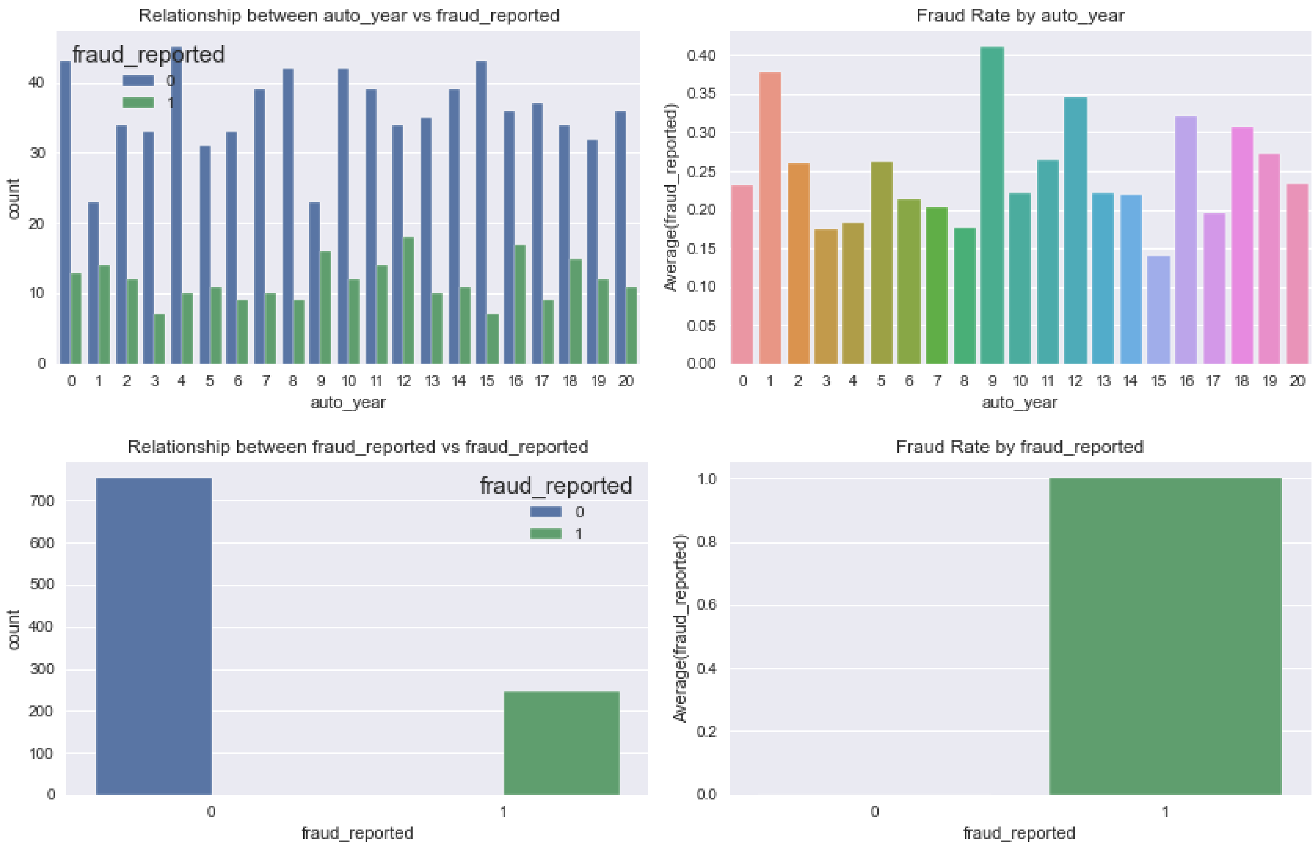

Fraud Rate by police_report_available
Average(fraud_reported)
police_report_available


Relationship between auto_make vs fraud_reported
fraud_reported
0
1
count
auto_make

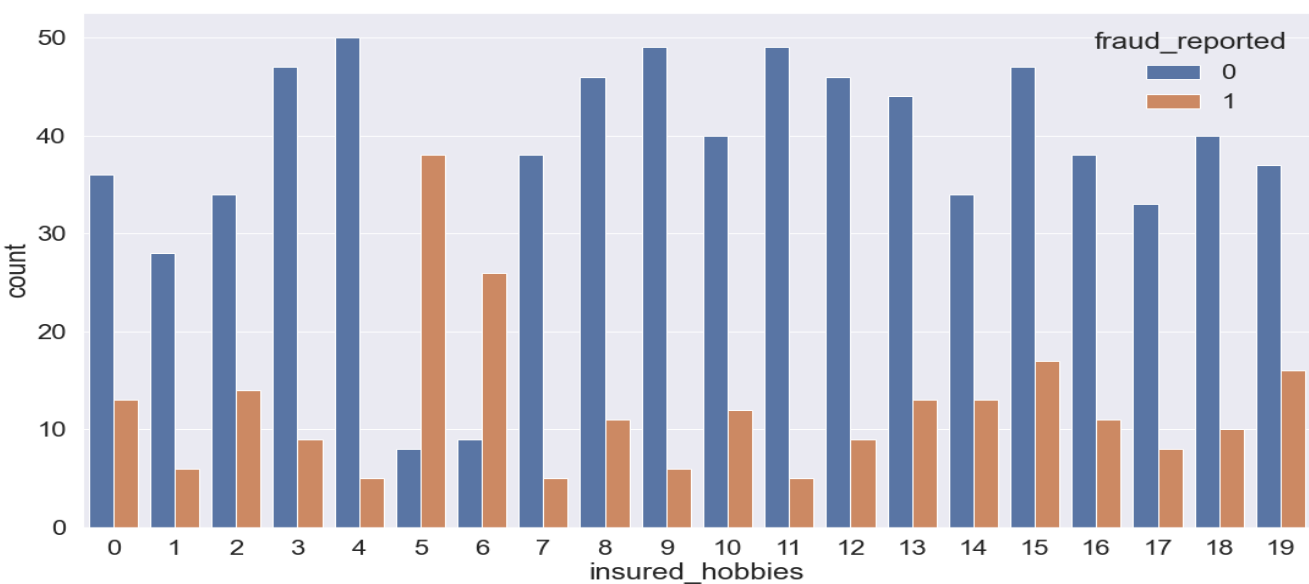

Fraud Rate by auto_make
Average(fraud_reported)
auto_make

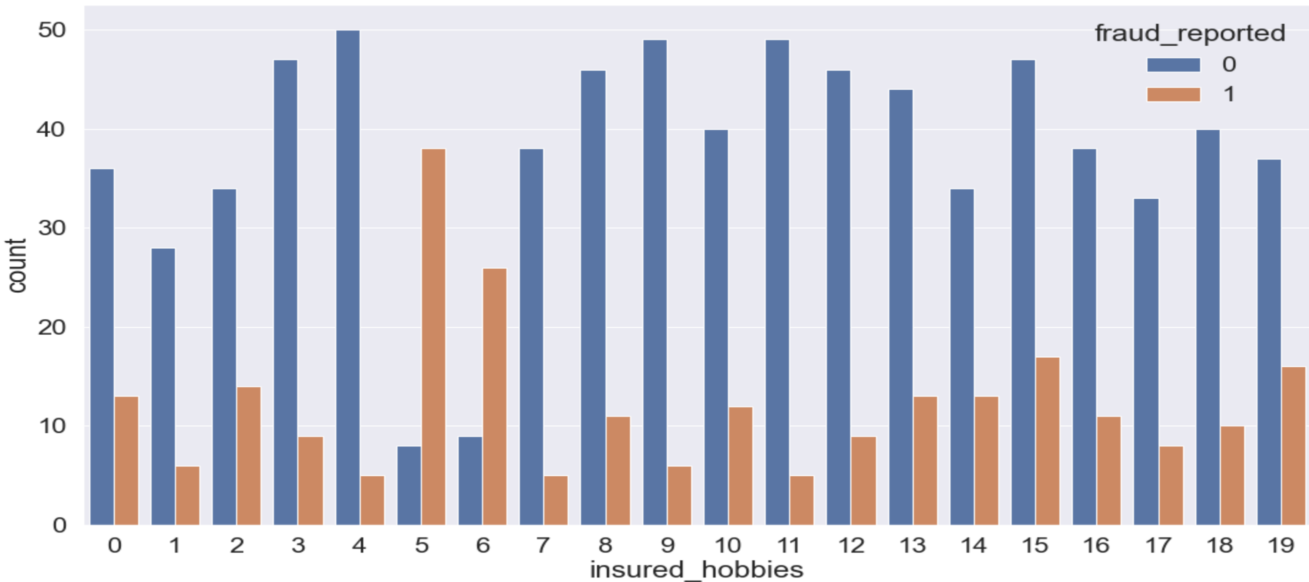

Relationship between auto_model vs fraud_reported
fraud_reported
0
1
count
auto_model

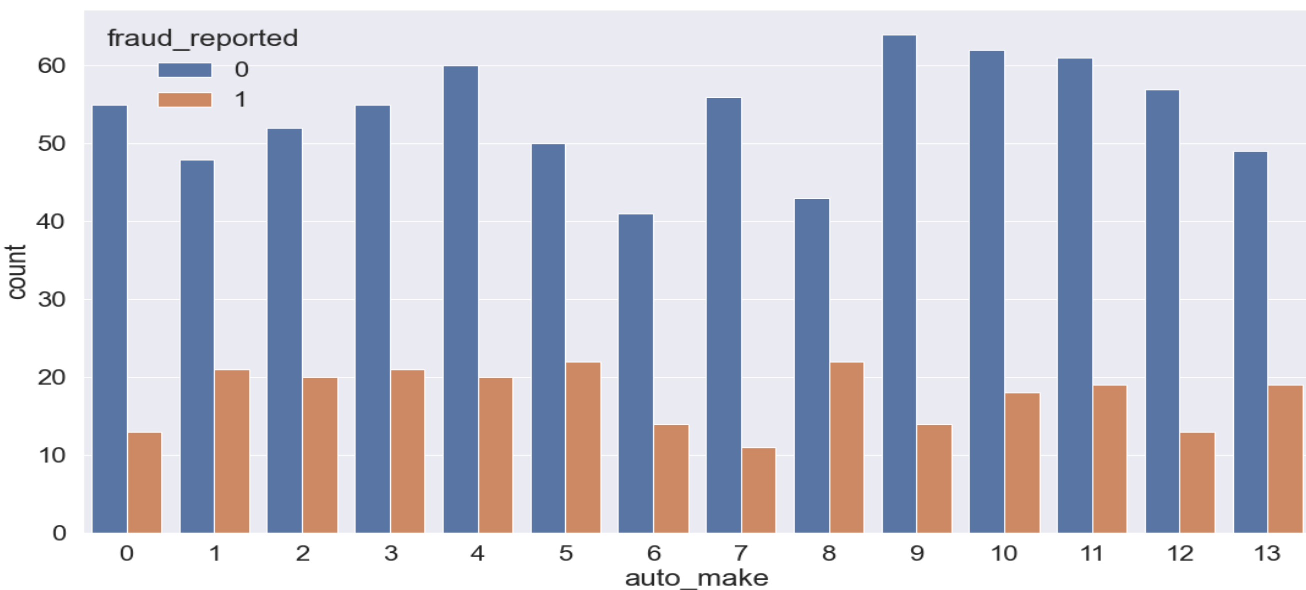

Fraud Rate by auto_model
Average(fraud_reported)
auto_model



- Next, a few more comparisons were made to see how these features affect the target Label.







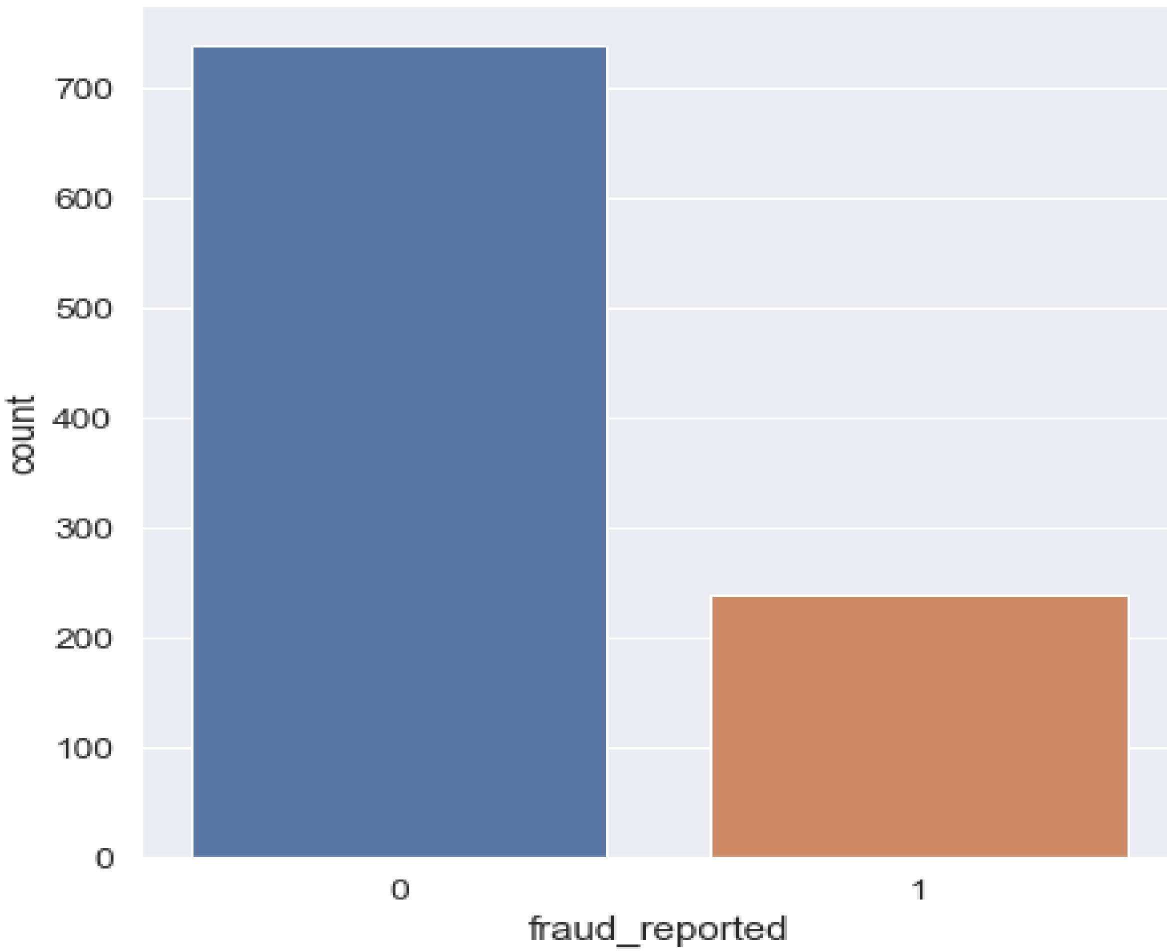
It is found that these features aren't exactly related to the target label.

- Next we can look at the Correlation Plot: (Displayed on the next page)
    1. We can see that total_claim_amount','injury_claim', 'property_claim', 'vehicle_claim' are correlated not only with the target, but also with each other.
    2. Same with 'incident_type', 'collision_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'incident_location', 'incident_hour_of_the_day','number_of_vehicles_involved. they are correlated with each other
    3. The 'collision_type', 'incident_severity' is correlated with the target.
    4. Rest of the features have low correlation.

5. These findings were also suggestive in the scatterplot. which means we can consider dropping the other numerical data columns.

In the above correlation heatmap, we can see all the features that have a relationship with the target label. In case the correlation coefficient is found negative, it means that there is a negative correlation between the feature and the Target Label.

- Next, we check to see if the Dataset has a balanced set of records with regard to the Target Label. (Attaching the plot below)
  We checked to see if SMOTE would remove the Imbalances, which it did, via Oversampling. We then used this technique while building model.



End of EDA.

Note: During the process of plotting the categorical values, we had to also encoded the attributes of each categorical data column. This was done using Label Encoding.

IV Pre-processing Pipeline

- This involved dropping of the above mentioned nominal and empty columns.
- Removing all Outliers using Zscore technique.
- After separating the Categorical data columns, we chose to encode the attributes of the columns using Label Encoder.
- Used Standard Scaler to scale the numerical data columns.
- Applied SMOTE to remove imbalances in the dataset with regard to the Label.
- Split the Data into Training and Testing datsets.
- Used KBest Feature selection technique, to choose the best features to predict the target Label

  Below are attached the codes for the same.

In [29]:
```
#The above plot shows outliers, lets be sure to find the same manually
#finding Outliers using z-score technique. if z>3 it is supposed to be an outlier
def z_outlier(df):
    df_z=df
    z=np.abs(zscore(df_z))
    a=int(df_z.size)
    df_z = df_z[(z<3).all(axis=1)]
    print('Percent of data retained = '+ str(int(df_z.size)/a*100),'%')
    return(df_z)
```

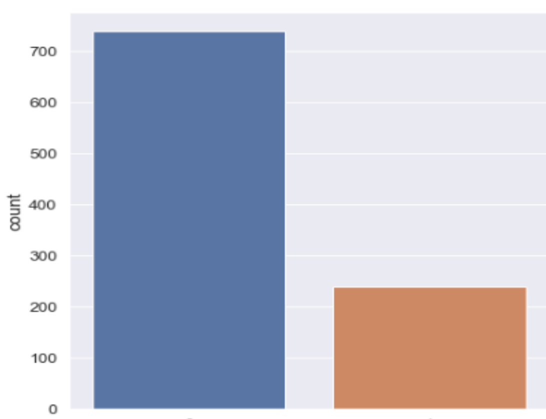
In [30]:
```
df=z_outlier(data)
```

```
Percent of data retained = 98.0 %
```

Checking of Dataset is balanced towards the target

In [31]:
```
f, ax = plt.subplots(figsize=(7, 7))
sns.countplot(x='fraud_reported',data=df)
```

Out[31]: <AxesSubplot:xlabel='fraud_reported', ylabel='count'>



In [147]:

```python
#lets do Lable enconding coding to make more features

LE = LabelEncoder()
LE_count = 0

# Iterate through the columns
for col in data[num_cat(data)[1]]:

    data[col] = LE.fit_transform(data[col])



    LE_count += 1

print('%d Catagorical Columns were label encoded.' % LE_count)
```

```
25 Catagorical Columns were label encoded.
```

Let us next scale the numerical data columns

In [34]:

```python
scalar=StandardScaler()
scaled_features=['months_as_customer','age','policy_annual_premium','insured_zip','capital-gains','capital-loss','total_clai
df[scaled_features] = scalar.fit_transform(df[scaled_features])
```

In [35]:

```python
df.columns
```

Out[35]:

```
Index(['months_as_customer', 'age', 'policy_state', 'policy_csl',
       'policy_deductable', 'policy_annual_premium', 'umbrella_limit',
       'insured_zip', 'insured_sex', 'insured_education_level',
       'insured_hobbies', 'insured_relationship', 'capital-gains',
       'capital-loss', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', 'incident_state', 'incident_city',
       'incident_hour_of_the_day', 'number_of_vehicles_involved',
       'property_damage', 'bodily_injuries', 'witnesses',
       'police_report_available', 'total_claim_amount', 'injury_claim',
       'property_claim', 'vehicle_claim', 'auto_make', 'auto_model',
       'auto_year', 'fraud_reported'],
      dtype='object')
```

In [64]:

```python
#Splitting Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (784, 20)
Number transactions y_train dataset:  (784,)
Number transactions X_test dataset:  (196, 20)
Number transactions y_test dataset:  (196,)
```


In [64]:

```python
#Splitting Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (784, 20)
Number transactions y_train dataset:  (784,)
Number transactions X_test dataset:  (196, 20)
Number transactions y_test dataset:  (196,)
```


In [65]:

```python
#Checking if SMOTE is capable removing the Imbalance
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))



SM = SMOTE(random_state = 2)
X_train_res, y_train_res = SM.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

```
Before OverSampling, counts of label '1': 191
Before OverSampling, counts of label '0': 593

After OverSampling, the shape of train_X: (1186, 20)
After OverSampling, the shape of train_y: (1186,)

After OverSampling, counts of label '1': 593
After OverSampling, counts of label '0': 593
```

In [67]:

```python
#Using Kbest for reference
bestk = SelectKBest(score_func=f_classif,k=20)
fit=bestk.fit(X_train_res,y_train_res)
df_scores=pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(X_train_res.columns)

feature_scores=pd.concat([df_columns,df_scores],axis=1)
feature_scores.columns=['Feature_Name','Score']
print(feature_scores.nlargest(20,'Score'))
```

```
               Feature_Name       Score
7         incident_severity  520.197202
18            vehicle_claim   69.486413
15       total_claim_amount   65.734306
14  police_report_available   48.721248
17           property_claim   48.668133
11          property_damage   39.078613
6             incident_type   34.143433
8      authorities_contacted   30.792979
2          policy_deductable   27.078802
3               insured_sex   22.389780
16             injury_claim   17.164890
9            incident_state   15.367533
10            incident_city   15.002751
1              policy_state   13.263134
4   insured_education_level   10.599387
12          bodily_injuries    9.146663
5           insured_hobbies    8.362187
19                auto_make    6.661657
13                witnesses    1.569473
0        months_as_customer    0.354347
```

- Based on certain domain Knowledge and the EDA and the above KBest feature prediction, we have picked the best 21 features, and additionally removed 'policy_csl' as it had a very close entropy value with 'police_report_available'

Based on the EDA, the KBest Feature Selection technique, and Domain knowledge, we removed 25 features, based on the entropy values.

- First we removed-
  'insured_zip','auto_model','incident_hour_of_the_day','age','policy_annual_premium','umbrella_limit','capital-loss','collision_type','capital-gains','insured_relationship','auto_year','number_of_vehicles_involved','policy_csl'
  on account of having the least entropy values.
- Using Domain Knowledge assisted by the common entropy values, or entropy values being too close, we decided to remove-
  'months_as_customer','witnesses','auto_make','incident_city','property_claim'

Thereby having a total of 15 features to predict the target label. They are:

'policy_state', 'policy_deductable', 'insured_sex', 'insured_education_level', 'insured_hobbies', 'incident_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'property_damage', 'bodily_injuries', 'police_report_available', 'total_claim_amount', 'injury_claim', 'vehicle_claim'

V Building Machine Learning Models

Using a User defined function, we automate a Model selection function that balances the data at every iteration using SMOTE Technique, and predicts the best model on the basis of accuracy score and the best random state. It also captures the incidents' F1score.

(Find the code below)

In [75]:
```python
#Function to choose best classification model, its metrics, and random state.
#including the SMOTE balancing into this loop
mod=[LogisticRegression(),KNeighborsClassifier(),DecisionTreeClassifier(),RandomForestClassifier()]
max_acc_score=0
max_f1score=0
for r_state in range(0,100):
    train_x,test_x,train_y,test_y=train_test_split(X,y,random_state=r_state,test_size=0.2)
    SM = SMOTE(random_state = r_state)
    X_train_res, y_train_res = SM.fit_resample(train_x, train_y.ravel())
    for i in mod:

        i.fit(X_train_res,y_train_res)
        pred_y = i.predict(test_x)
        acc_score=accuracy_score(test_y,pred_y)
        f1_Score=f1_score(test_y,pred_y)



        print(i,"Max Accuracy score for random state  ",r_state,"is",acc_score, "with f1 Score ", f1_Score)
        if acc_score>max_acc_score:
            max_acc_score=acc_score
            max_f1score=f1_Score
            final_state= r_state
            final_model = i


print("\n\n\n\n\n")
print("Max Acc_score for random state ",final_state,"is",max_acc_score,"and best model is ",final_model, "with f1score as ",
```

```
KNeighborsClassifier() Max Accuracy score for random state   97 is 0.6428571428571429 with f1 Score  0.4067796610169492
DecisionTreeClassifier() Max Accuracy score for random state   97 is 0.7244897959183674 with f1 Score  0.44897959183673475
RandomForestClassifier() Max Accuracy score for random state   97 is 0.8010204081632653 with f1 Score  0.597938144329897
LogisticRegression() Max Accuracy score for random state   98 is 0.7040816326530612 with f1 Score  0.546875
KNeighborsClassifier() Max Accuracy score for random state   98 is 0.6479591836734694 with f1 Score  0.4732824427480916
DecisionTreeClassifier() Max Accuracy score for random state   98 is 0.8163265306122449 with f1 Score  0.660377358490566
RandomForestClassifier() Max Accuracy score for random state   98 is 0.7959183673469388 with f1 Score  0.5918367346938775
LogisticRegression() Max Accuracy score for random state   99 is 0.7244897959183674 with f1 Score  0.5909090909090908
KNeighborsClassifier() Max Accuracy score for random state   99 is 0.6581632653061225 with f1 Score  0.510948905109489
DecisionTreeClassifier() Max Accuracy score for random state   99 is 0.7857142857142857 with f1 Score  0.6379310344827586
RandomForestClassifier() Max Accuracy score for random state   99 is 0.7857142857142857 with f1 Score  0.6440677966101694





Max Acc_score for random state  39 is 0.8622448979591837 and best model is  RandomForestClassifier() with f1score as  0.68965
51724137931
```

Based on the above prediction, we find that the best random_state at 39, and Random Forest classifier giving us the best result of Accuracy score of 0.8622448979591837 and f1score of 0.6896551724137931.

Next: We build the model predicted above using the random state parameter.

The screenshot also contains a comparison between the actual and predicted Y samples.

Initialising the Best model with the predicted best random state.

In [76]:
```
train_x,test_x,train_y,test_y=train_test_split(X,y,random_state=39,test_size=0.2)
SM = SMOTE(random_state = 39)
X_train_res, y_train_res = SM.fit_resample(train_x, train_y.ravel())
RFC=RandomForestClassifier(random_state = 39)
RFC.fit(X_train_res,y_train_res)


```

Out[76]:
```
RandomForestClassifier
RandomForestClassifier(random_state=39)
```

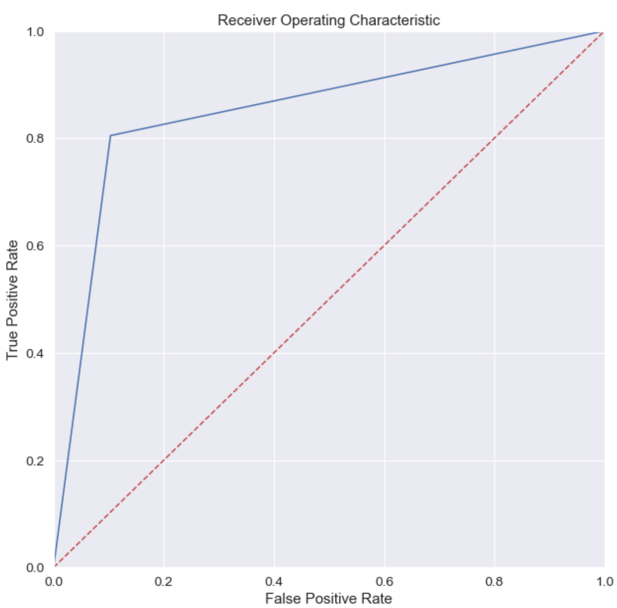
In [77]:
```
#Metrics
pred_y = RFC.predict(test_x)
acc_score=accuracy_score(test_y,pred_y)
```

In [94]:
```
#Comparing Actual and Predicted Values.
compare=pd.DataFrame({'Actual':test_y,'Predicted':pred_y})
compare.sample(10)
```

Out[94]:

	Actual	Predicted
564	0	0
709	1	1
153	0	0
773	0	0
399	0	0
112	0	0
135	1	1
129	1	1
525	0	0
917	0	1

Below is the ROC plot of the above model.



VI Hyper parameter Tuning:

We used both GridsearchCV and Randomised SearchCV for the hyperparameter tuning, but found RandomisedSearchCV to be more productive. Gridsearch CV's tuning wasn't as effective. (Codes attached below)

1. Grid Search CV

In [114]:
```
param_grid = {  'bootstrap': [True], 'max_depth': [5, 10, None], 'max_features': ['auto', 'log2'], 'n_estimators': [5, 6, 7,
g_search=GridSearchCV(RFC,param_grid = param_grid, cv = 3, n_jobs = 1, verbose = 0, return_train_score=True)
g_search.fit(X_train_res,y_train_res)
n_neighbor=g_search.best_params_
n_neighbor
```

Out[114]:
```
{'bootstrap': True,
 'max_depth': None,
 'max_features': 'auto',
 'n_estimators': 13}
```

In [116]:
```
best_param_grid = {  'bootstrap': [True], 'max_depth': [None], 'max_features': ['auto'], 'n_estimators': [13]}
best_g_search=GridSearchCV(RFC,param_grid = best_param_grid, cv = 3, n_jobs = 1, verbose = 0, return_train_score=True)

best_g_search.fit(X_train_res,y_train_res)
```

Out[116]:
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [122]:
```
GSCV_pred_y = best_g_search.predict(test_x)
acc_score_GSCV=accuracy_score(test_y,GSCV_pred_y)

```

In [123]:
```
print(f'Prediction',GSCV_pred_y)
print(f'Accuracy Score',acc_score_GSCV)
```

```
Prediction [0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0
 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1
 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1
 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0
 0 0 0 0 0 1 1 1 0 1 1]
Accuracy Score 0.8520408163265306
```

We found no change with this method, the accuracy score was still 0.8520

2. RandomizedSearchCV

In [119]:
```
#initialising range of parameters for sequential testing
n_estimators = [int(x) for x in np.linspace(start=20, stop=150, num=50)]
max_depth = [int(x) for x in np.linspace(20, 150, num=25)]

param_dist = {
    'n_estimators': n_estimators,
    'max_depth': max_depth,
}
```

In [120]:
```
#Fitting the above range into previously found model
RFC_HPT = RandomForestClassifier(random_state=39)
rf_cv = RandomizedSearchCV(
    estimator=RFC_HPT, param_distributions=param_dist, cv=5, random_state=39)
```

In [121]:
```
rf_cv.fit(X_train_res,y_train_res)
```

Out[121]:
```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [100]:
```
print(f'\nBest parameters are ', rf_cv.best_params_)
```

```
Best parameters are  {'n_estimators': 101, 'max_depth': 112}
```

In [101]:
```
RFC_best = RandomForestClassifier(max_depth=101, n_estimators=112, random_state=39)
RFC_best.fit(X_train_res,y_train_res)
```

Out[101]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=101, n_estimators=112, random_state=39)
```

In [124]:
```
Y_pred_RFC_best = RFC_best.predict(test_x)
```

In [125]:
```
RSCV_pred_y = RFC_best.predict(test_x)
acc_score_GSCV=accuracy_score(test_y,RSCV_pred_y)
```

In [126]:
```
print('Random Forest Classifier:')
print('Accuracy score:', round(accuracy_score(test_y, RSCV_pred_y) * 100, 2))
print('F1 score:', round(f1_score(test_y, RSCV_pred_y) * 100, 2))
```

```
Random Forest Classifier:
Accuracy score: 87.76
F1 score: 72.73
```

Using Randomised SearchCV found a better accuracy, of about 87.76%, which is about 1.5% more than the accuracy of the Random Forest Classifier without HPT.

Hence, we used this technique, and evaluated it using Area under ROC and classification report. (attached under the next heading)

VII Metrics:

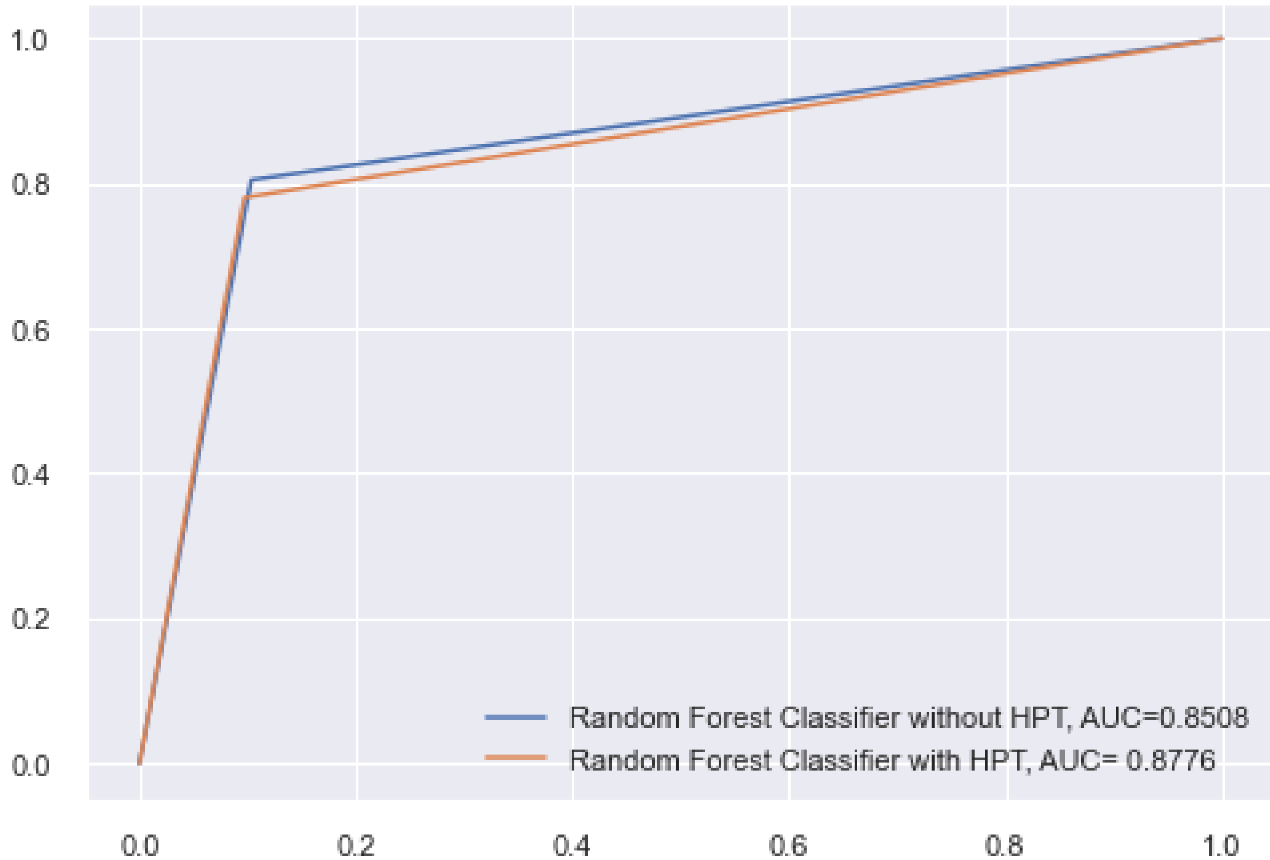
It is clear that we have fine tuned the Random Forest Classifier model to compensate for overfitting and yet give a better accuracy rate. Let us now look at other metrics.

Confusion Matrix:

We find that, we have a very good true positive number, we also have a fairly small false negatives which really complements this model.

Let us look at another metric, of Area under ROC.

```
In [160]:
#set up plotting area
plt.figure(0).clf()

#fit ROC before

y_predd = RFC.predict_proba(test_x)[:, 1]
fpr1, tpr1, _ = roc_curve(test_y, pred_y)
auc1 = round(roc_auc_score(test_y, pred_y), 4)
plt.plot(fpr1,tpr1,label="Random Forest Classifier without HPT, AUC="+str(auc1))

#fit gradient boosted model and plot ROC curve

y_predd2 = RFC_best.predict_proba(test_x)[:, 1]
fpr2, tpr2, _ = roc_curve(test_y, RSCV_pred_y)
auc2 = round(roc_auc_score(test_y, RSCV_pred_y), 4)
plt.plot(fpr2,tpr2,label="Random Forest Classifier with HPT, AUC= "+str(auc2))

#add legend
plt.legend()
```



Here we see the Area under the curve as 0.8776 which is another metric to say we have the same 87.76% accuracy.

Finally let us see what the Classification Report says:

In [136]:
```
#Finally lets look at the classification report

print(classification_report(test_y, RSCV_pred_y))
```

```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92       155
           1       0.68      0.78      0.73        41

    accuracy                           0.88       196
   macro avg       0.81      0.84      0.82       196
weighted avg       0.89      0.88      0.88       196
```

A Comparison of the the Final Predicted values after Hyper Parameter Tuning and the actual test values were compared.
Two iterations were sampled, just for our understanding of our model and its predictions.

We see an increase in accuracy with the Randomised seacrch CV HPT, it has increased to 87.76% from 86.22%

In [132]:
```
compare2=pd.DataFrame({'Actual':test_y,'Predicted':Y_pred_RFC_best})
#Random Samples-1
compare2.sample(10)
```

Out[132]:

	Actual	Predicted
970	0	0
548	0	0
161	0	0
6	0	0
879	0	0
793	0	0
523	0	0
266	1	1
929	1	1
441	0	0

In [107]:
```
#Random Samples-2
compare2.sample(10)
```

Out[107]:

	Actual	Predicted
789	0	0
716	0	0
620	1	1
574	1	1
107	0	0
13	0	0
418	0	0
462	1	1
879	0	0
202	0	0

VIII Summary:

- We have finally achieved success in the prediction of an insurance fraud, where we have an accuracy of about 84%
- The classification report is shown above for a more detail analysis.
- From the ROC curve we have more or less achieved a right angle L, which further strengthens our finding.
- Between the Y/N in the target column, the model has better chances of predicting a "N", than calling it a fraud.
- Basically this model can basically say that 94% of the time its prediction of a "N" being a negative for fraudulent claim. Which is a good thing.

End of Project

IX Acknowledgements

My sincere and heartfelt gratitude to the Institute-Datatrained, for giving me this learning experience and guidance thus far. My gratitude to the many authors and their content on the web, which catapult my understanding and enhanced my doubt and error resolution capacity.

Thank you.