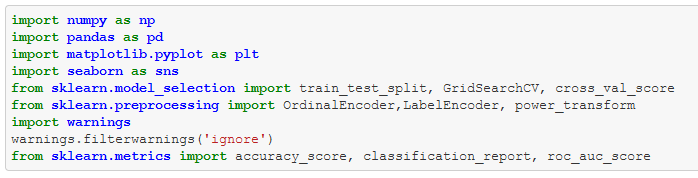
Insurance Fraud Prediction

In this blog post, I will be going through the entire process of building a machine learning model on the fraud claims in motor vehicle insurance.

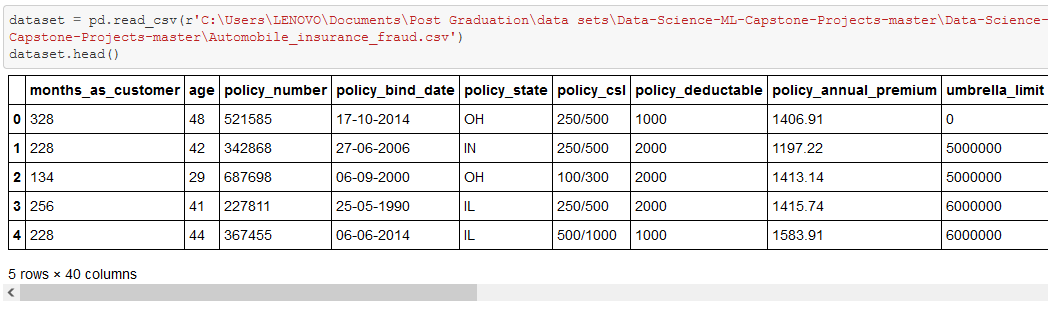
Insurance frauds cover the range of improper activities which an individual may commit in order to achieve a favourable outcome from the insurance company. This could range from staging the incident, misrepresenting the situation including the relevant actors and the cause of incident and finally the extent of damage caused.

There are lot of factors in insurance claims, which makes it next to impossible for a human eye to identify whether a claim is fraud or not. Here comes the predictive analysis, where we use Machine Learning algorithm to identify whether a claim is fraud or not.

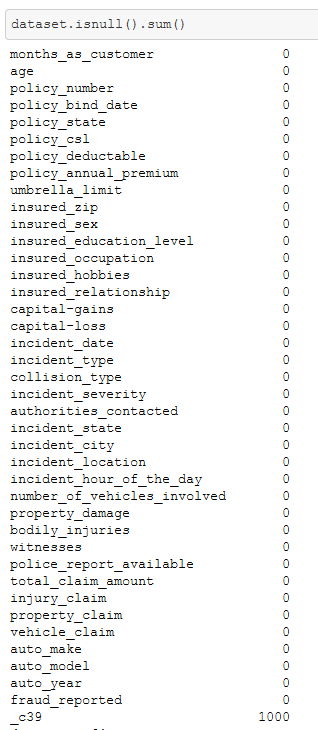
Let’s proceed with importing the necessary libraries for the problem



Import the dataset and looking at the glimpse of it. Here the dataset has 1000 rows and 40 columns.



Before proceeding further, I’m checking for the null values present in the dataset

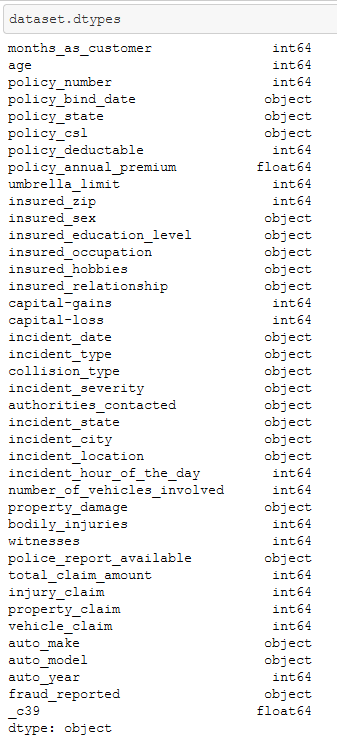


Here I can see that the feature **\_c39** is completely empty and there are no null values in the rest of the dataset. Therefore I’m removing the **\_c39** feature from the dataset.

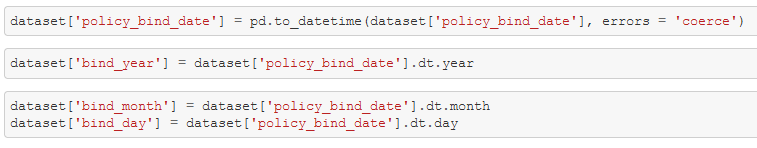
There is a feature named **policy\_number** in the dataset which is unique for every row and will not be helpful in the fraud detection. Therefore removing the same.



Checking for the data types in the dataset.

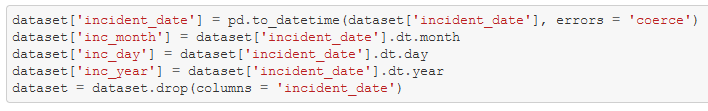


I can see that there are 21 object type feature including the target variable and there are 2 date type variables, i.e., **policy\_bind\_date** and **incident\_date.** We can’t build a model with the date type variable, therefore extracting features from the same.



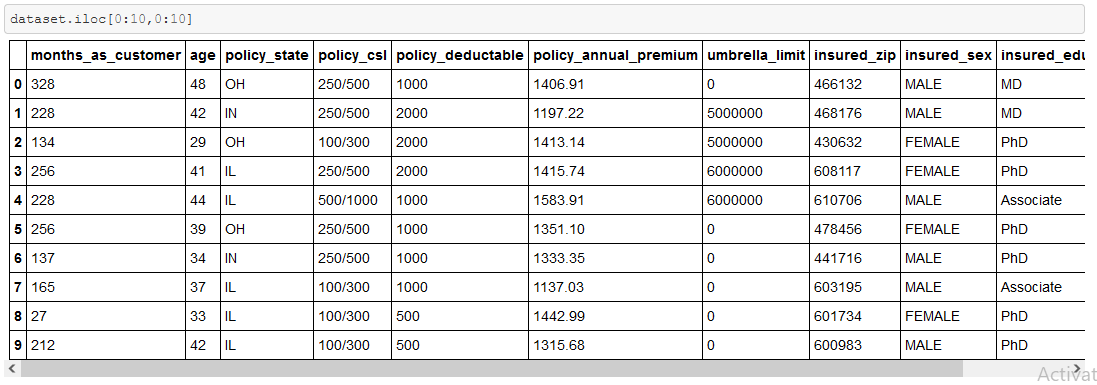
Note: we have to convert the variable to date time format before we can extract features.

Likewise, I’m extracting the features from the incident date variable.

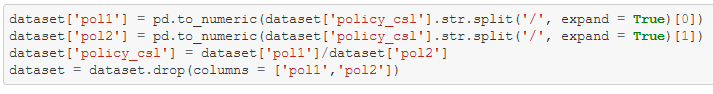


Now that I have extracted the features from the date variables, I can drop the actual date **policy\_bind\_date** and **incident\_date** using drop function.

There is another variable which needs to be converted to numerical, i.e. **policy\_csl**



I’m correcting the same using the below code.

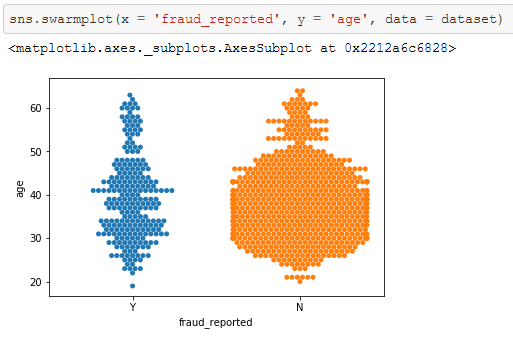
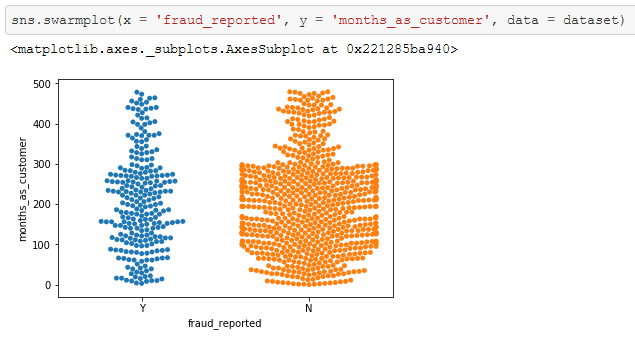


Further, we can drop the variable **incident\_location**, which is unique for each and every row.

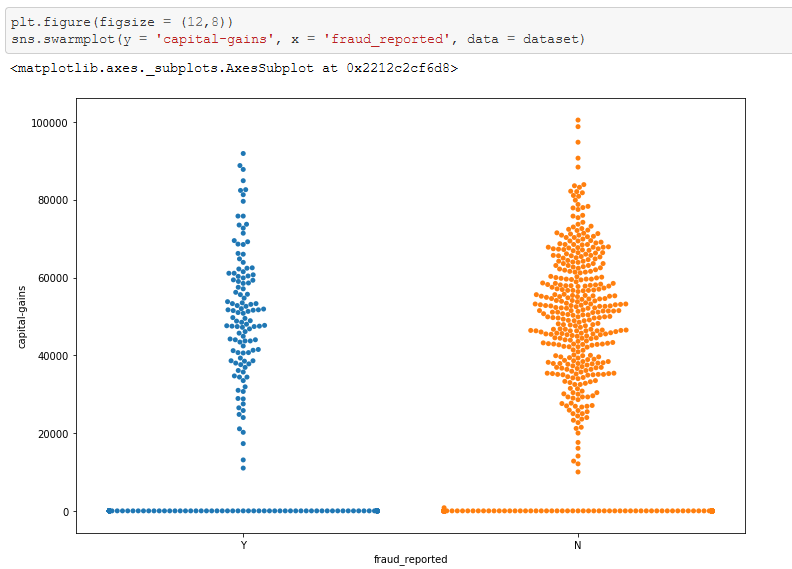
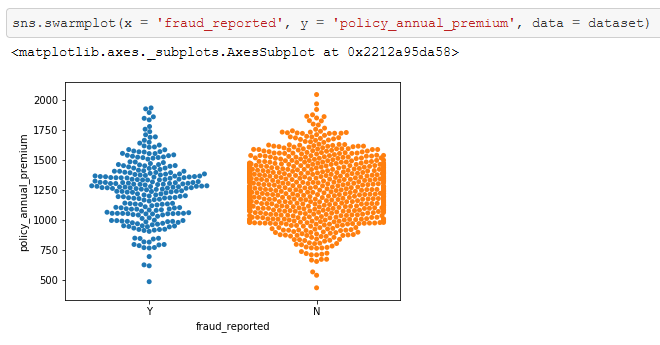


Now that I have corrected the numerical variables which were considered as ‘object’ using feature extraction and math. I can proceed with identifying the relation between the variables.

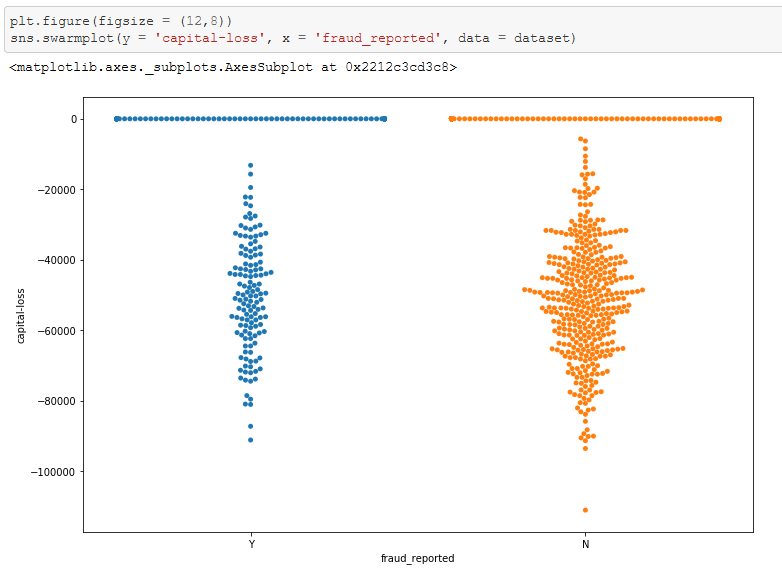
Using swarmplot to visualize the relationship between a categorical and continuous variable.

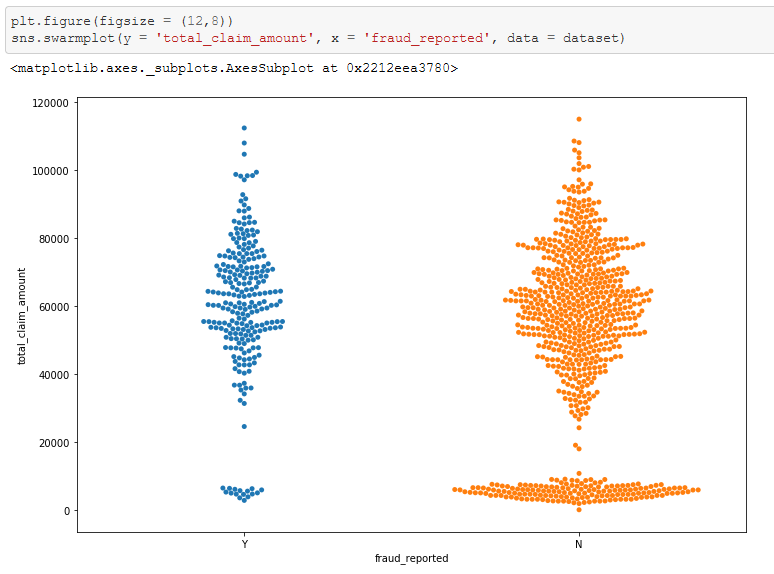


From the above plots we can say that there is a lesser fraud reported after 300 months as customer and relatively higher number of fraud were reported for the customers from the age 25 to 50 years

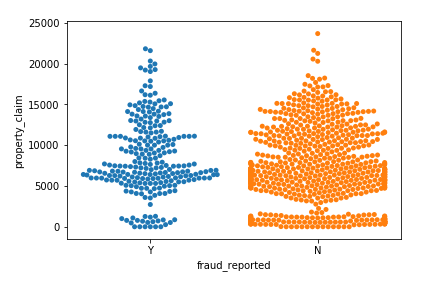


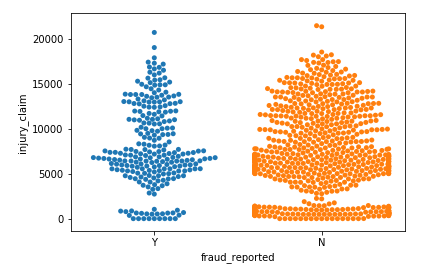
From the range 50000 USD to 80000 USD capital gains we can see higher number of frauded cases recorded and for the annual premium between 800 USD to 1600 USD we can see higher number of fraud case reported

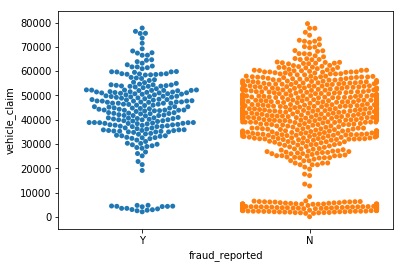




From the range 50000 USD to 80000 USD total claim amount we can see higher number of fraud cases recorded. Further with 0 capital loss and loss ranging from -25000 to -60000 has higher frauds reported

Likewise for other variables…





From the above swarm distribution, I can see that the highest fraud recorded was for the claim amount of 4500 to 7500 USD Approx. for injury claims, highest fraud recorded was for the claim amount of 4000 to 11000 USD Approx. for property claims and the highest fraud recorded was for the claim amount of 30000 to 60000 USD Approx. for vehicle claims.

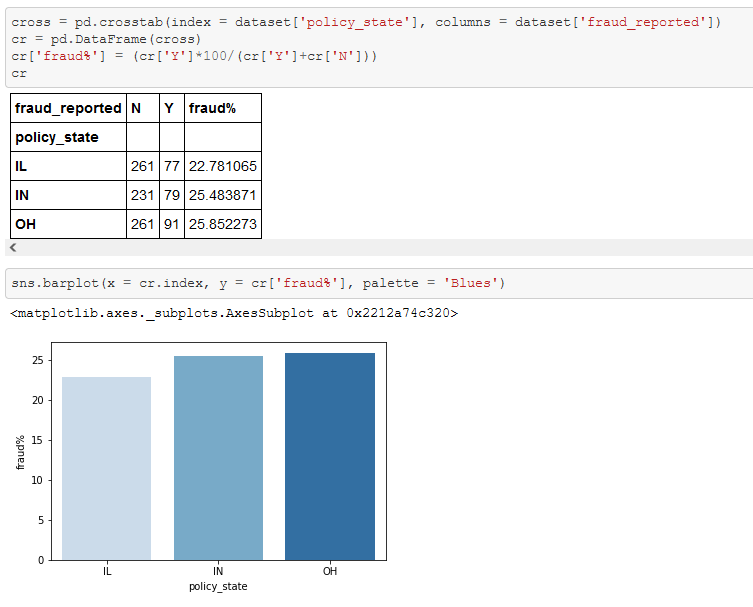
Now that we have seen the relationship with the categorical and continuous variables. We can now look at the relationship between the 2 categorical variable. Dependent (categorical) and Independents variables.

Let’s say I’m using the below code to visualize a count plot for a relationship between **policy\_state** and **fraud\_reported.**



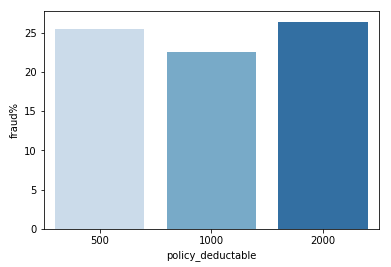
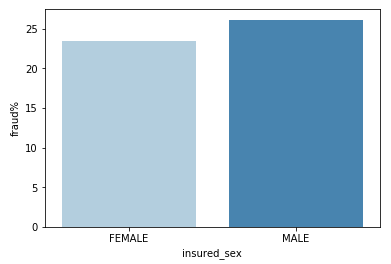
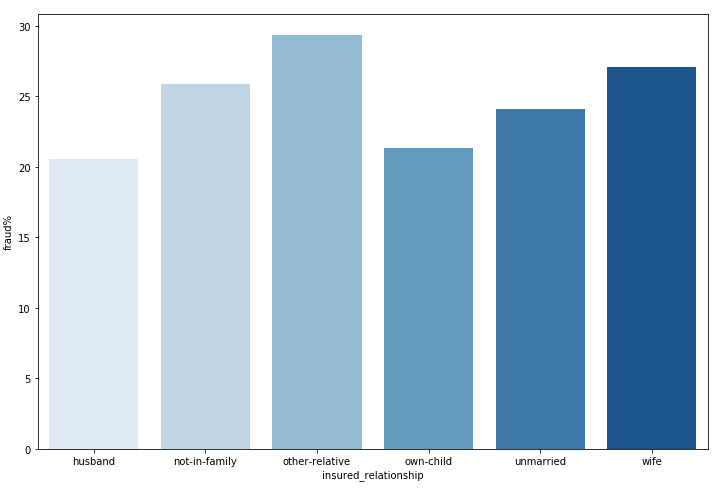
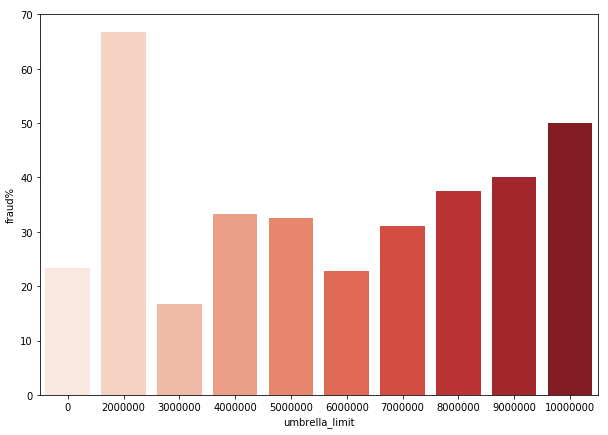
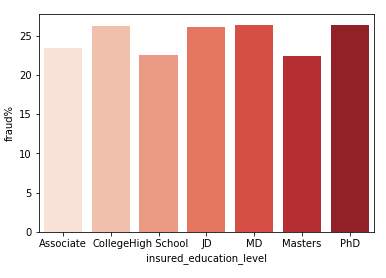
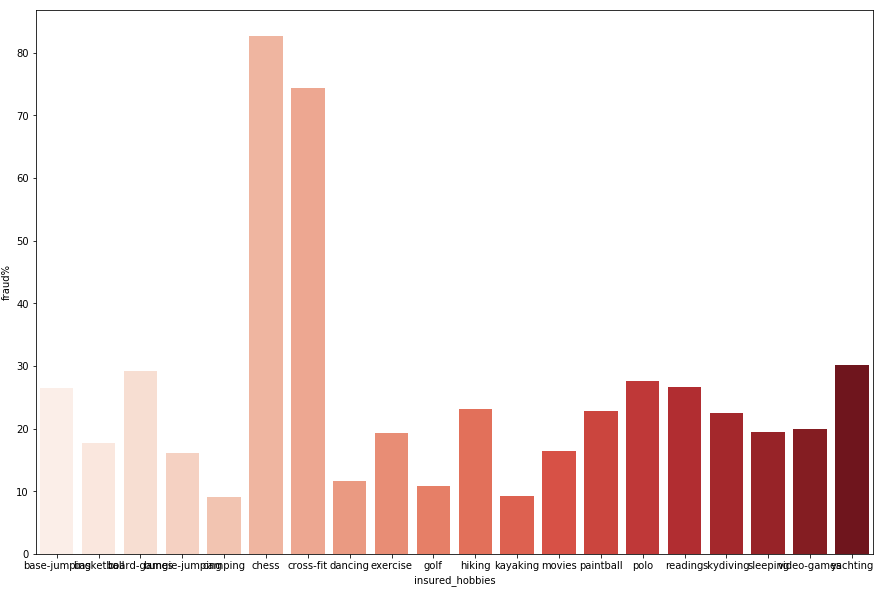
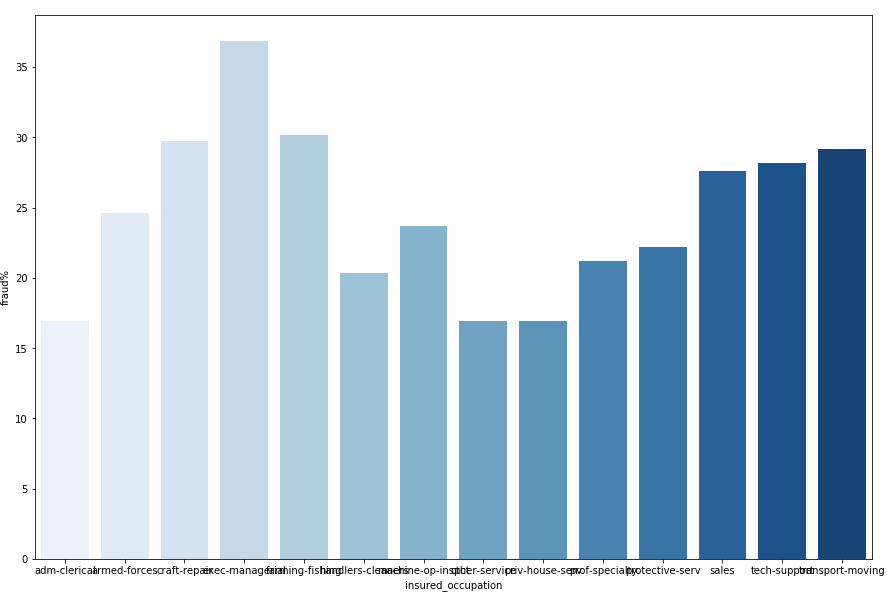
This plot will not show us the actual proportion of fraud cases in each state respectively. For example the OH (Ohio) state might have 100 non fraud case and 20 fraud case and the Il (Illinois) state might have 200 non fraud case and 30 fraud case. Looking at the numbers we cannot say that Il has higher number of fraud, here we have to use the proportion instead of count. I.e., 20/120 = 17% fraud claims for Ohio and 30/230 = 13% fraud claims for Illinois. Therefore, I can say that the state Ohio has higher fraud rates.

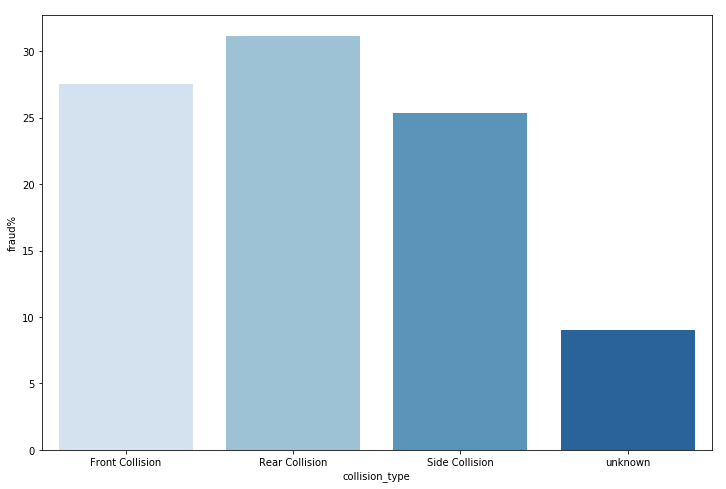
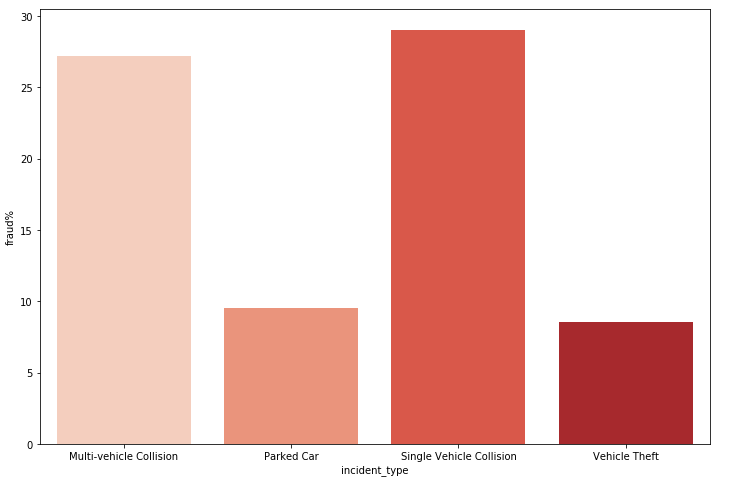
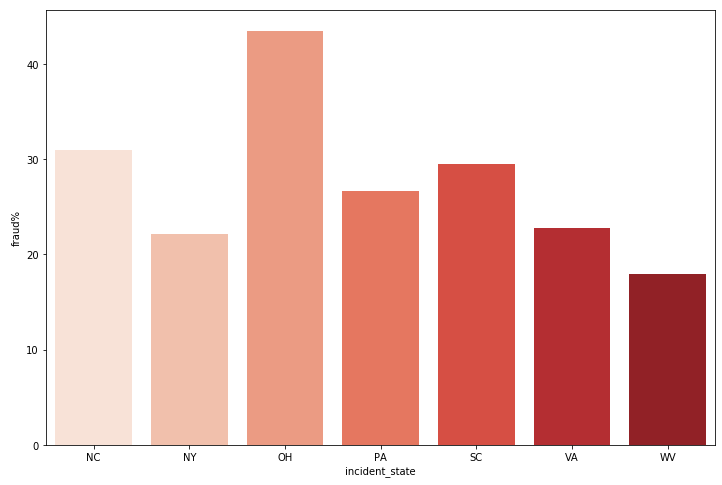
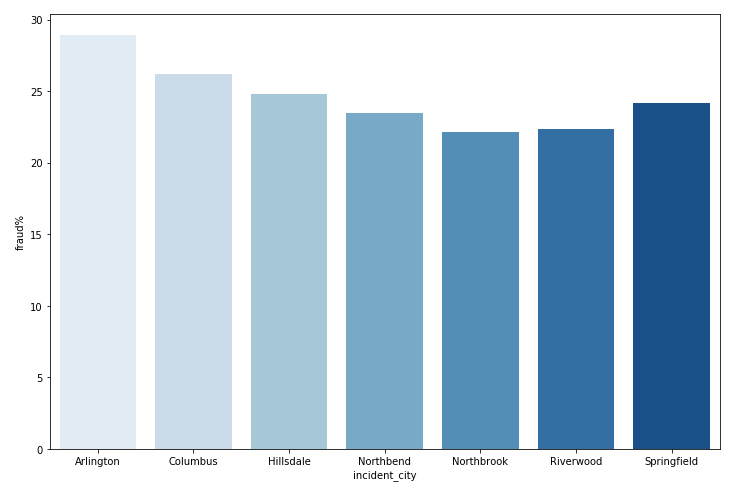
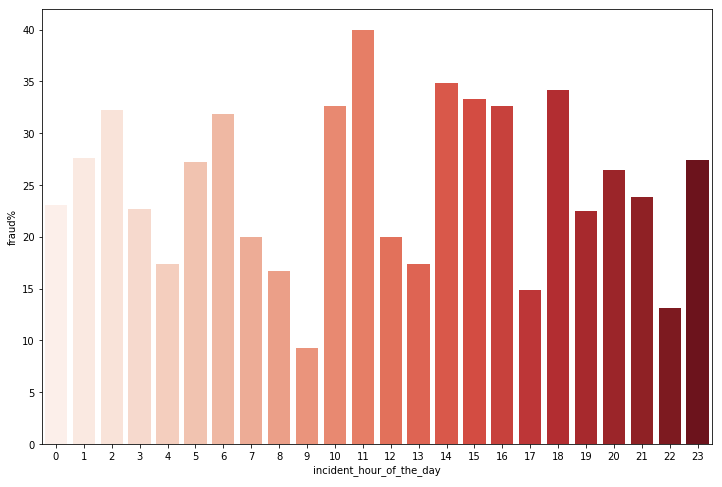
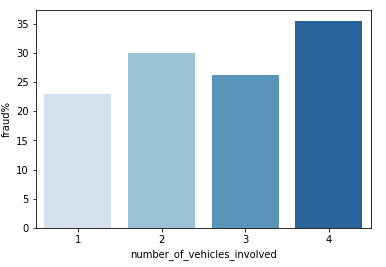
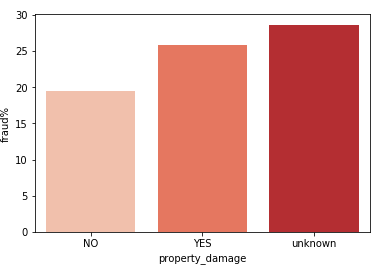
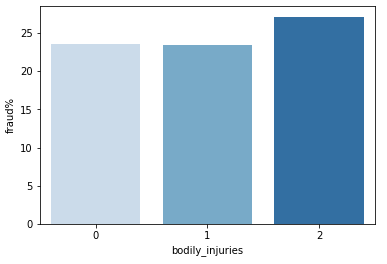
You can see the difference from the above example, using the same logic we will see the relationships for multiple variables. This can be achieved using the cross-tab function from pandas library.

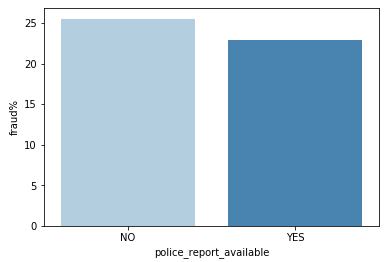
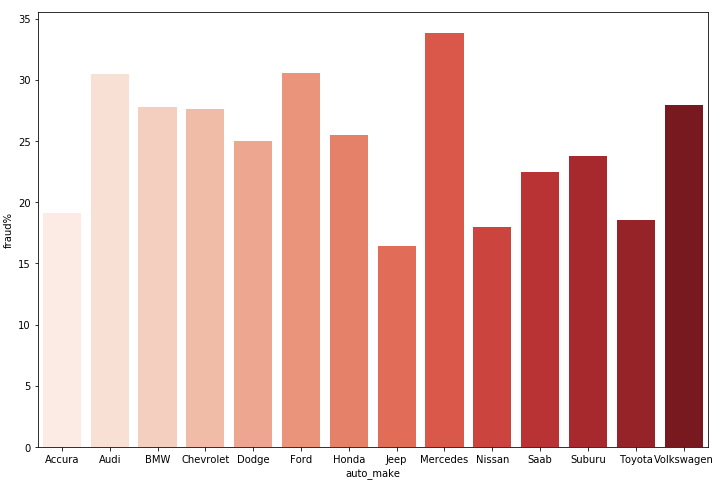
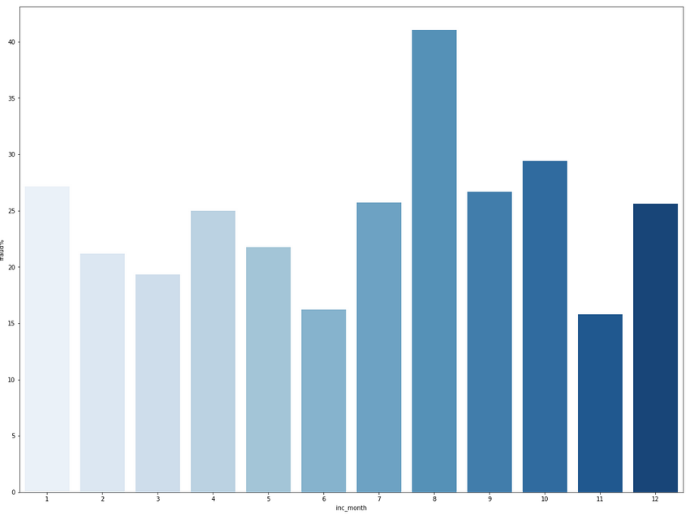
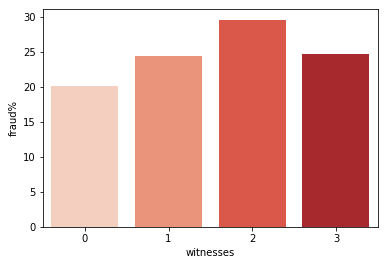


You can see from the above codes that we calculated the fraud proportion (percentage) and we visualized the same using the bar plot and we can clearly see that the state Ohio recorded higher fraud percentage, followed by Indiana and Illinois. This information were unable to interpret before using countplot.

Therefore, I’m using the same technique to visualize the relationship between categorical independent variables and the dependent variable (which is also categorical)

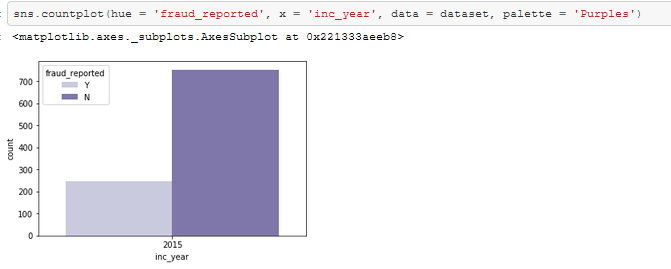
Using the same method I have plotted the below figures.



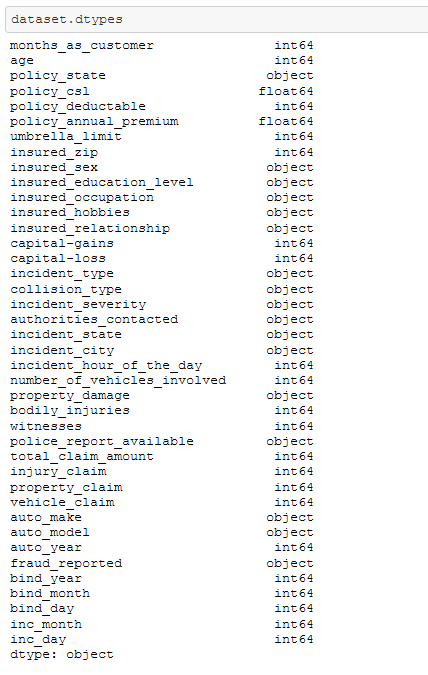
The analysis using the above plots will help us to answer any specific questions related to the dataset. Imagine if the clients asks us “when it comes to number of vehicles involved which category is more likely to be a fraud claim” and we could say “If there are 4 or more vehicles involved then it is more likely to be a fraud claim”.

Further, we can drop the **inc\_year** (we extracted from **incident\_date**) column because it is same (2015) in all rows and it won’t be helpful in ourprediction

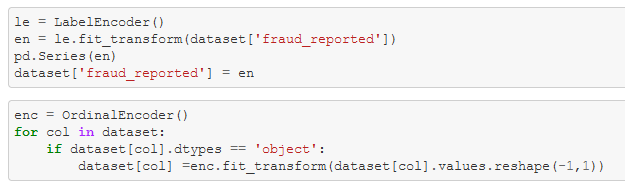


Now I’m proceeding with the data pre-processing before I can view the correlation table.

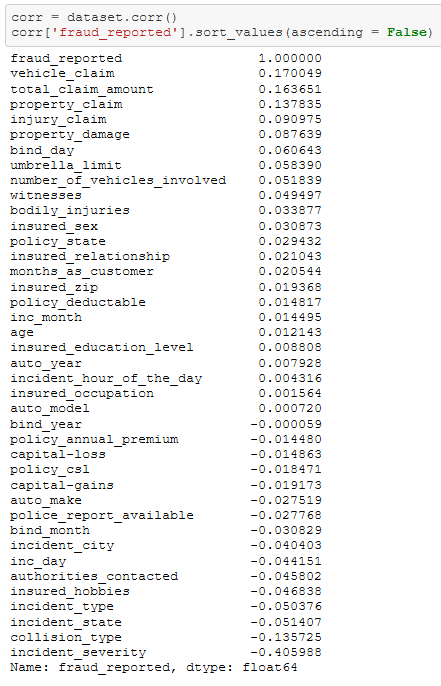
Checking for the data types



I’m encoding the dependent variable with the Label Encoder and using Ordinal Encoder on all the remaining object type variables using the below code

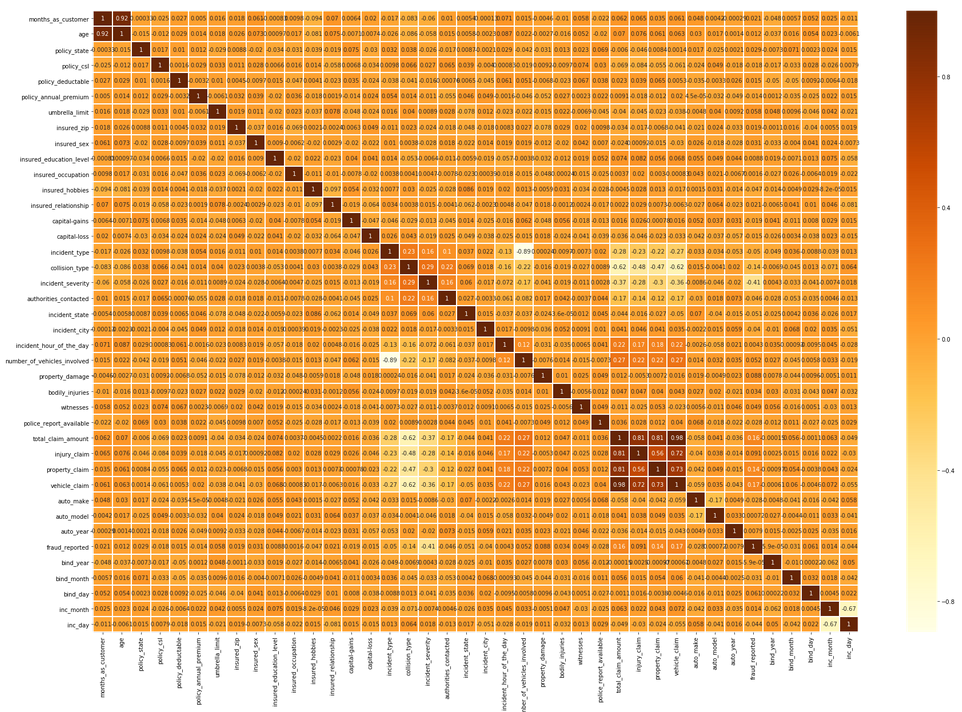


Checking the actual correlation numbers for the independent variable with the target variable.



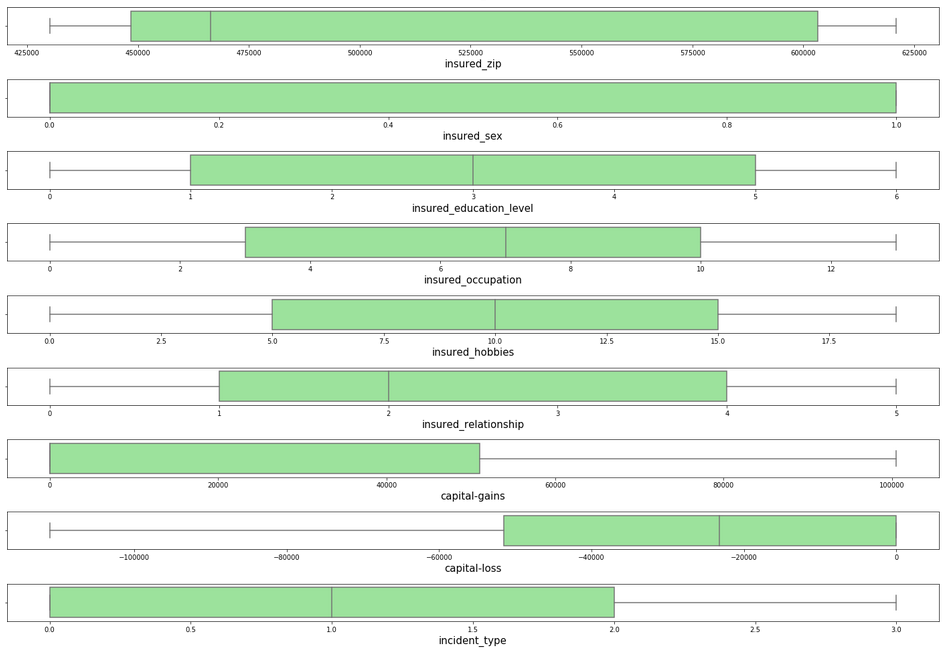
I can say that the attributes **incident\_severity**, **vehicle\_claim**, **total\_claim\_amount** and **property\_claim** has the highest correlation with the target variable.

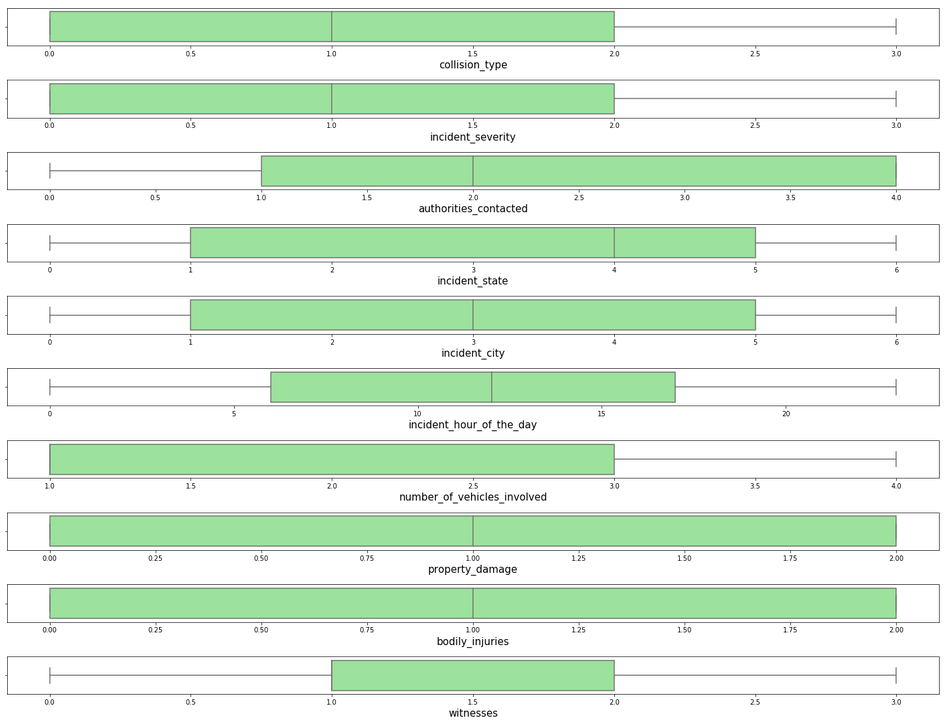
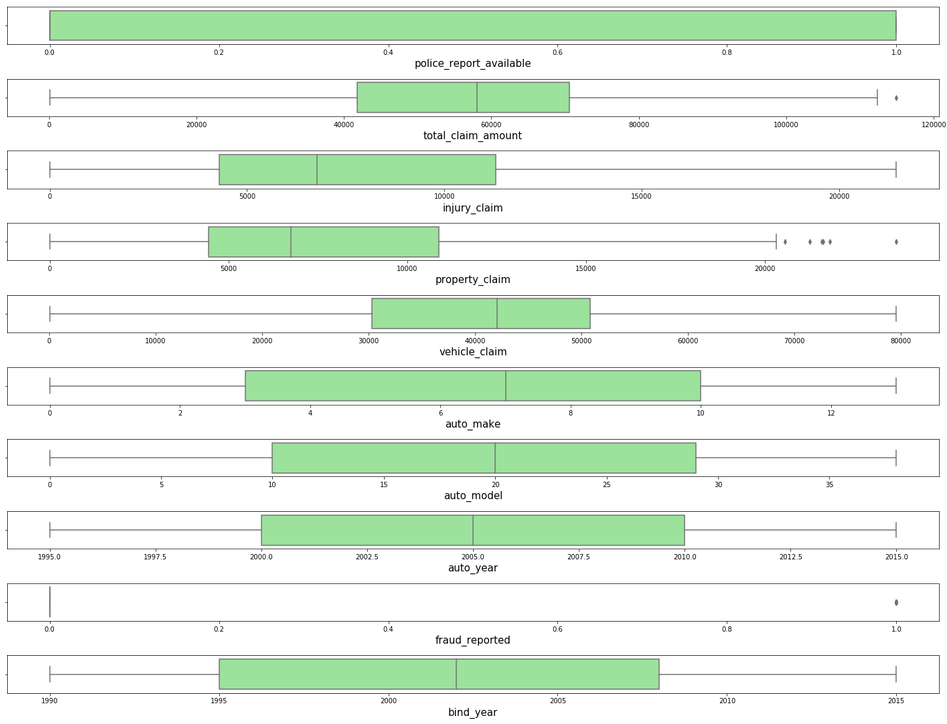
We can now plot the correlation table using heat map from the seaborn library to visualize the correlation and check for any multi-collinearity.

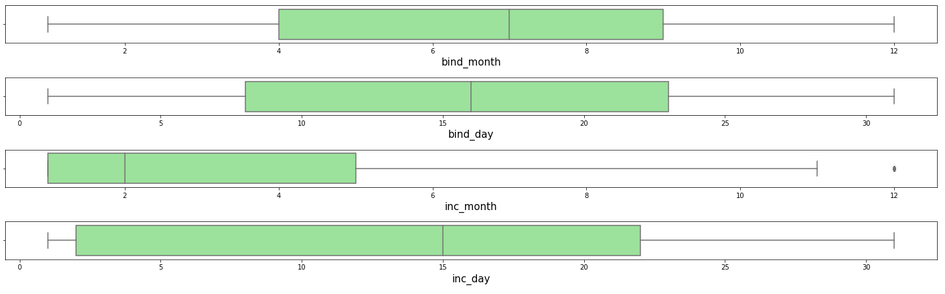


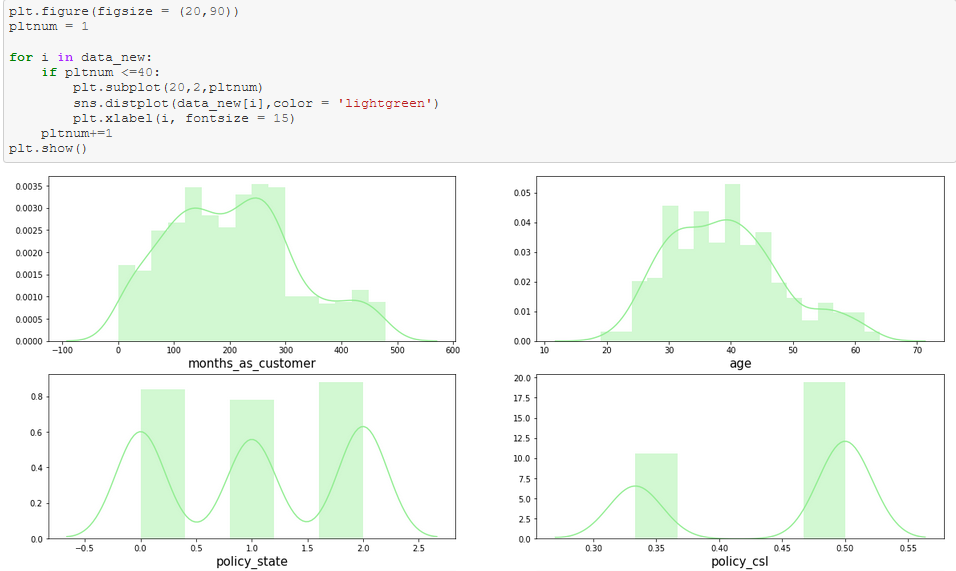
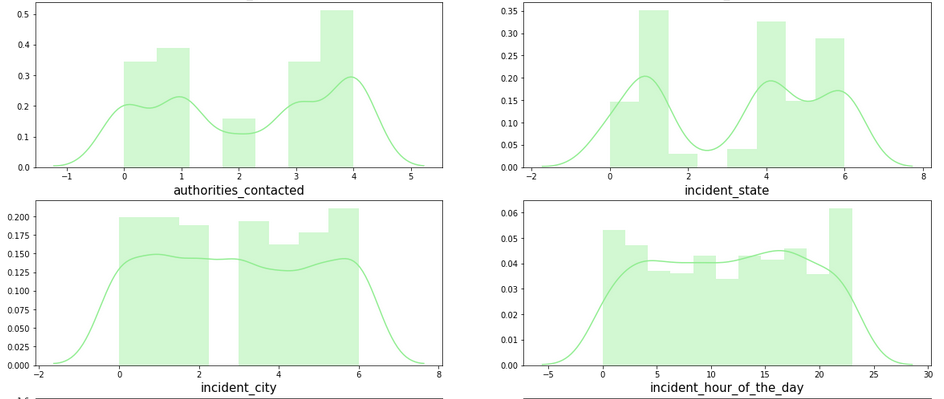
I can see that some of the variables show multi-collinearity in the dataset, it is not severe and the multi-collinearity will not affect the predictions.

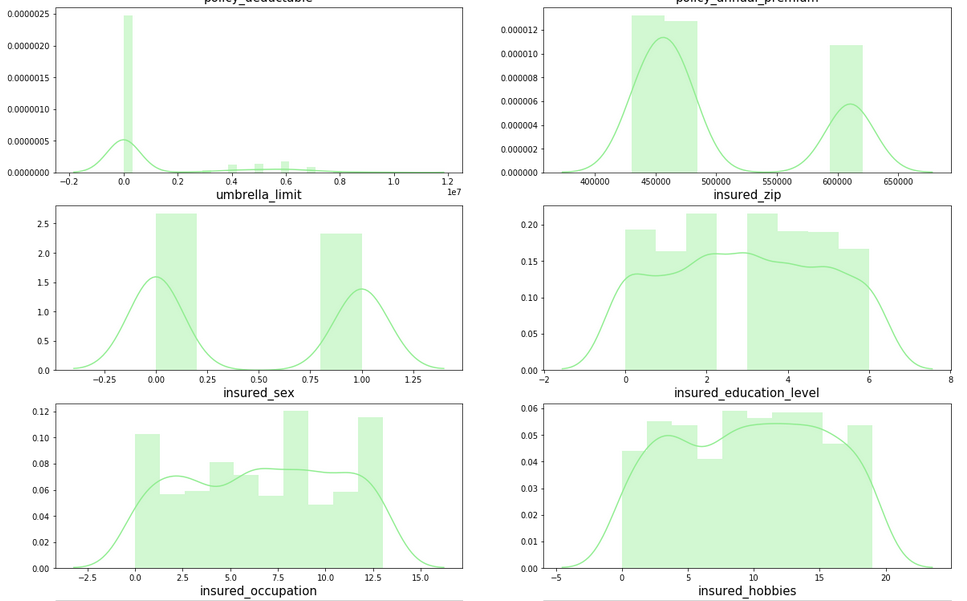
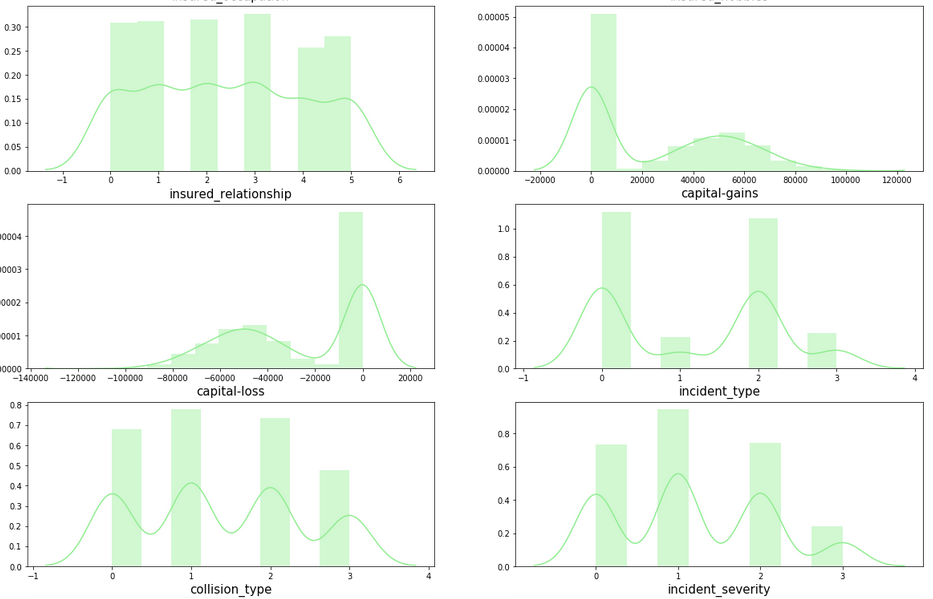
We can now proceed with checking for outliers using boxplot from seaborn library

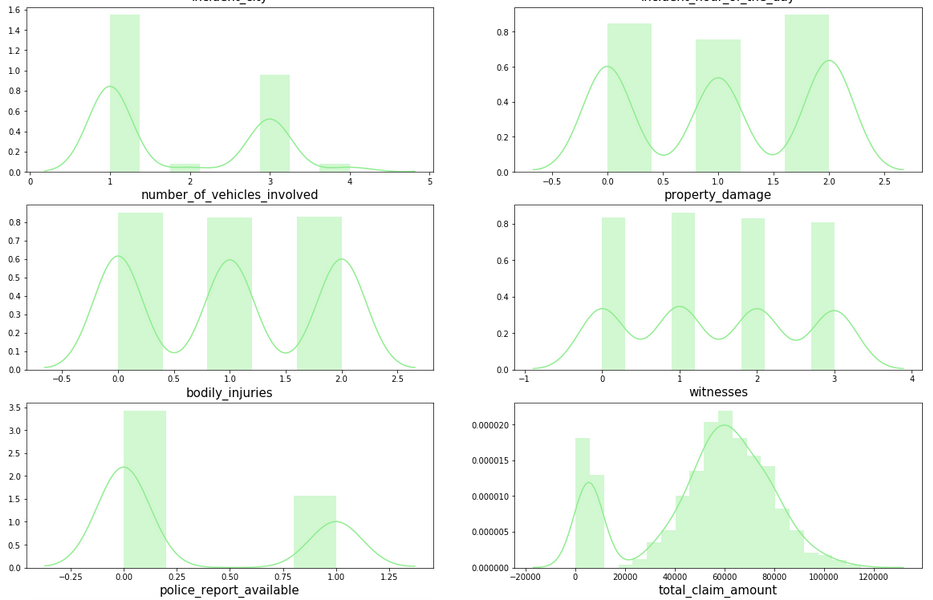
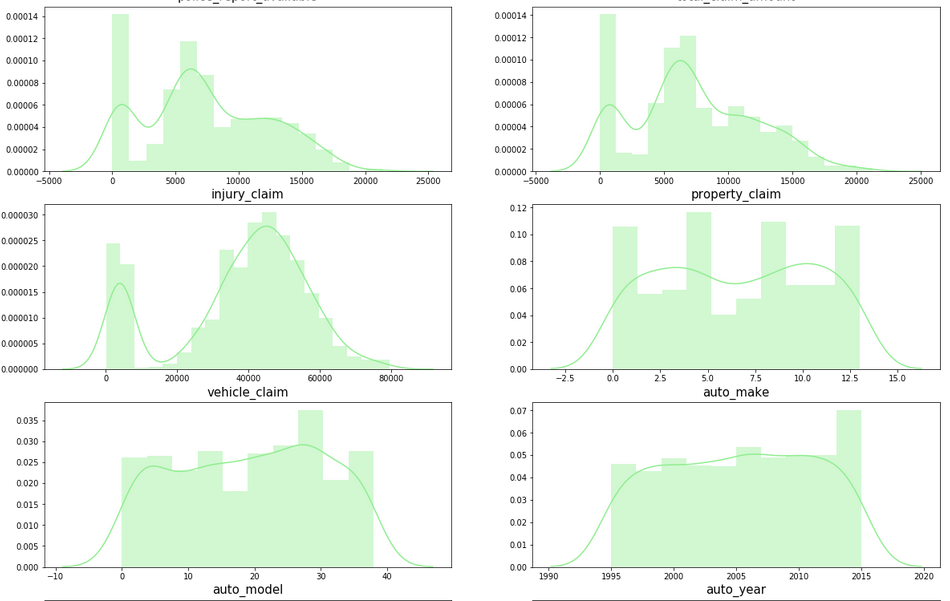


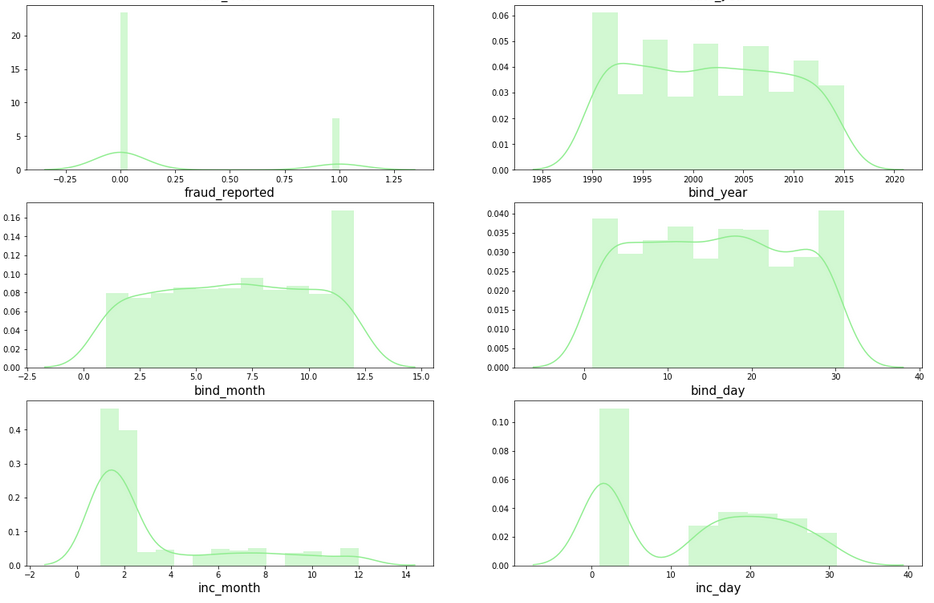
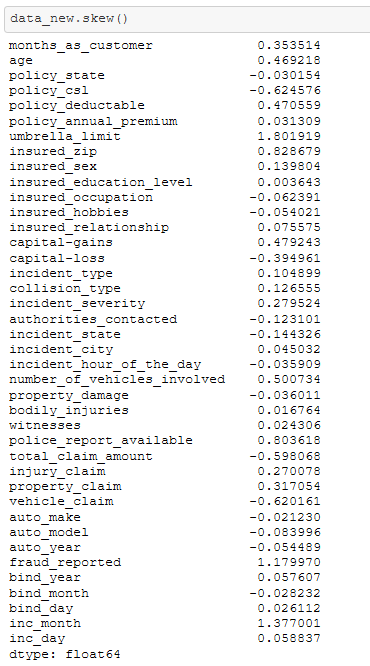




I can see there aren’t much outliers in the data, let’s proceed to look at the data distribution using distplot from seaborn library.





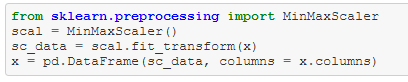
Although there aren’t any outlier, I can see some skewness in the data. Let’s see the actual skewness in the dataset.

We can see that the skewness for few continuous variables are not in the range of -0.5 to +0.5 which is considered to be normal.

Since I’m applying transformation technique on the independent features we will split the data to x (independent variable) and y(dependent variable)



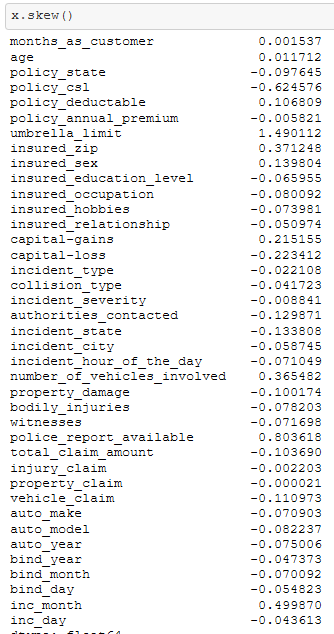
Once we have split the data, I’m scaling the data with a Min Max Scaler before applying the transformation.



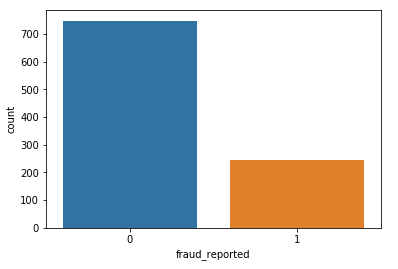
The data consists of negative values post scaling, therefore applying the yeo-johnson’s transformation to reduce the skewness in the data.



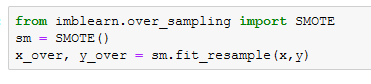
Let’s check the skewness again.



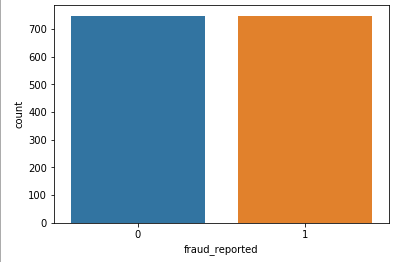
Now the skewness is brought under control. I can proceed with the model building, but before that I’m checking for the class imbalance in the dataset because, if there is a class imbalance then the machine learning models tend to predict the majority class better than the other.



I can defenitely see the class imbalance from the above plot and I’m using SMOTE over sampling technique to address the same before building the Machine Learning model.

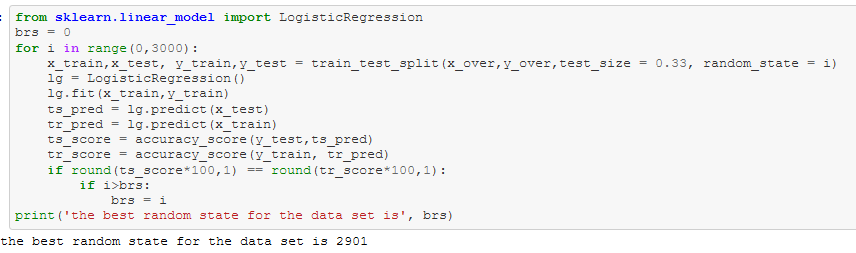


Verifying the class, post oversampling



I can now say that the class has been balanced. Hence proceeding with further steps.

I’m splitting the data in the upcoming step. However, before splitting the data I’m finding the best random state to split the data, this may control the over fitting of the model to certain extent.



The logic is simple, for a particular random state, if the accuracy score of the training dataset is equal to the accuracy score of test data then, the random state is considered as best random state for the split.

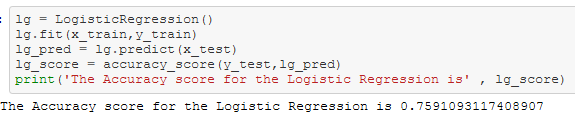
Now we got the best random state, therefore proceeding with the split.



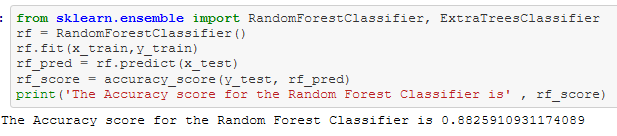
The data is now split into 0.77:0.33 proportion, let’s build Machine learning models on the same.

We will determine the best model based on the accuracy score and the cross validated accuracy score. Once the best model is determined, we’ll look at more performance metrics for the same

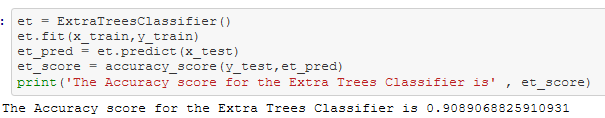
**Model 1: Logistic Regression**



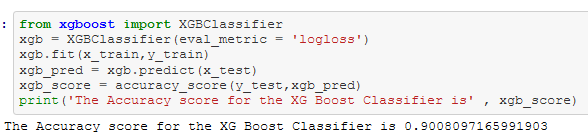
**Model 2: Random Forest Classifier**



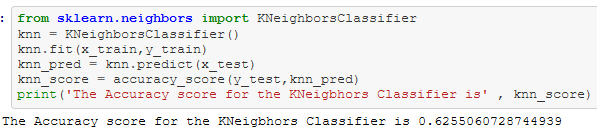
**Model 3: Extra Trees Classifier**



**Model 4: XG Boost Classifier**

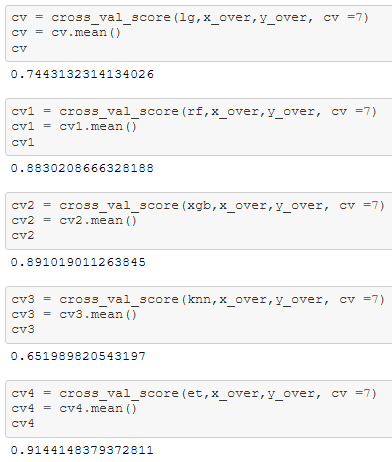


**Model 5: K-Neighbors Classifier**



**Cross validation scores of each model**

Here I’m using the 7 fold cross validation

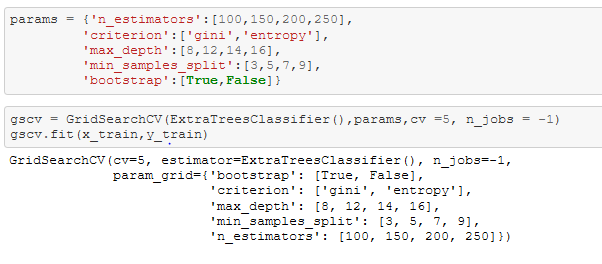


Let’s look at the differences in the accuracy score to identify the best model for the dataset.

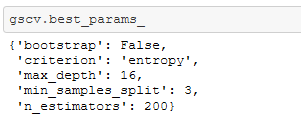
|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Model's Accuracy** | **Cross Validated Accuracy** | **Difference** |
| Logistic Regression | 0.759109 | 0.744313 | 0.014796 |
| Random Forest Classifier | 0.882591 | 0.883021 | -0.00043 |
| XGB Classifier | 0.90081 | 0.891019 | 0.009791 |
| K-Neighbors Classifier | 0.625506 | 0.65199 | -0.026484 |
| Extra Trees Classifier | 0.908907 | 0.914415 | -0.005508 |

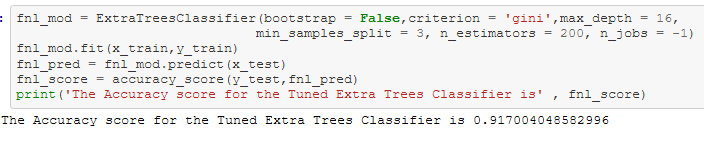
From the above table I can see that there wan’t much difference in the scores of Extra Tress Classfier and it is reaching the accuracy score of 0.91. Therefore considering the model as the best fit and proceeding with the hyper parameter tuning.

**Hyper-parameter Tuning**

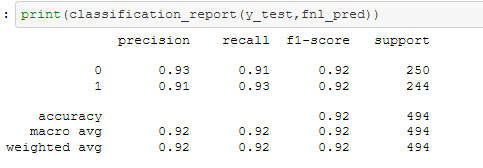


Finding the best paramters, fitting them into the model and training the dataset.





The final hyper tuned model is giving me the accuracy of 0.92. let’s look at few more metrics

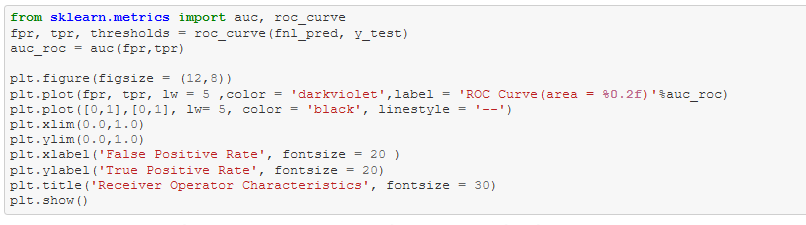


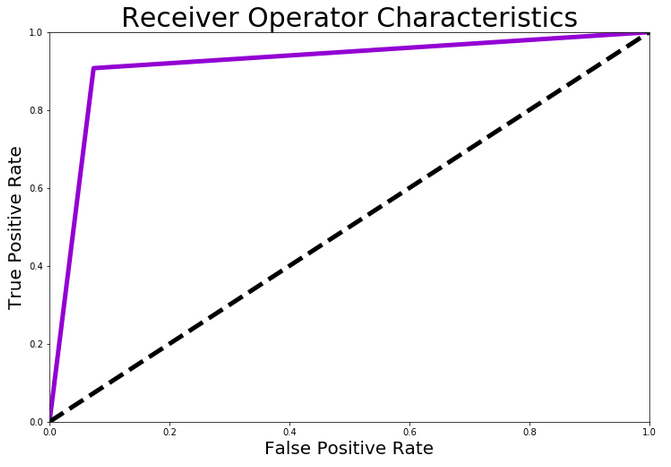
I can see from the above figure that the average precision, recall and the F1 scores are also 0.92.

Let’s look at the model’s ability to distingush between the classes using the AUC ROC score



Plotting the AUC ROC Curve using the below code





For most of the datasets I have build Machine Learning model using the Extra Trees Algorithm and it gave me better score than the other models almost every time.

Extra Trees is an ensemble machine learning algorithm that combines the predictions from many decision trees.

It is related to the widely used random forest algorithm. It can often achieve as-good or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble.

It is also easy to use given that it has few key hyperparameters and sensible heuristics for configuring these hyperparameters.