Final Project Report

Fraud Detection in Auto Insurance Claims

Analysis Using Machine Learning Techniques

Prepared By:

Umair Chaanda

Executive Summary:

Globally, insurance fraud is a major concern for Insurers which continues to increase year by year. Claims fraud is the most common buzz around the P&C Insurance industry with the auto segment being the major contributor. Car insurance fraud occurs when someone deceives an auto insurance company in order to benefit financially. Some car insurance fraud cases are more severe than others, but fraud is by no means a victimless crime. Fraud losses and risks can lead to price increases for loyal customers as well as introduce additional time and review before insurers pay legitimate claims. This increased scrutiny of honest customers is only visible when they feel most vulnerable and are in the greatest need of the insurer's services. According to Coalition Against Insurance Fraud, each year US insurance companies lose more than $80 billion due to fraud, and this results in increased premiums for every stakeholder. Using predictive modeling, insurers can compare a person's data against past fraudulent profiles and identify cases that require more investigation.

The main goal of this project is the implementation of various data mining and machine learning techniques and their applications. The exploration of multiple data analysis tasks on the targeted data, including both supervised knowledge discovery (predictive modeling) as well as unsupervised knowledge discovery for exploratory data analysis. The basic objective is to create a best predictive model for classifying fraud, whether a claimant is likely to commit fraud in auto insurance claims. This is a pure classification task which also includes the misclassification costs. Accurate prediction of fraud in claims will lead to a number of benefits described above to several stakeholders.

To accomplish this goal, some of the important numerical and categorical features are analyzed in relation with the main parameter of interest "fraud_detected". We experimented and compared with various classifiers provided as part of the scikit-learn machine learning module, as well as with some of its parameters optimization and model evaluation capabilities. In particular, we used the following list of classification algorithms to classify the data: (*K-Nearest-Neighbor (KNN) , Naive Bayes (Gaussian), Decision Tree, Linear Discriminant Analysis (LDA), Rocchio, Random Forest, Ada Boost, Gradient Boosting, Support Vector Machines, Logistic Regression*). The accuracy of the different models were compared to select the final model for prediction. The *Gradient Boosting* model was selected based on its performance. It had the highest test accuracy rate of 0.865. The future recommendation is to test and run this model in real time on a bigger dataset, since our dataset had only around 1000 instances. This model can flag potential fraud in insurance claims and can lead to reduction in insurance premiums, claim turnaround time, and increase company's profitability.

Dataset:

The data used in this project is publicly-available and acquired from Kaggle[1]. It contains information about historic claims and policy information. It has 39 columns including numerical, categorical and other types of variables with 1000 rows.

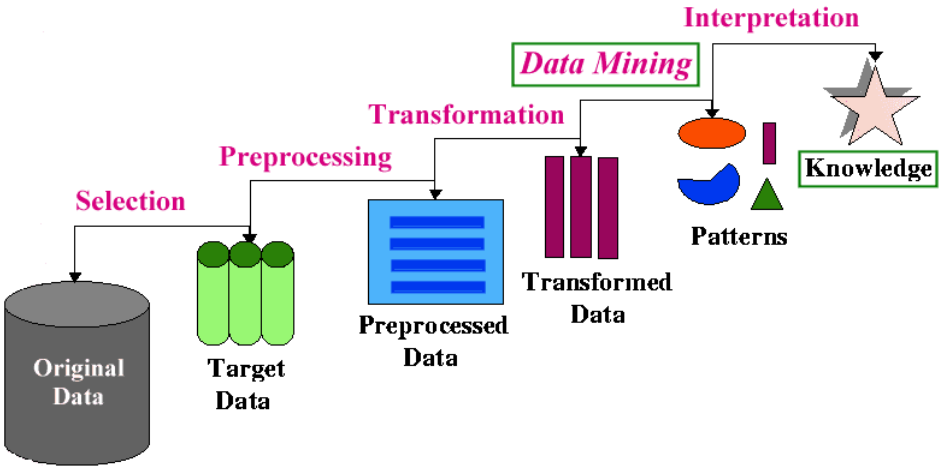
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 39 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          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
dtypes: float64(1), int64(17), object(21)
memory usage: 304.8+ KB
```

[1] https://www.kaggle.com/roshansharma/insurance-claim

KDD Process:

The unifying goal of the KDD process is to extract knowledge from data in the context of large databases. The following main tasks will be performed as part of the project:

- Identifying and separating the target attribute
- Cleaning and Preprocessing of the data which includes data discretization, converting data types, handling missing data, dropping some features, splitting data into training and test sets, normalize / scaling the train and test data, and converting data into standard spreadsheet format.
- Data explorations (using statistical approaches to provide an overview of data characteristics).
- Data Visualization.
- Clustering - Finding patterns and possibility of dimensionality reduction
- Applying machine learning techniques on the data.
- Building 10 different classification models, and evaluating predictive models using GridSearchCV from Scikit-learn.
- Using GridSearchCV from Scikit-learn, evaluate predictive models on 10-fold cross validation and experiment with different parameters to optimize the results.
- Comparing multiple classifier models for the same classification task.
- Comparing average accuracy score on the test and the training data sets.
- Thorough discussion and analysis of data mining results, including an analysis of how the approaches used worked in accomplishing the project objectives.
- Draw Conclusions
- Future Recommendations



Tools Used:

The project involves the use of Python scripts in Jupyter Notebook to perform various data analysis or mining tasks including available modules or libraries such as:

- NumPy: For carrying out efficient computations
- Pandas: For reading and writing spreadsheets
- Matplotlib: For displaying plots
- Scikit-learn: Machine Learning Library for several ML tasks
- Seaborn: For visualization of data

Data Cleaning, Exploration and Preprocessing:

There were several data cleaning, data exploration and preprocessing steps were performed as part of the project. Some of the which are following:

Information About Data:

After reading the csv dataset into a Dataframe in python, the first step was to look at the general information about the data such as feature names and their types, missing values, number of unique values and the categorical levels for each feature. The second step was to look at the basic statistics of features to explore the general characteristics of the data as a whole: examine the means, standard deviations, and other statistics associated with the numerical attributes.

Data Discretization:

Data discretization is defined as a process of converting continuous data attribute values into a finite set of intervals and associating with each interval some specific data value. The attribute (*incident_hour_of_the_day*) had 24 uniques values and it was not suitable to treat this variable as numerical so we discretized the incident_hour_of_the_day attribute into 3 categories corresponding to *'Morning'*, *'Afternoon'*, *'Evening'* (*bins=[0, 11, 17, 23]*) using pandas pd.cut function.

Convert some Features Types:

In many situations, we need to convert variables from one type into another. Type conversion is a method of changing features from one data type to another. An example of typecasting is converting an integer to a string. The features (*'policy_deductable', 'number_of_vehicles_involved', 'bodily_injuries', and 'witnesses'*) had very few unique values and they are actually categorical in nature so we converted the data type of these features from int64 into objects. We also converted the feature (*'policy_annual_premium'*) from float to int64 for efficient computation.

Handling Missing Data and Drop some Features:

The total number of rows in the data is 1000 and there are three features which have a very large number of rows with missing values. Those features are *collision_type, property_damage*, and *policy_report_available*. These are categorical attributes so instead of removing a very large number of rows from the data, we are just going to drop these three columns. Finally, we also dropped some of the variables (*'policy_number', 'policy_bind_date', 'insured_zip', 'incident_date', 'incident_location', 'auto_model'*) which are not going to be used in our analysis.

General Information of the Cleaned Data:

The total number of rows and columns in the cleaned data 1000 x 30. We can see in the picture that there are no missing values in the data and all the features are converted into appropriate types.

```
# this is the reduced number of features in the data
print(claims_clean.shape)
```

```
(1000, 30)
```

	Dtype	Levels	Null_Count	Number_Unique_Values
months_as_customer	int64	[328, 228, 134, 256, 137, 165, 27, 212, 235, 4...	0	391
age	int64	[48, 42, 29, 41, 44, 39, 34, 37, 33, 61, 23, 3...	0	46
policy_state	object	[OH, IN, IL]	0	3
policy_csl	object	[250/500, 100/300, 500/1000]	0	3
policy_deductable	object	[1000, 2000, 500]	0	3
policy_annual_premium	int64	[1406, 1197, 1413, 1415, 1583, 1351, 1333, 113...	0	612
umbrella_limit	int64	[0, 5000000, 6000000, 4000000, 3000000, 800000...	0	11
insured_sex	object	[MALE, FEMALE]	0	2
insured_education_level	object	[MD, PhD, Associate, Masters, High School, Col...	0	7
insured_occupation	object	[craft-repair, machine-op-inspct, sales, armed...	0	14
insured_hobbies	object	[sleeping, reading, board-games, bungie-jumpin...	0	20
insured_relationship	object	[husband, other-relative, own-child, unmarried...	0	6
capital-gains	int64	[53300, 0, 35100, 48900, 66000, 38400, 52800, ...	0	338
capital-loss	int64	[0, -62400, -46000, -77000, -39300, -51000, -3...	0	354
incident_type	object	[Single Vehicle Collision, Vehicle Theft, Mult...	0	4
incident_severity	object	[Major Damage, Minor Damage, Total Loss, Trivi...	0	4
authorities_contacted	object	[Police, None, Fire, Other, Ambulance]	0	5
incident_state	object	[SC, VA, NY, OH, WV, NC, PA]	0	7
incident_city	object	[Columbus, Riverwood, Arlington, Springfield, ...	0	7
number_of_vehicles_involved	object	[1, 3, 4, 2]	0	4
bodily_injuries	object	[1, 0, 2]	0	3
witnesses	object	[2, 0, 3, 1]	0	4
total_claim_amount	int64	[71610, 5070, 34650, 63400, 6500, 64100, 78650...	0	763
injury_claim	int64	[6510, 780, 7700, 6340, 1300, 6410, 21450, 938...	0	638
property_claim	int64	[13020, 780, 3850, 6340, 650, 6410, 7150, 9380...	0	626
vehicle_claim	int64	[52080, 3510, 23100, 50720, 4550, 51280, 50050...	0	726
auto_make	object	[Saab, Mercedes, Dodge, Chevrolet, Accura, Nis...	0	14
auto_year	int64	[2004, 2007, 2014, 2009, 2003, 2012, 2015, 199...	0	21
fraud_reported	object	[Y, N]	0	2
incident_part_of_the_day	category	[Morning, Evening, Afternoon] Categories (3, o...	0	3

Basic Statistics of Features on Cleaned Data:

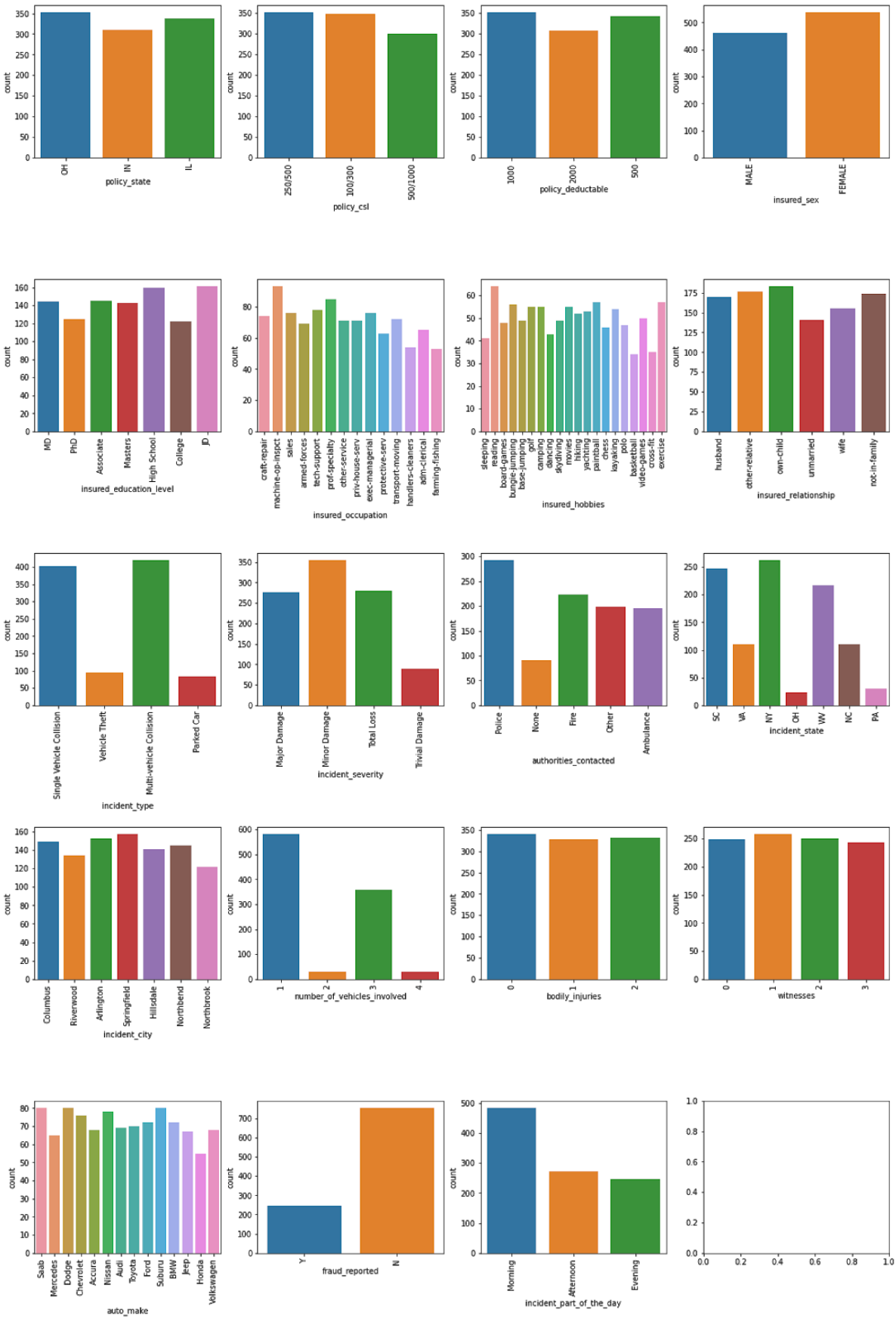
Let's explore the general characteristics of the data as a whole: examine the means, standard deviations, and other statistics associated with the numerical attributes.

The target feature (*'fraud_reported'*) has a class imbalance (N=No Fraud Reported has 753 instances) whereas (Y=Yes Fraud Reported has 247 instances).

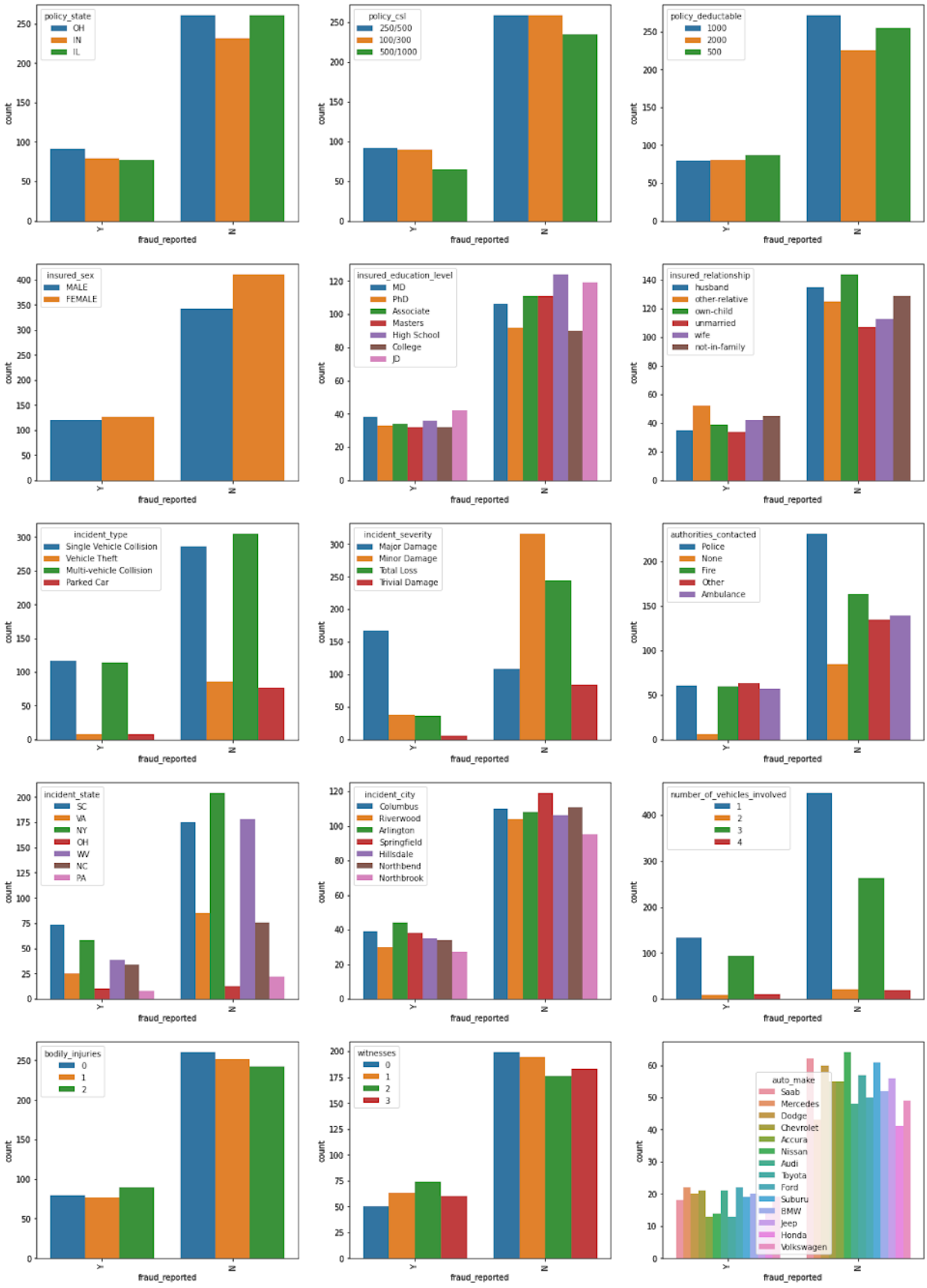
Below is the table which displays the total count of instances, number of unique features, top categorical value and frequency of that value, and some statistical measures such as Standard Deviation, Minimum value, 25% percentile, 50% percentile, 75% percentile, and the maximum value of each feature.

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
months_as_customer	1000	NaN	NaN	NaN	203.954	115.113	0	115.75	199.5	276.25	479
age	1000	NaN	NaN	NaN	38.948	9.14029	19	32	38	44	64
policy_state	1000	3	OH	352	NaN	NaN	NaN	NaN	NaN	NaN	NaN
policy_csl	1000	3	250/500	351	NaN	NaN	NaN	NaN	NaN	NaN	NaN
policy_deductable	1000	3	1000	351	NaN	NaN	NaN	NaN	NaN	NaN	NaN
policy_annual_premium	1000	NaN	NaN	NaN	1255.91	244.145	433	1089.5	1257	1415	2047
umbrella_limit	1000	NaN	NaN	NaN	1.101e+06	2.29741e+06	-1e+06	0	0	0	1e+07
insured_sex	1000	2	FEMALE	537	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_education_level	1000	7	JD	161	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_occupation	1000	14	machine-op-inspct	93	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_hobbies	1000	20	reading	64	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_relationship	1000	6	own-child	183	NaN	NaN	NaN	NaN	NaN	NaN	NaN
capital-gains	1000	NaN	NaN	NaN	25126.1	27872.2	0	0	0	51025	100500
capital-loss	1000	NaN	NaN	NaN	-26793.7	28104.1	-111100	-51500	-23250	0	0
incident_type	1000	4	Multi-vehicle Collision	419	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_severity	1000	4	Minor Damage	354	NaN	NaN	NaN	NaN	NaN	NaN	NaN
authorities_contacted	1000	5	Police	292	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_state	1000	7	NY	262	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_city	1000	7	Springfield	157	NaN	NaN	NaN	NaN	NaN	NaN	NaN
number_of_vehicles_involved	1000	4	1	581	NaN	NaN	NaN	NaN	NaN	NaN	NaN
bodily_injuries	1000	3	0	340	NaN	NaN	NaN	NaN	NaN	NaN	NaN
witnesses	1000	4	1	258	NaN	NaN	NaN	NaN	NaN	NaN	NaN
total_claim_amount	1000	NaN	NaN	NaN	52761.9	26401.5	100	41812.5	58055	70592.5	114920
injury_claim	1000	NaN	NaN	NaN	7433.42	4880.95	0	4295	6775	11305	21450
property_claim	1000	NaN	NaN	NaN	7399.57	4824.73	0	4445	6750	10885	23670
vehicle_claim	1000	NaN	NaN	NaN	37928.9	18886.3	70	30292.5	42100	50822.5	79560
auto_make	1000	14	Suburu	80	NaN	NaN	NaN	NaN	NaN	NaN	NaN
auto_year	1000	NaN	NaN	NaN	2005.1	6.01586	1995	2000	2005	2010	2015
fraud_reported	1000	2	N	753	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_part_of_the_day	1000	3	Morning	483	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Visualize Distribution of Categorical Features:



Visualize Cross-Tabulation of Categorical Attributes:



The above visualization is the Cross Tabulated View using Barplots for Comparing categorical attributes with respect to target attribute (*fraud_reported*). This requires the aggregation of the occurrences of each *fraud_reported* value (*Y or N*) separately for each value of the other attributes. Here, we are using the SEABORN visualization package to create bar charts graphs to visualize the relationships between these sets of variables.

Following are some of the key points observed from these visualizations:

- **Policy_state**: All of the claims or instances are from the three states (OH, IN, IL) and OH state has the highest number of fraud in claims compared to the other two states (IN, IL).

- **Insured_relationship**: This is the relationship of the person with the policyholder who committed the fraud in insurance claims. It looks like that the category 'other-relative' has significantly higher fraud in claims compared to the other insured_relationships (husband, wife, own-child etc.) .

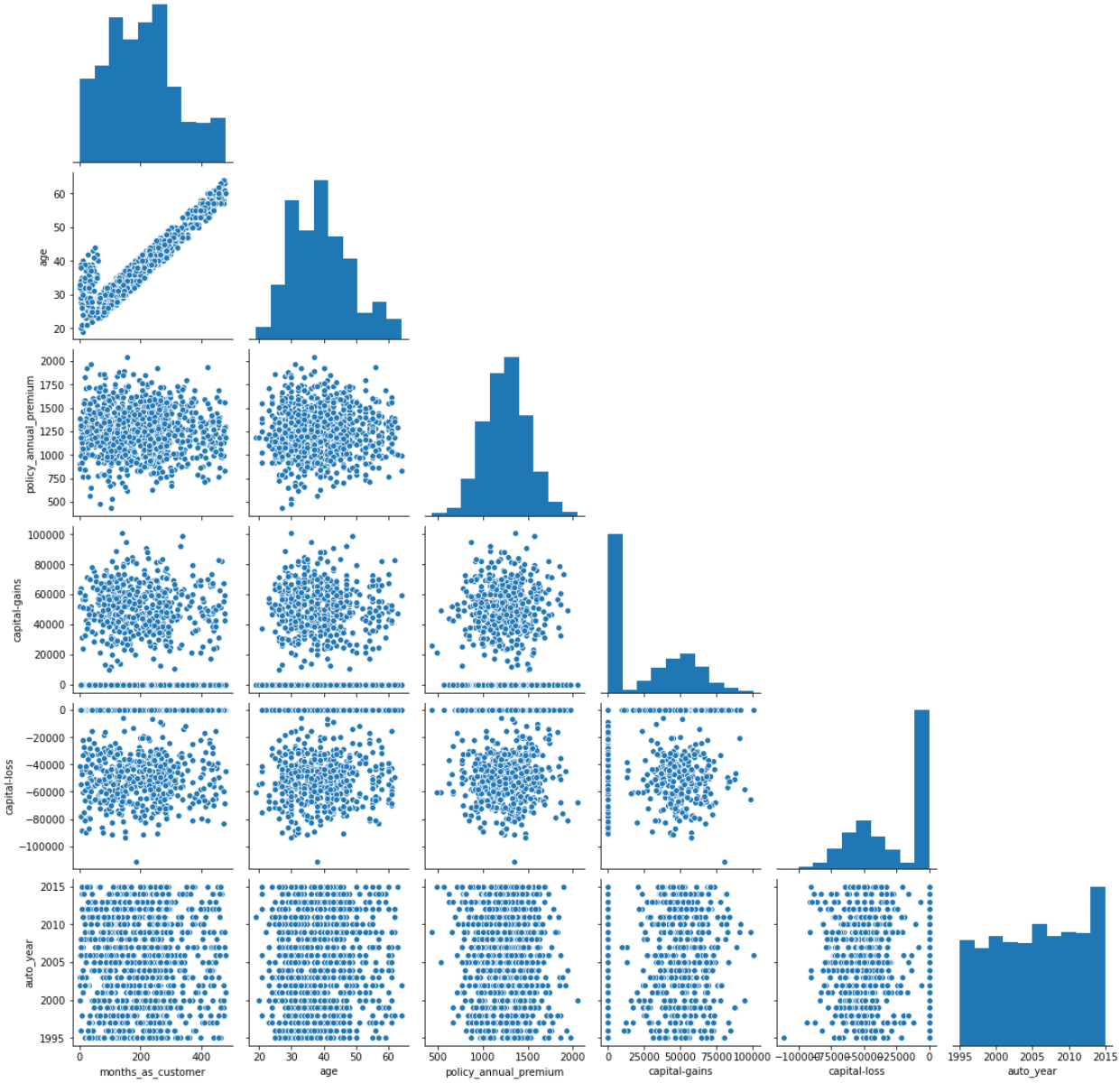
- **Incident_type**: Single Vehicle Collisions and Multi Vehicle Collisions have significantly higher number of fraud in claims compared to the Vehicle Theft and Parked Car. This actually makes sense because in collision type claims, there is not only the damage to the vehicle but also the bodily injuries sustained to the driver and the passengers. The claims amount for the bodily injuries sustained is usually higher than just the property damage.

- **Incident_severity**: We can observe that there is a huge spike in vehicle claims fraud when it comes to the severity of the incident. People tend to commit more fraud when there is a major damage to the vehicle. One of the reasons for this could be that the person does not want to commit a fraud when there is not too much damage or claim amount involved.

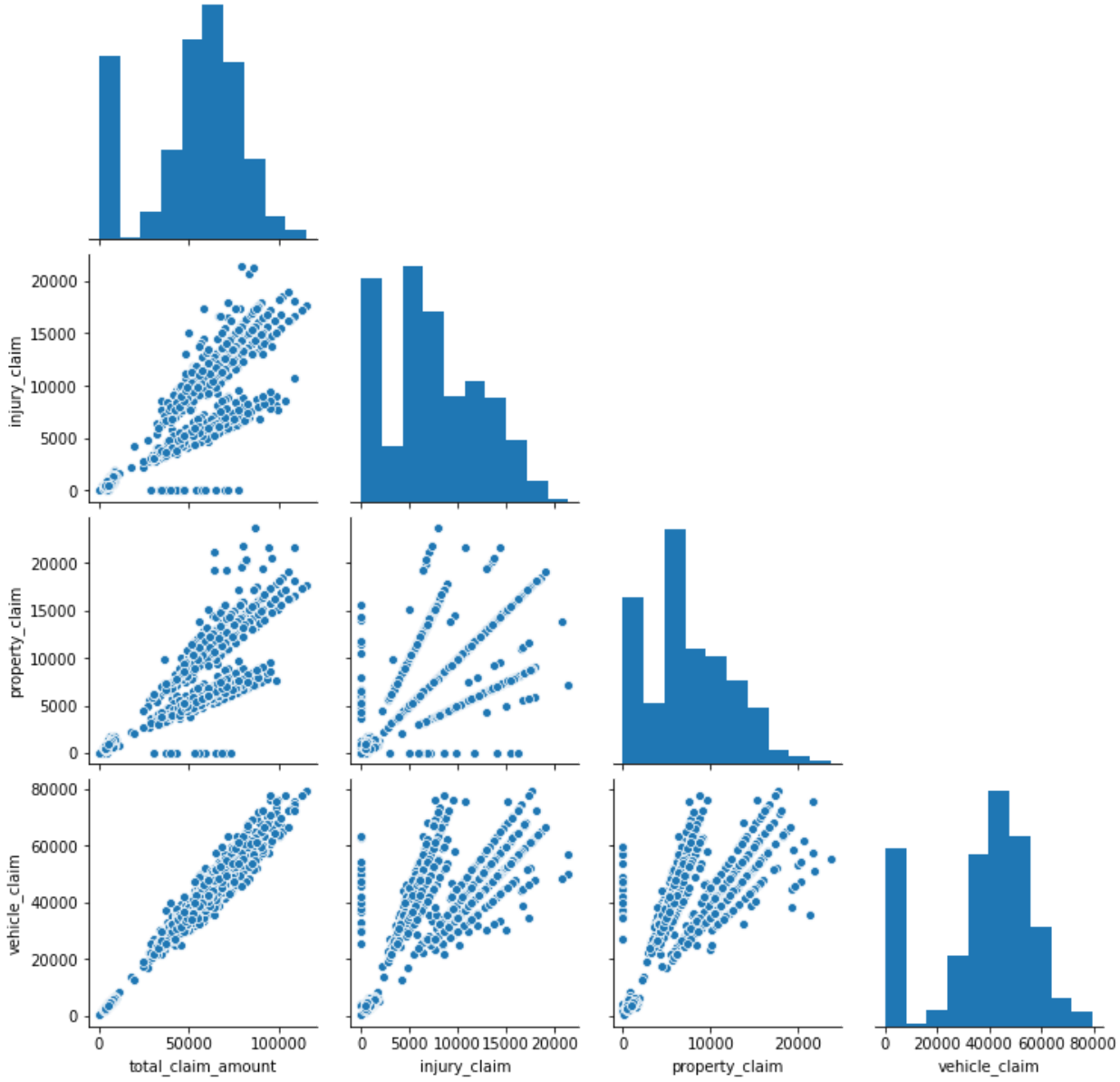
- **Number_of_vehicles_involved**: This is surprising to know that there is more fraud in claims when only one vehicle is involved in the accident. It may be easier to hide the facts and figures when it comes to just reporting single vehicle collisions.

- **Incident_part_of_the_day**: Most of the accidents occurred during midnight or early morning when the fraud is detected in claims.

Visualize Pairwise Relationships of Numerical Features:

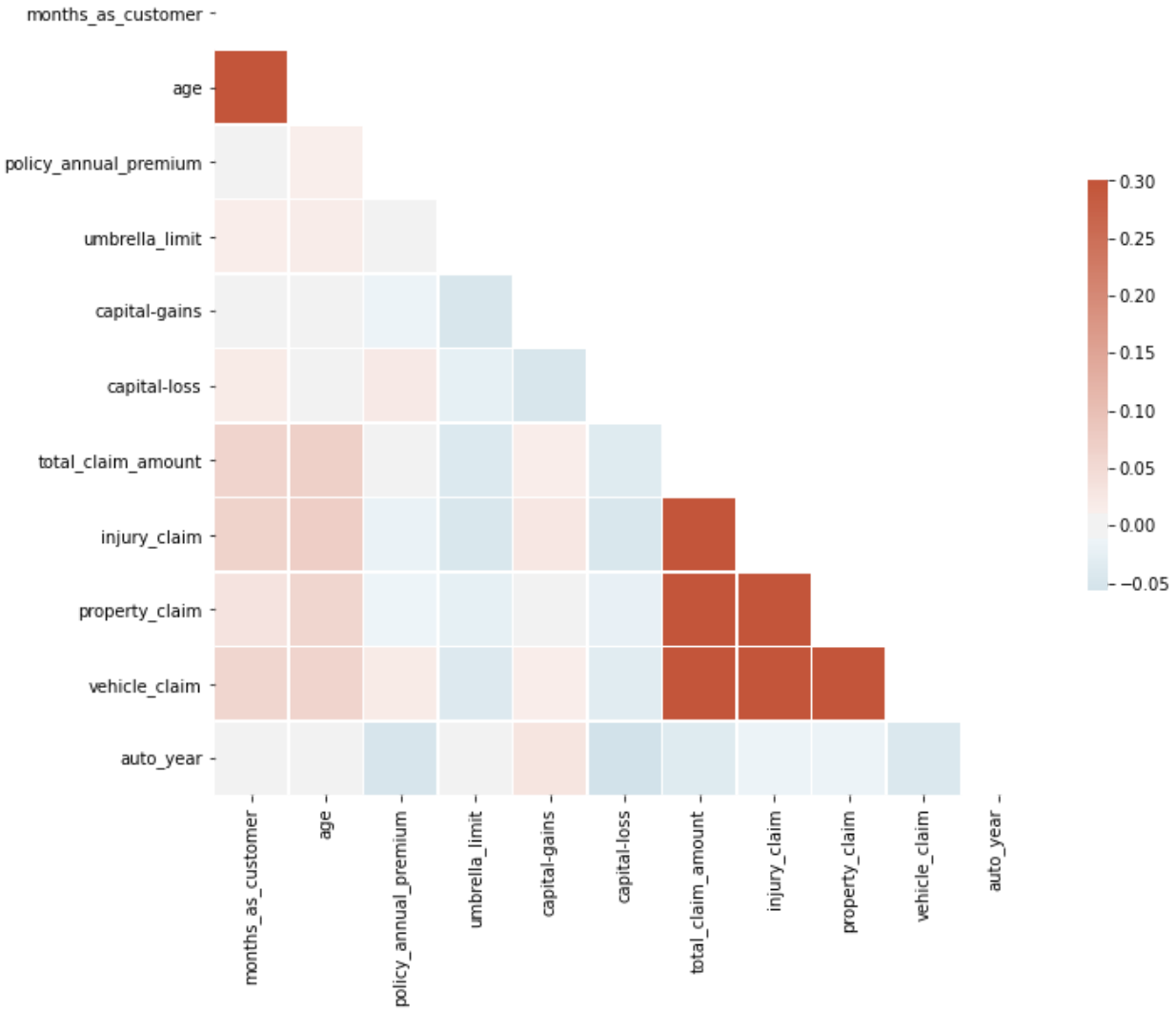


For this group of variables, we can see that the feature ‘age’ has a strong positive linear relationship with months_as_customer. There don't appear to be any significant outliers in this plot. However, if we look at the other features, there do not seem to be any positive or negative relationships among them so we are not too much concerned about the multicollinearity in the data.



If we look at this group of features (*total_claim_amount, injucy_claim, property_claim, vehicle_claim*), they definitely have strong positive relationships among them. One of the reasons is that the total_claim_amount is the sum of *injury_claim*, *property_claim*, and *vehicle_claim*.

Visualize Correlation Matrix:



Using the numeric attributes, we performed basic correlation analysis among the attributes. The above Complete Correlation Matrix shows any significant positive or negative correlations among pairs of attributes. A correlation matrix is a table showing correlation coefficients between sets of variables. Correlation analysis is a very important step in pre-processing because it helps in identifying multicollinearity and building components in Principal Components Analysis (PCA).

The features that seem Strong Positively Correlated are (*total_claim_amount, injucy_claim, property_claim, vehicle_claim*). This is definitely a distinct group (orange squared boxes) of features. This group is the same set of features which we identified above in the scatterplot matrix.

Split Data into Training and Test Sets:

The next step after cleaning and exploration of variables is to separate the target attribute ("fraud_reported") and the attributes used for model training. We created a separate data frame which contains the records without target attribute. Then we pulled the target attribute and stored it separately.

Once we separated the target attribute from the data, then we Split the data into training and test sets (using 80%-20% randomized split). Note that the same split was also performed on the target attribute). We set aside the 20% test portion; and the 80% training data partition was used for model training, cross-validation and various other tasks specified in the beginning.

Normalization / Rescale Numeric Features:

The next thing that we did was performing min-max normalization to rescale numeric features. We used scikit-learn's preprocessing package for Min-Max Normalization to transform the values of all numeric attributes in the table onto the range 0.0-1.0. We fit the MinMaxScaler on the training data first and then transformed the training and test data using this scaler.

Convert Data into Standard Spreadsheet Format:

This requires converting each categorical attribute into multiple binary ("dummy") attributes (one for each value of the categorical attribute) and assigning binary values corresponding to the presence or not presence of the attribute value in the original record). The numeric attributes should remain unchanged. We first transformed the numerical features above because in general, it is best to avoid normalizing dummy variables. Even if the numerical values do not change, the underlying data type will be changed to float. Once converted to float, the variables are no longer treated as mutually exclusive binary values. It is not detrimental to KNN, but it can make the models lose the benefit of using one-hot encoding for many other ML algorithms (especially neural networks).

auto_make_Jeep	auto_make_Mercedes	auto_make_Nissan	auto_make_Saab	auto_make_Suburu	auto_make_Toyota	auto_make_Volkswagen
0	0	0	0	1	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	1	0	0

Discover Groupings in Data (Unsupervised Learning):

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. In this section, we experimented with an unsupervised clustering technique (KMeans) using Scikit-learn implementation and also the dimensionality reduction technique called Principal Components Analysis (PCA).

KMeans Clustering:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. The objective of K-means clustering to group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset. The K-means algorithm identifies the k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

After extracting the quantitative features from the data, we fit the KMeans algorithm using scikit-learn KMeans package where we used (n_clusters=6, max_iter=500, verbose=1). Then we utilized the scikit-learn *.cluster_centers_ function* to get following centroids:

Cluster Centroids:

	months_as_customer	age	policy_annual_premium	umbrella_limit	capital-gains	capital-loss	auto_year	total_claim_amount	injury_claim	property_claim
0	0.32	0.35	0.51	0.17	0.11	0.97	0.76	0.53	0.40	0.38
1	0.56	0.56	0.52	0.20	0.09	0.50	0.25	0.59	0.48	0.41
2	0.41	0.42	0.50	0.21	0.23	0.73	0.51	0.05	0.04	0.04
3	0.30	0.34	0.53	0.18	0.36	0.96	0.22	0.53	0.39	0.36
4	0.30	0.34	0.49	0.19	0.33	0.40	0.65	0.54	0.42	0.36
5	0.75	0.72	0.52	0.18	0.38	0.84	0.64	0.52	0.38	0.38

In the above cluster centroids, the first row is the Mean vector associated with cluster 0. These are mean values for each one of the attributes. Centroids help us categorize different groups and see why there are certain attributes in different clusters.

Cluster Sizes:

```
Size of Cluster 0 =  116
Size of Cluster 1 =  116
Size of Cluster 2 =  155
Size of Cluster 3 =  127
Size of Cluster 4 =  171
Size of Cluster 5 =  115
```

Evaluate Clusters:

To evaluate clusters, we performed Silhouette analysis on the clusters (compute Silhouette values for all instances in the data, and then compute the overall mean Silhouette value). One way to measure the quality of clustering is to compute the Silhouette values for each instance in the data. The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It is the ratio of the difference between in-cluster dissimilarity and the closest out-of-cluster dissimilarity, and the maximum of these two values.

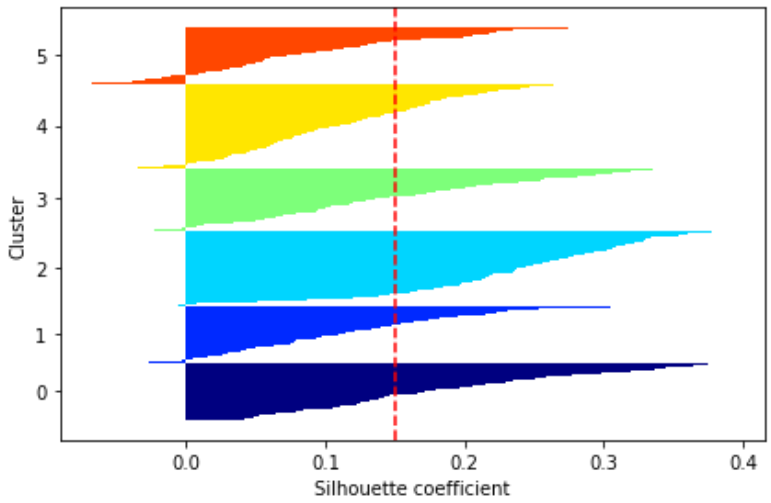
Mean of silhouettes values:

```
# mean of silhouettes values
# value closest to 1 is better
# we can use different number of k values and compare
print(silhouettes.mean())
```

```
0.14972254040960067
```

The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and well separated from other clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

Visualization of the Silhouettes:



Some of the clusters are largest in size in terms of Silhouette coefficient. Dotted line is the mean silhouette value. Some clusters are significantly above the mean silhouette value.

Completeness and Homogeneity:

```
Completeness :  0.012149124861312529
Homogeneity  :  0.03861986585765167
```

Homogeneity: Each cluster contains only members of a single class. The homogeneity score approaches 1 when all the clusters contain almost only data points that are members of a single class.

Completeness: All members of a given class are assigned to the same cluster. The completeness score approaches 1 when most of the data points that are members of a given class are elements of the same cluster.

Principal Components Analysis (PCA):

Principal Component Analysis (PCA) is a statistical procedure that allows us to summarize the information contained in a large set by means of a smaller set of "summary indices" that can be more easily visualized and analyzed. It is a very common technique for "dimensionality reduction" and finding the "latent/hidden" factors from the data.

- The goal of PCA is to simplify model features into fewer, uncorrelated features to help visualize patterns in the data and help it run faster.
- It reduces the number of variables while maintaining the majority of the important information.
- It can help solve the very common problem of "Curse of Dimensionality".
- We should only apply PCA to continuous data and the data should be scaled before applying PCA technique.
- PCA can be very helpful in predicting the parameter of interest.
- PCA can help us build the parsimonious model which is easier to explain.
- PCA can also be used in relation with other clustering techniques for better results.

Principle Components:

```
[[ 0.09 -0.14  0.05 -0.53 -0.29 -0.07]
 [-0.31 -0.25  0.3   0.02  0.37 -0.11]
 [ 0.59 -0.35  0.17  0.34  0.24 -0.1 ]
 ...
 [ 0.41  0.14 -0.62  0.09 -0.22  0.38]
 [-0.02  0.01  0.49 -0.11  0.25  0.18]
 [-0.16  0.65  0.5  -0.23 -0.25  0.26]]
```

The rows are still the claims data but the columns are the reduced # of features OR the principal components. We can use these components to cluster our claims data. This is something similar to reduced features in classification.

```
print(pca.explained_variance_ratio_)
```

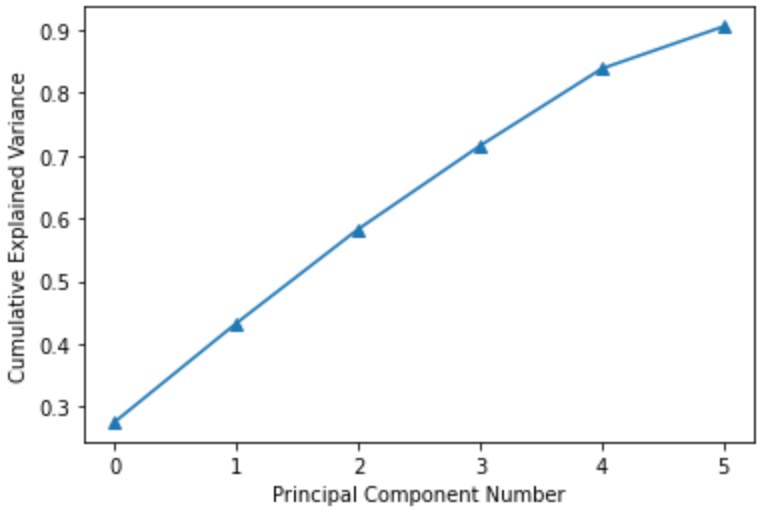
```
[0.28 0.16 0.15 0.13 0.12 0.07]
```

```
sum(pca.explained_variance_ratio_ * 100)
```

```
90.53794091774775
```

First PC explains 28% of variance in the data. Second PC explains 0.16% of variance in the data. They all combined represent about 90% of variance in the data. It can be observed that the first 6 components capture (explain) 90% of the variance in the data.

Plot of PC Variances:



These 6 components account for just over 90% of the variance.

KMeans on Transformed Data:

Perform Kmeans again, but this time on the lower dimensional transformed data.

```
Completeness on full data              :  0.012149124861312529
Completeness using PCA                 :  0.012755436483637712

Homogeneity on full data               :  0.03861986585765167
Homogeneity using PCA                  :  0.04054020981456376
```

We can observe from the above comparison that the Mean of silhouettes values improved using lower dimensional transformed data. It means that the quality of clustering has improved. However, there is not a very big difference in the completeness and homogeneity values.

The case for doing a cluster analysis (or other dimensionality reduction methods such as PCA) is probably to reduce the number of features in a way that the learning model is more robust. In the case of this dataset, we do not have that many number of good numerical variables to group.

Clustering (unsupervised learning technique) comes into the play as our saviour when we do not have data to supervise. It is in general not a good idea to turn supervised learning (classification) into unsupervised learning (clustering) as it will sacrifice the vital information: labels.

Build and Evaluate Predictive Models (Classification):

We are going to experiment with various classification models provided as part of the scikit-learn (sklearn) machine learning module using the pre-processed insurance claims data set. In particular, we will use the following list of classification algorithms to classify the data:

- K-Nearest-Neighbor (KNN)
- Naive Bayes (Gaussian)
- Decision Tree
- Linear Discriminant Analysis (LDA)
- Rocchio
- Random Forest
- Ada Boost
- Gradient Boosting
- Support Vector Machines (SVC Classifier)
- Logistic Regression

The basic objective is to create a best predictive model for classifying fraud, whether a claimant is likely to commit fraud in auto insurance claims. This is a pure classification task which also includes the misclassification costs.

In addition to building the best model, we will also generate the confusion matrix (visualize it using Matplotlib), as well as the classification report.

Using GridSearchCV from Scikit-learn, evaluate predictive models on 10-fold cross validation. We will also experiment with GridSearchCV for parameters optimization to see if we can improve accuracy (we will not provide the details of all of our experimentation, but will provide a short discussion on what parameters worked best as well as our final results).

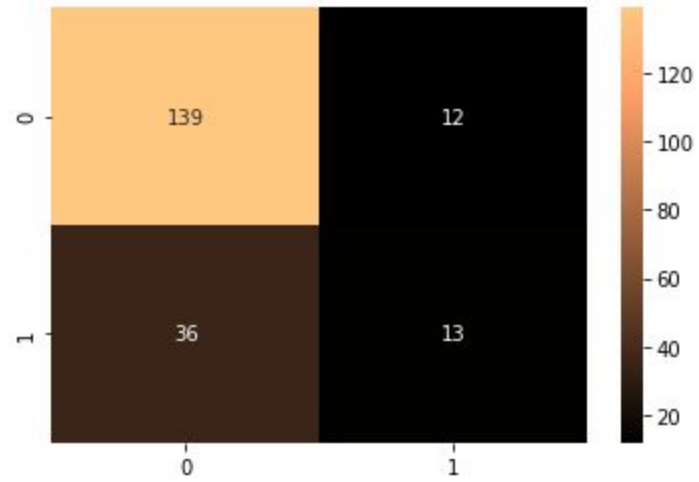
We are using Grid Search because it allows us to explore the parameter space more systematically and lets us select the best tuning parameters (aka "hyperparameters"). Grid Search allows us to define a grid of parameters that are searched using K-fold cross-validation.

Finally, the accuracy of the different models will be compared to select the final model for prediction and the overview of model performances will be presented in a tabulated form.

K-Nearest-Neighbor (KNN):

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve the classification problems. For each sample in data: It calculates the distance between the query example and the current example from the data. The algorithm is simple and easy to implement.

```
Wall time: 10.5 s

{'n_neighbors': 9, 'weights': 'distance'} 0.77374999995

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.79      0.92      0.85       151
           Y       0.52      0.27      0.35        49

    accuracy                           0.76       200
   macro avg       0.66      0.59      0.60       200
weighted avg       0.73      0.76      0.73       200


===============CONFUSION MATRIX===============
```



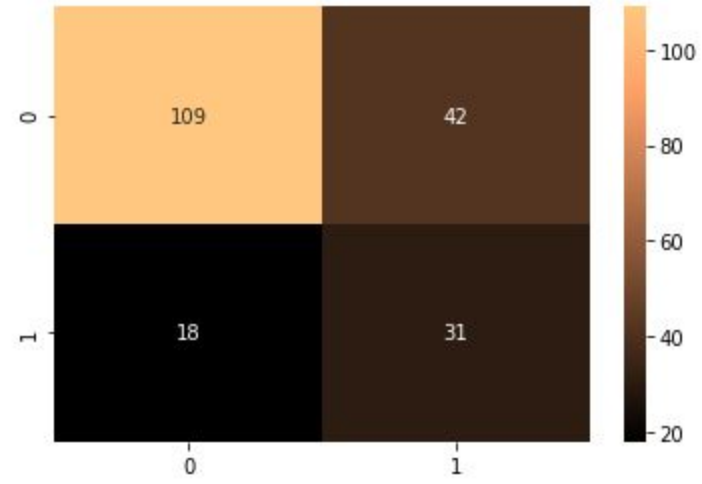

```
Training Accuracy: 1.000
Test Accuracy    : 0.760
```

The k-nearest neighbors (KNN) did not take longer time to run and the Wall Time is significantly lower. We can see the Wall time, best parameters, best CV score, Training accuracy, and test accuracy along with the Classification report and confusion matrix.

Naive Bayes (Gaussian) :

The Naïve Bayes Classifier belongs to the family of probability classifiers, using Bayesian theorem. The reason why it is called 'Naïve' is because it requires rigid independence assumption between input variables. Therefore, it is more proper to call Simple Bayes or Independence Bayes. This algorithm has been studied extensively since the 1960s. Simple though it is, Naïve Bayes Classifier remains one of popular methods to solve text categorization problems, the problem of judging documents as belonging to one category or the other, such as email spam detection. Bayes's theorem plays a critical role in probabilistic learning and classification.

```
Wall time: 274 ms

{'priors': None, 'var_smoothing': 1e-09} 0.705

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.86      0.72      0.78       151
           Y       0.42      0.63      0.51        49

    accuracy                           0.70       200
   macro avg       0.64      0.68      0.65       200
weighted avg       0.75      0.70      0.72       200


===============CONFUSION MATRIX===============
```

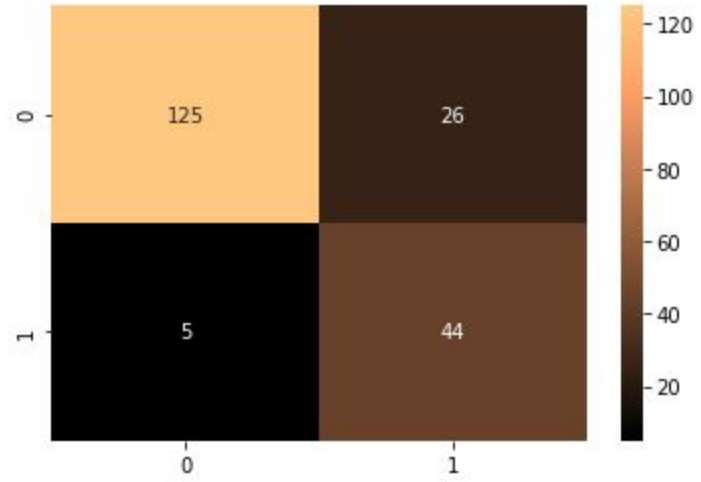



```
Training Accuracy: 0.762
Test Accuracy    : 0.700
```

Decision Tree:

Decision Tree algorithm belongs to the family of supervised learning algorithms. In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

```
Wall time: 3min 24s

{'criterion': 'gini', 'max_depth': 3.0, 'min_samples_leaf': 1,


'min_samples_split': 0.30000000000000004} 0.85625


===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.96      0.83      0.89       151
           Y       0.63      0.90      0.74        49

    accuracy                           0.84       200
   macro avg       0.80      0.86      0.81       200
weighted avg       0.88      0.84      0.85       200


===============CONFUSION MATRIX===============
```

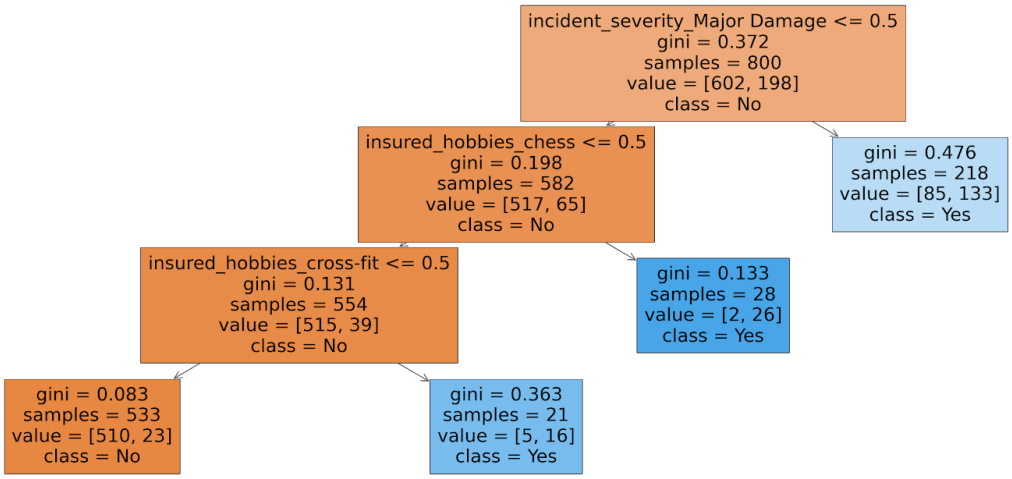



```
Training Accuracy: 0.856
Test Accuracy    : 0.845
```

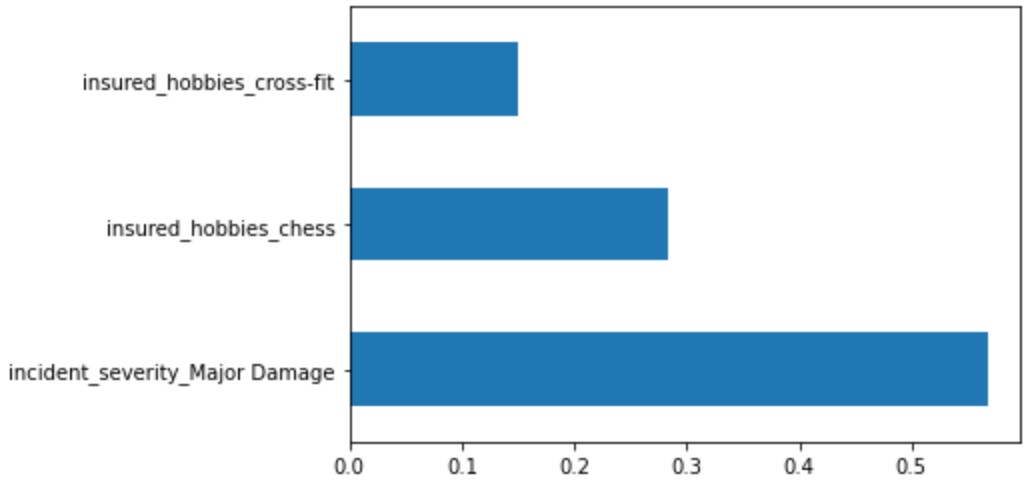
Visualizing the Decision Tree:

A decision tree is a flow-chart-like tree structure:

- Internal node denotes a test on an attribute (feature)
- Branch represents an outcome of the test
- All records in a branch have the same value for the tested attribute
- Leaf node represents class label or class label distribution



Feature Importances:



Linear Discriminant Analysis (LDA) :

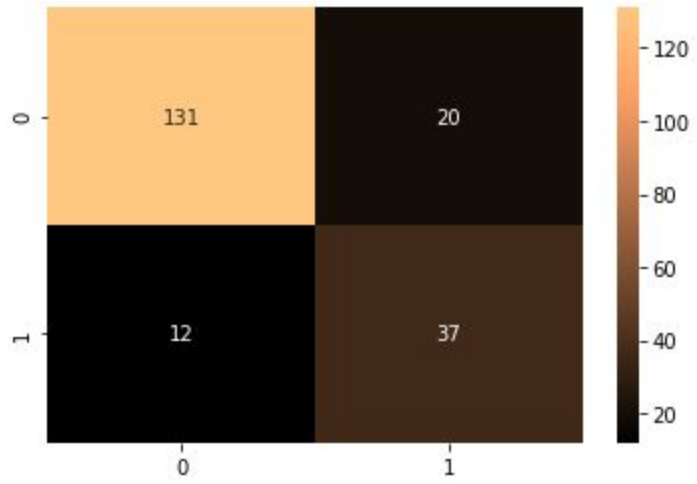
Linear discriminant analysis, normal discriminant analysis, or discriminant function analysis is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events. Wikipedia.

```
Wall time: 1min 7s

{'shrinkage': 0.052000000000000005, 'solver': 'lsqr'} 0.845
```

==========CLASSIFICATION REPORT==========

	precision	recall	f1-score	support
N	0.92	0.87	0.89	151
Y	0.65	0.76	0.70	49
accuracy			0.84	200
macro avg	0.78	0.81	0.79	200
weighted avg	0.85	0.84	0.84	200

==========CONFUSION MATRIX==========

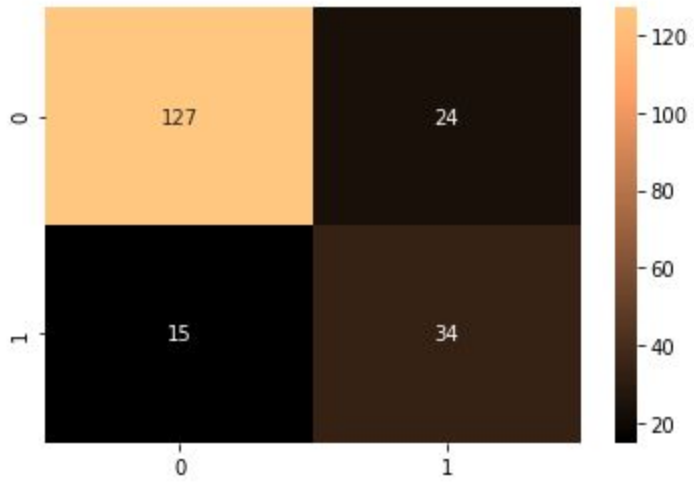



```
Training Accuracy: 0.885
Test Accuracy    : 0.840
```

Rocchio Method:

The Rocchio algorithm is based on a method of relevance feedback found in information retrieval systems which stemmed from the SMART Information Retrieval System which was developed 1960-1964. Like many other retrieval systems, the Rocchio feedback approach was developed using the Vector Space Model. Wikipedia.

```
Wall time: 30.8 s

{'metric': 'euclidean', 'shrink_threshold': 0.4100000000000003} 0.8125

==============CLASSIFICATION REPORT==============
              precision    recall  f1-score   support

           N       0.89      0.84      0.87       151
           Y       0.59      0.69      0.64        49

    accuracy                           0.81       200
   macro avg       0.74      0.77      0.75       200
weighted avg       0.82      0.81      0.81       200


===============CONFUSION MATRIX===============
```

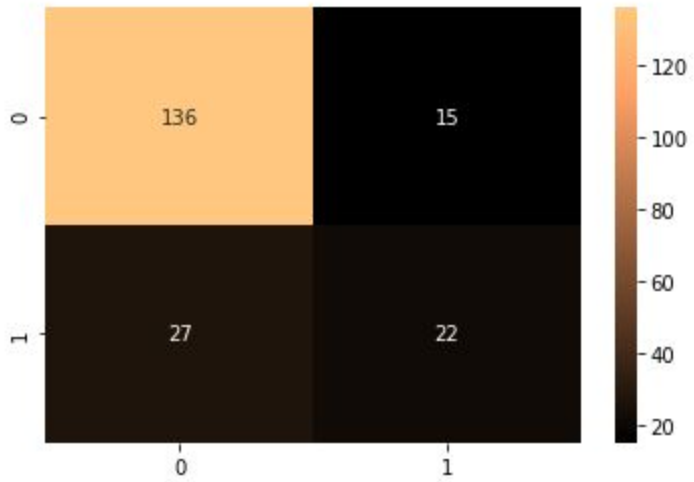



```
Training Accuracy: 0.816
Test Accuracy    : 0.805
```

Random Forest (an example of bagging):

- Each classifier in the ensemble is a decision tree classifier and is generated using a random selection of attributes at each node to determine the split.
- During classification, each tree votes and the most popular class is returned.
- Comparable in accuracy to Adaboost, but more robust to errors and outliers.
- Insensitive to the number of attributes selected for consideration at each split, and faster than boosting.

```
Wall time: 13min 22s

{'criterion': 'entropy', 'min_samples_split': 0.1, 'n_estimators': 5} 0.789

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.83      0.90      0.87       151
           Y       0.59      0.45      0.51        49

    accuracy                           0.79       200
   macro avg       0.71      0.67      0.69       200
weighted avg       0.78      0.79      0.78       200


===============CONFUSION MATRIX===============
```

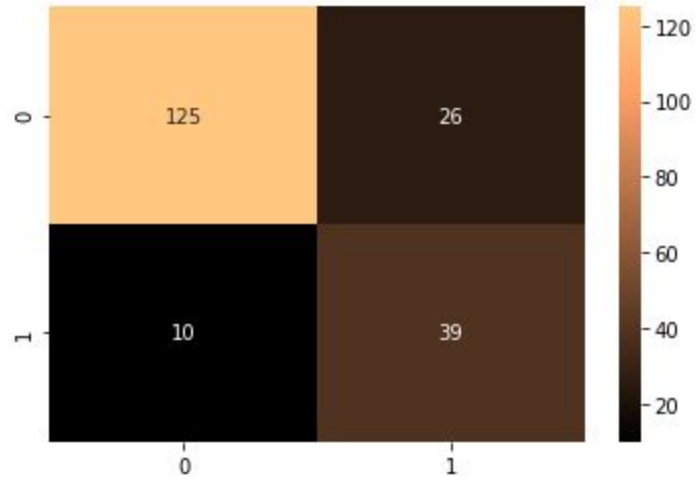



```
Training Accuracy: 0.830
Test Accuracy    : 0.790
```

Ada Boost:

AdaBoost, short for “Adaptive Boosting”, is the first practical boosting algorithm proposed by Freund and Schapire in 1996. It focuses on classification problems and aims to convert a set of weak classifiers into a strong one.

```
Wall time: 5min 32s

{'learning_rate': 1.8, 'n_estimators': 10} 0.836249995

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.93      0.83      0.87       151
           Y       0.60      0.80      0.68        49

    accuracy                           0.82       200
   macro avg       0.76      0.81      0.78       200
weighted avg       0.85      0.82      0.83       200


===============CONFUSION MATRIX===============
```




```
Training Accuracy: 0.843
Test Accuracy    : 0.820
```

Gradient Boosting:

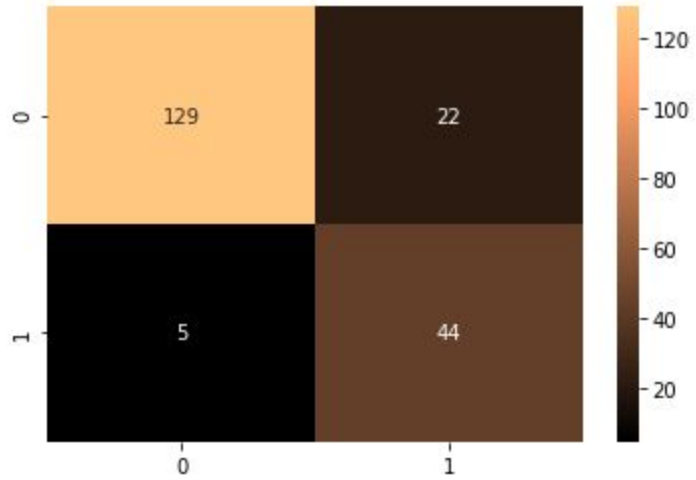
The following is an example of using Gradient Boosted Decision Trees. GBDT is a generalization of boosting to arbitrary differentiable loss functions. GBDT is an accurate and effective procedure that can be used for both regression and classification.

Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy

How boosting works?

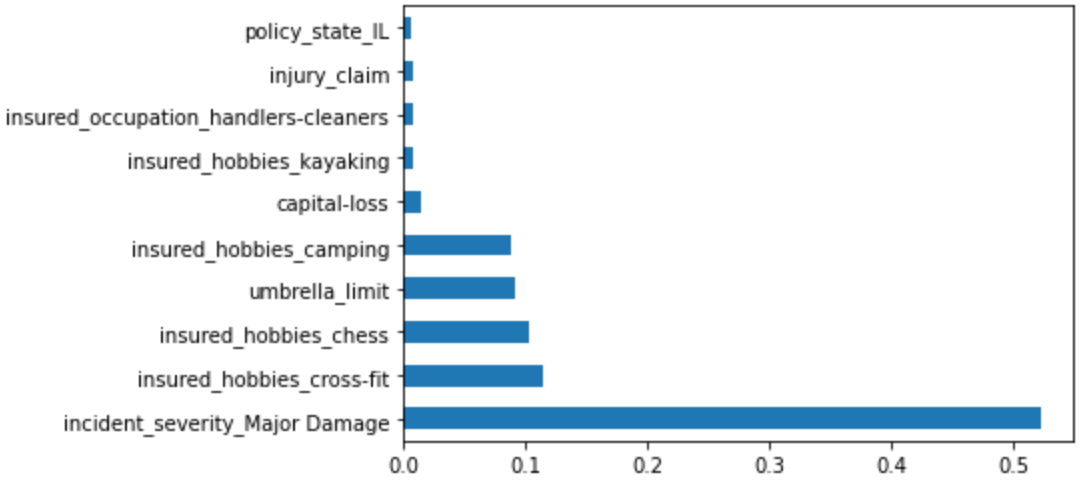
- Weights are assigned to each training tuple
- A series of k classifiers is iteratively learned
- After a classifier Mi is learned, the weights are updated to allow the subsequent classifier, Mi+1 , to pay more attention to the training tuples that were misclassified by Mi
- The final M* combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithms can be extended for numeric prediction. Compared to bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

```
Wall time: 5min 25s

{'learning_rate': 1.5, 'n_estimators': 5, 'random_state': 0} 0.8375

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.96      0.85      0.91       151
           Y       0.67      0.90      0.77        49

    accuracy                           0.86       200
   macro avg       0.81      0.88      0.84       200
weighted avg       0.89      0.86      0.87       200


===============CONFUSION MATRIX===============
```




```
Training Accuracy: 0.877
Test Accuracy    : 0.865
```

Feature Importances:

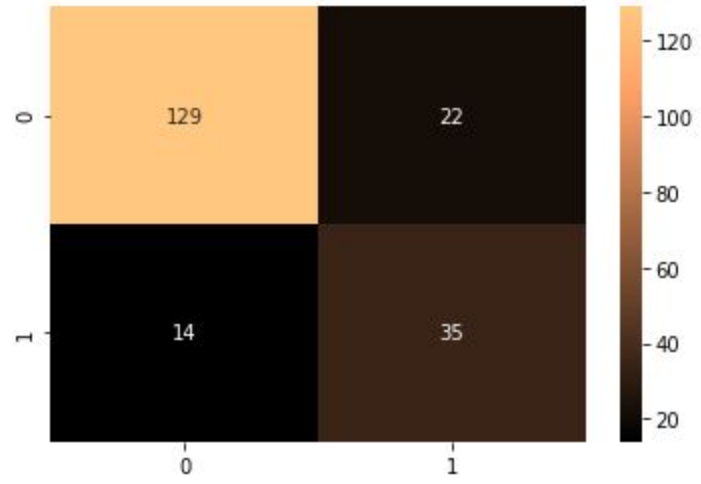


Support Vector Machines (SVC Classifier):

Many classification models, such as Naïve Bayes, try to model the distribution of each class, and use these models to determine labels for new points. Sometimes called generative classification models.

SVMs are discriminative classification models: Rather than modeling each class, they simply find a line or curve (in two dimensions) or a manifold (in multiple dimensions) that divides the classes from each other.

```
Wall time: 21min 6s

{'C': 5, 'degree': 1, 'kernel': 'poly'} 0.8525

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.90      0.85      0.88       151
           Y       0.61      0.71      0.66        49

    accuracy                           0.82       200
   macro avg       0.76      0.78      0.77       200
weighted avg       0.83      0.82      0.82       200


===============CONFUSION MATRIX===============
```

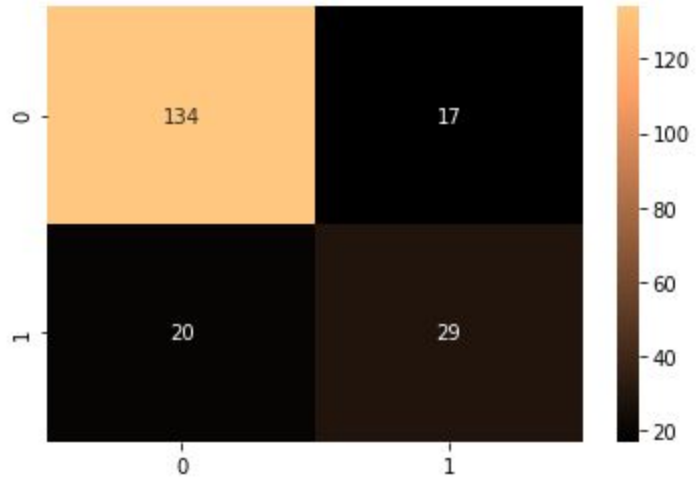



```
Training Accuracy: 0.882
Test Accuracy    : 0.820
```

Logistic Regression :

Logistic regression is a classification algorithm, used when the value of the target variable is categorical in nature. Logistic regression is most commonly used when the data in question has binary output, so when it belongs to one class or another, or is either a 0 or 1.

```
Wall time: 2min 36s

{'l1_ratio': 0.0001, 'penalty': 'l2'} 0.8125

===============CLASSIFICATION REPORT===============
              precision    recall  f1-score   support

           N       0.87      0.89      0.88       151
           Y       0.63      0.59      0.61        49

    accuracy                           0.81       200
   macro avg       0.75      0.74      0.74       200
weighted avg       0.81      0.81      0.81       200


===============CONFUSION MATRIX===============
```




```
Training Accuracy: 0.891
Test Accuracy    : 0.815
```

Conclusion and Future Recommendations:

To accomplish the classification goal, some of the important numerical and categorical features were analyzed in relation with the main parameter of interest “fraud_detected”. We experimented and compared with various classifiers provided as part of the scikit-learn machine learning module, as well as with some of its parameters optimization and model evaluation capabilities. In particular, we used the following list of classification algorithms to classify the data: (*K-Nearest-Neighbor (KNN) , Naive Bayes (Gaussian), Decision Tree, Linear Discriminant Analysis (LDA), Rocchio, Random Forest, Ada Boost, Gradient Boosting, Support Vector Machines, Logistic Regression*).

The below table shows the performance metrics of different models. The accuracy of the different models is compared to select the final model for prediction. The Gradient Boosting model was selected based on its performance. It had the highest test accuracy rate of 0.865. The Decision Tree model and LDA models have test accuracy rates around 0.84.

The future recommendation is to test and run this model in real time on a bigger dataset, since our dataset had only around 1000 instances. This model can flag potential fraud in insurance claims and can lead to reduction in insurance premiums, claim turnaround time, and increase company’s profitability.

Performance of Different Models:

Model	GSCV_BestScore	Train_Accuracy	Test_Accuracy
Gradient Boosting	0.84	0.88	0.86
Decision Tree	0.86	0.86	0.84
Linear Discriminant Analysis	0.85	0.89	0.84
Ada Boost	0.84	0.84	0.82
Support Vector Machines	0.85	0.88	0.82
Logistic Regression	0.81	0.89	0.81
Rocchio Classifier	0.81	0.82	0.81
Random Forest	0.79	0.83	0.79
K-Nearest-Neighbor	0.77	1.00	0.76
Naive Bayes (Gaussian)	0.70	0.76	0.70

References:

- Machine Learning in Action, by Peter Harrington, Manning Publications, 2012. https://www.manning.com/books/machine-learning-in-action
- Kaggle: https://www.kaggle.com/roshansharma/insurance-claim
- Wikipedia: https://www.wikipedia.org/
- Towards Data Science: https://towardsdatascience.com/
- https://scikit-learn.org/
- Python for Data Analysis Book: https://wesmckinney.com/pages/book.html
- KDnuggets: https://www.kdnuggets.com/