Problem Statement:

Every year there is various insurance fraud take place and it 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 the Fraud problem.

 In this project, we will be using the dataset of insurance policy which has the details of the customer, along with the details of the accident based on which the claims have been made. an example, will be considered with few auto insurance data to demonstrate how we can create a predictive model that predicts if an insurance claim is fraudulent or not.

We begin our analysis by importing the 2 important libraries which will be used for computing numerical problems and creating Data Frame

Pandas makes importing and analyzing the dataset much easier, it provides various filtering & transformations as provided by the excel

Numpy is that Computing in python. It provides various tools for mathematical computation. Some of them are np.array; array.shape; np.append etc

Other libraries are

1.Seaborn

This library is mostly used for creating graphical visual for analysis purpose. It helps one explore and understand your data

2.Scikit-learn :

**Scikit-learn** has a wide range of supervised and unsupervised learning algorithms which are used for various data analytics and data mining. The machine learning algorithms used in this projects are linear\_model, StandardScaler , LabelEncoder , LogisticRegression

**DATA ANALYSIS PROCESS**

We use import function to get the python libraries. We have imported pandas & numpy for basic computation and making the data frame, sklearn for machine learning , matplotlip for creating the graphs and warnings , which is used to show warning messages , we have filtered these messages using “ignore”

“import pandas as pd

from matplotlib import pyplot as plt

import seaborn as sns

from sklearn.ensemble import ExtraTreesRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

import numpy as np

import sklearn.metrics

from pylab import rcParams

%matplotlib inline

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.max\_rows', 500)”

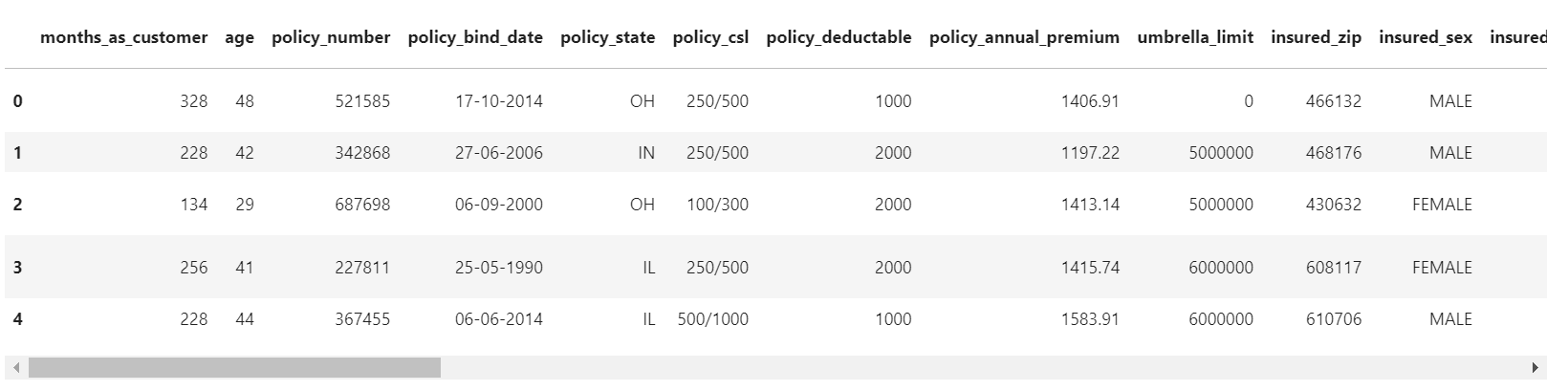
next we will be extracting the data into the program using pandas,

df=pd.read\_csv(r'https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile\_insurance\_fraud.csv')

df.head()

result of the above code: -

|  |  |
| --- | --- |
| Columns in the policy datasets | contant type |
| months\_as\_customer | int64 |
| age | int64 |
| policy\_number | int64 |
| policy\_bind\_date | object |
| policy\_state | object |
| policy\_csl | object |
| policy\_deductable | int64 |
| policy\_annual\_premium | float64 |
| umbrella\_limit | int64 |
| insured\_zip | int64 |
| insured\_sex | object |
| insured\_education\_level | object |
| insured\_occupation | object |
| insured\_hobbies | object |
| insured\_relationship | object |
| capital-gains | int64 |
| capital-loss | int64 |
| incident\_date | object |
| incident\_type | object |
| collision\_type | object |
| incident\_severity | object |
| authorities\_contacted | object |
| incident\_state | object |
| incident\_city | object |
| incident\_location | object |
| incident\_hour\_of\_the\_day | int64 |
| number\_of\_vehicles\_involved | int64 |
| property\_damage | object |
| bodily\_injuries | int64 |
| witnesses | int64 |
| police\_report\_available | object |
| total\_claim\_amount | int64 |
| injury\_claim | int64 |
| property\_claim | int64 |
| vehicle\_claim | int64 |
| auto\_make | object |
| auto\_model | object |
| auto\_year | int64 |
| fraud\_reported | object |
| \_c39 | float64 |



Columns considered for the analysis are as below;

'months\_as\_customer', 'age', 'policy\_number', 'policy\_bind\_date', 'policy\_state', 'policy\_csl', 'policy\_deductable', 'policy\_annual\_premium', 'umbrella\_limit', 'insured\_zip', 'insured\_sex', 'insured\_education\_level', 'insured\_occupation', 'insured\_hobbies', 'insured\_relationship', 'capital-gains', 'capital-loss', 'incident\_date', 'incident\_type', 'collision\_type', 'incident\_severity', 'authorities\_contacted', 'incident\_state', 'incident\_city', 'incident\_location', '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'.

Next we need to get the size of the dataset by using df.shape , and on running the program code we identified the dataset considered consists of 1000 rows and 40 columns

Next we have to identify the each column data contents using the df.nunique(), once the code is run we have below result;

|  |  |
| --- | --- |
| Column | unique data count |
| months\_as\_customer | 391 |
| age | 46 |
| policy\_number | 1000 |
| policy\_bind\_date | 951 |
| policy\_state | 3 |
| policy\_csl | 3 |
| policy\_deductable | 3 |
| policy\_annual\_premium | 991 |
| umbrella\_limit | 11 |
| insured\_zip | 995 |
| insured\_sex | 2 |
| insured\_education\_level | 7 |
| insured\_occupation | 14 |
| insured\_hobbies | 20 |
| insured\_relationship | 6 |
| capital-gains | 338 |
| capital-loss | 354 |
| incident\_date | 60 |
| incident\_type | 4 |
| collision\_type | 4 |
| incident\_severity | 4 |
| authorities\_contacted | 5 |
| incident\_state | 7 |
| incident\_city | 7 |
| incident\_location | 1000 |
| incident\_hour\_of\_the\_day | 24 |
| number\_of\_vehicles\_involved | 4 |
| property\_damage | 3 |
| bodily\_injuries | 3 |
| witnesses | 4 |
| police\_report\_available | 3 |
| total\_claim\_amount | 763 |
| injury\_claim | 638 |
| property\_claim | 626 |
| vehicle\_claim | 726 |
| auto\_make | 14 |
| auto\_model | 39 |
| auto\_year | 21 |
| fraud\_reported | 2 |
| \_c39 | 0 |

As we see there are no null values in the dataset so can proceed with EDA.

However, in most of the cases will encounter null values. To deal with this the most common function we use to fill missing values is fillna using mean, median or mode methods whichever applicable. Before using this understanding of the data distribution is mandatory

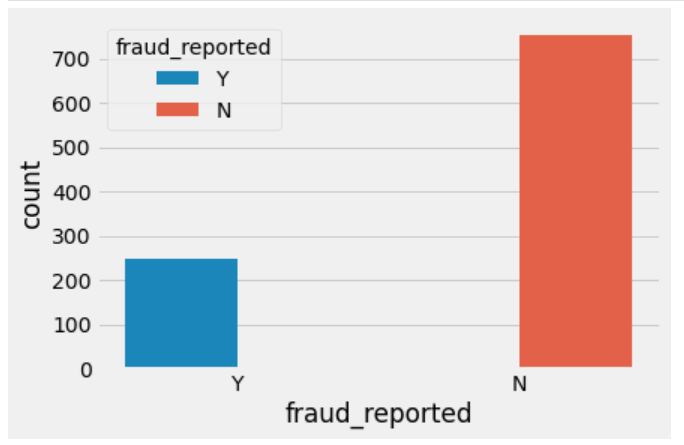
Exploratory Data Analysis

Here we use various graphs to analyze the data. This is used to communicate the information to the end user in most profound and understandable manner for decision making.

Next we have below code to plot the graph fraud reported

plt.style.use('fivethirtyeight')

ax=sns.countplot(x='fraud\_reported',data=df,hue='fraud\_reported')



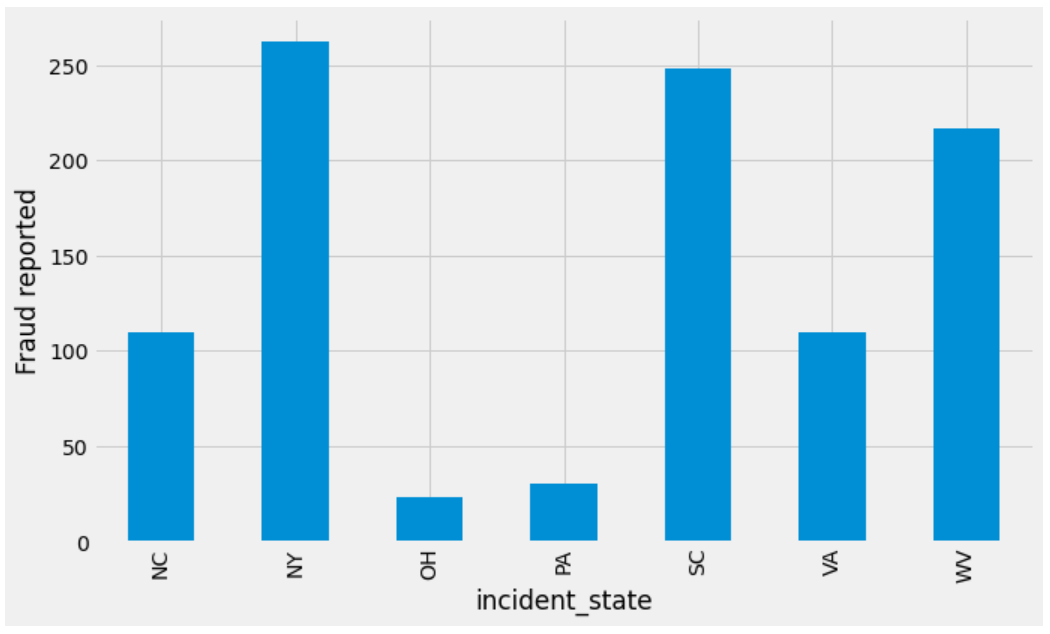
Here we see that almost 25% fraud reported. Let’s try to look for an indicative variable. Let’s analyze location. This dataset only has information from the mid-Atlantic states from the USA.

df['incident\_state'].value\_counts()

|  |  |
| --- | --- |
| NY 262 | SC 248 |
| WV 217 | VA 110 |
| NC 110 | PA 30 |
| OH 23 |  |

Name: incident\_state, dtype: int64

So on plotting the above data into graphically we identify that new York has max fraud claims



Next we will check on the age criteria that has contributed for the fraud using below code;

plt.rcParams['figure.figsize'] = [15, 8]

ax= plt.style.use('fivethirtyeight')

table=pd.crosstab(df.age, df.fraud\_reported)

table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)

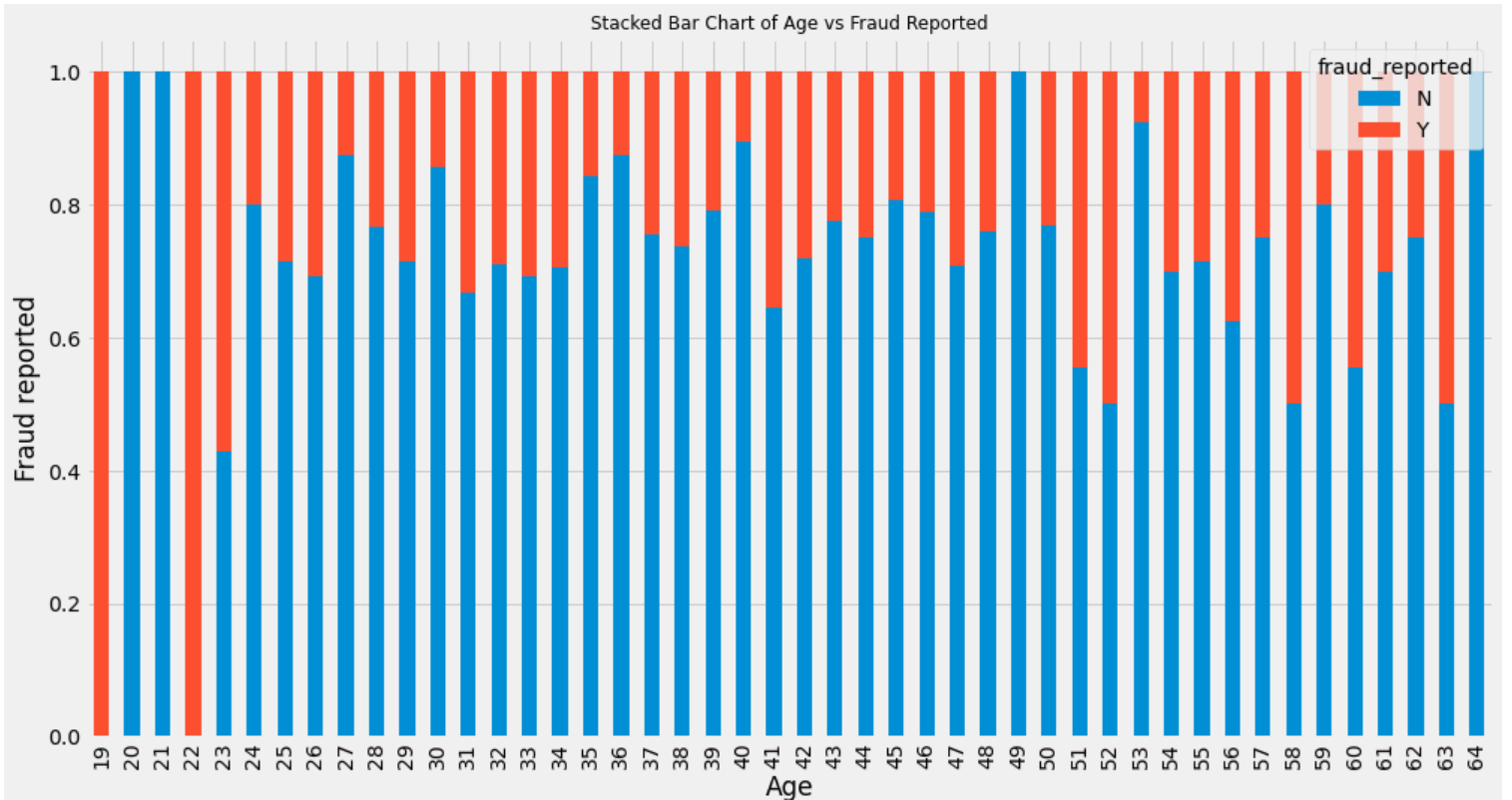
plt.title('Stacked Bar Chart of Age vs Fraud Reported', fontsize=12)

plt.xlabel('Age')

plt.ylabel('Fraud reported')

plt.show()

and running the code we get below graph



From above plot, it is obvious that, age is an important predictor for fraud reported. Age between 19-23 shows substantial number od fraud report.

Now we will check the amount that is considered for fraud and on which date of incident, using below code;

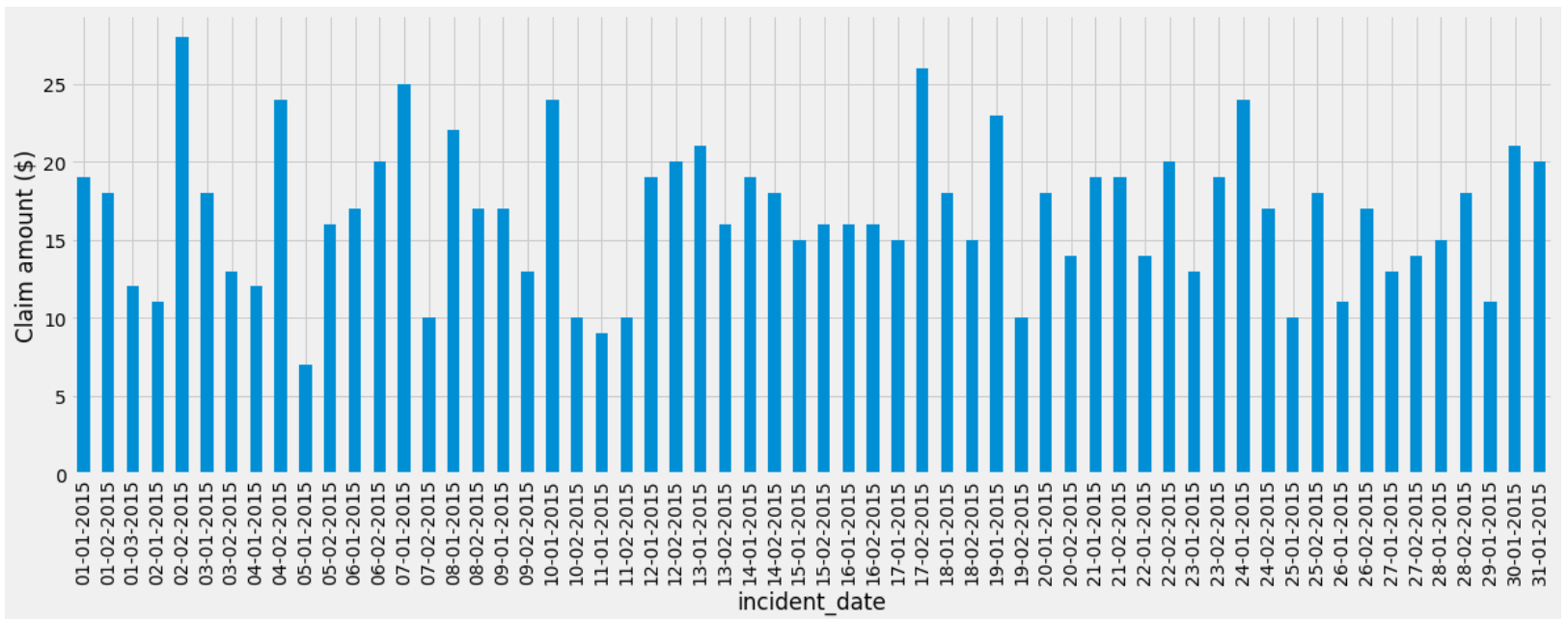
plt**.**style**.**use('fivethirtyeight')

fig **=** plt**.**figure(figsize**=**(18,6))

ax **=** df**.**groupby('incident\_date')**.**total\_claim\_amount**.**count()**.**plot**.**bar(ylim**=**0)

ax**.**set\_ylabel('Claim amount ($)')

plt**.**show()



And from the plot we see 20015 February second day is the highest with value above 25$

Next we need to check the incident type that is maximum; using below code we can get the bar chart that display top incident from the datasets;

plt**.**style**.**use('fivethirtyeight')

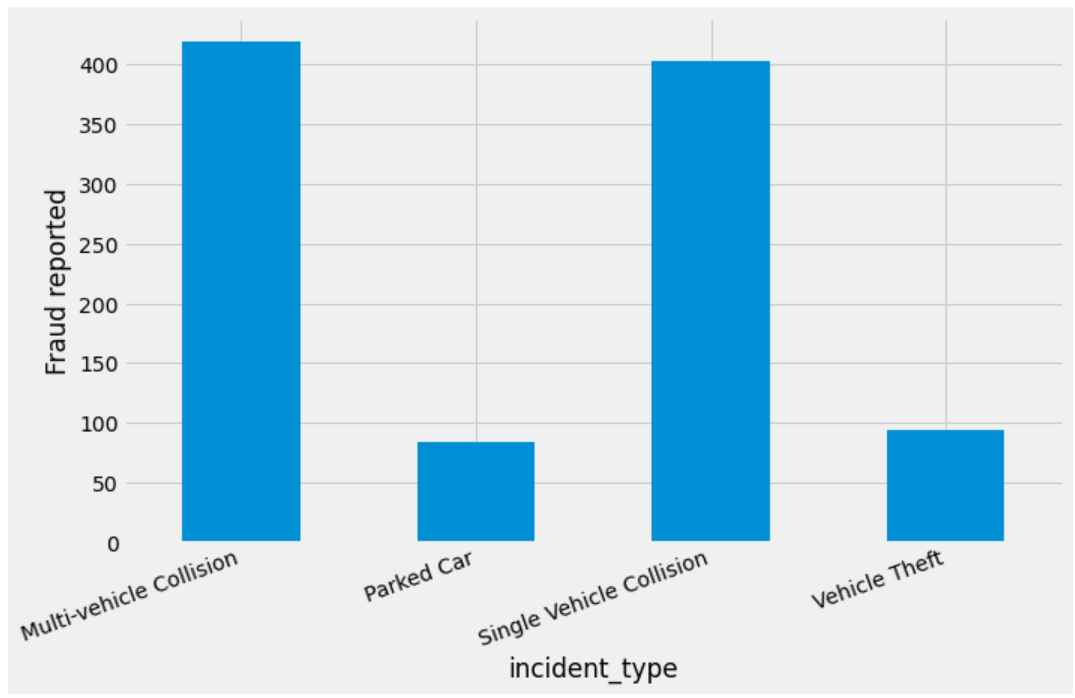
fig **=** plt**.**figure(figsize**=**(10,6))

ax **=** df**.**groupby('incident\_type')**.**fraud\_reported**.**count()**.**plot**.**bar(ylim**=**0)

ax**.**set\_xticklabels(ax**.**get\_xticklabels(), rotation**=**20, ha**=**"right")

ax**.**set\_ylabel('Fraud reported')

plt**.**show()



Now we need to check the education level grouped for fraud; below is the code for the same;

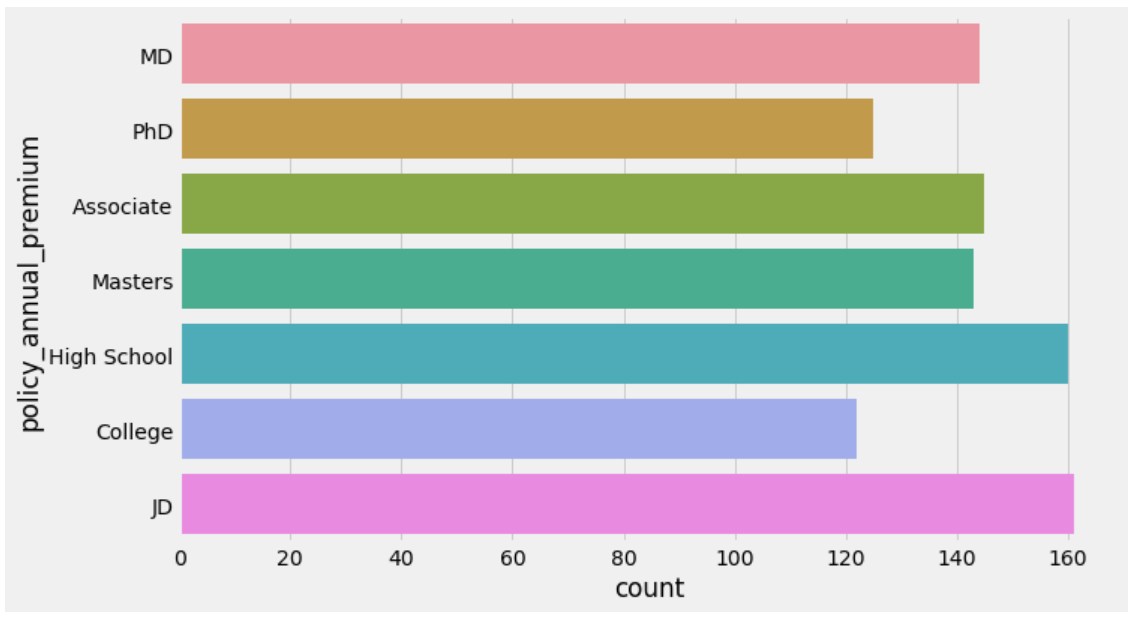
fig **=** plt**.**figure(figsize**=**(10,6))

ax **=** sns**.**countplot(y **=** 'insured\_education\_level', data**=**df)

ax**.**set\_ylabel('policy\_annual\_premium')

plt**.**show()

*# # Breakdown of Average Vehicle claim by insured's education level, grouped by fraud reported*



Similarly we have below code that explains the fraud details;

plt**.**rcParams['figure.figsize'] **=** [10, 6]

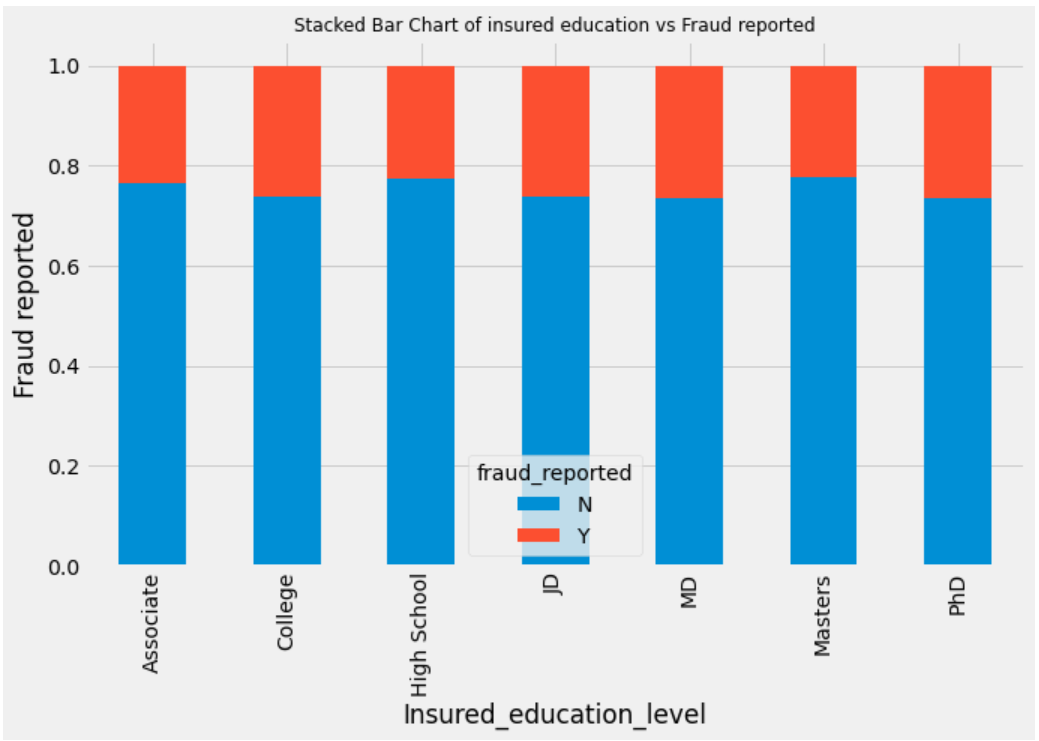
table**=**pd**.**crosstab(df**.**insured\_education\_level, df**.**fraud\_reported)

table**.**div(table**.**sum(1)**.**astype(float), axis**=**0)**.**plot(kind**=**'bar', stacked**=True**)

plt**.**title('Stacked Bar Chart of insured education vs Fraud reported', fontsize**=**12)

plt**.**xlabel('Insured\_education\_level')

plt**.**ylabel('Fraud reported')



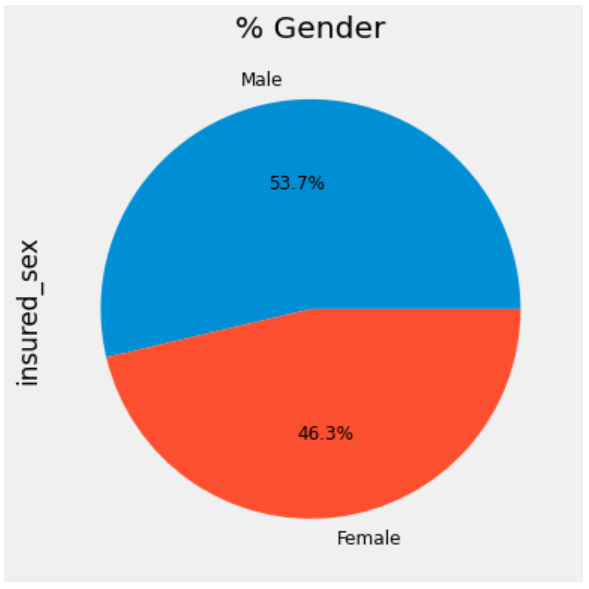
Education level involved in the Insured

ax **=** (df['insured\_sex']**.**value\_counts()**\***100.0 **/**len(df))\

**.**plot**.**pie(autopct**=**'%.1f%%', labels **=** ['Male', 'Female'], fontsize**=**12)

ax**.**set\_title('% Gender')

plt**.**show()



Gender involved most in the insured

table**=**pd**.**crosstab(df**.**insured\_sex, df**.**fraud\_reported)

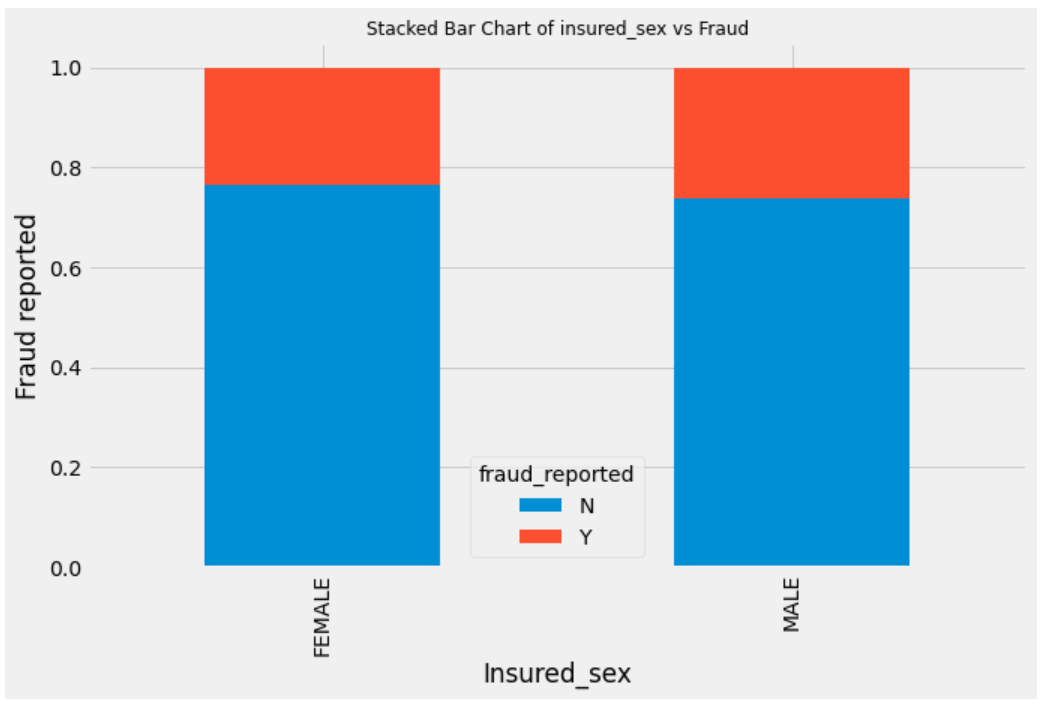
table**.**div(table**.**sum(1)**.**astype(float), axis**=**0)**.**plot(kind**=**'bar', stacked**=True**)

plt**.**title('Stacked Bar Chart of insured\_sex vs Fraud', fontsize**=**12)

plt**.**xlabel('Insured\_sex')

plt**.**ylabel('Fraud reported')

plt**.**show()



Fraud in the insurance reported details;

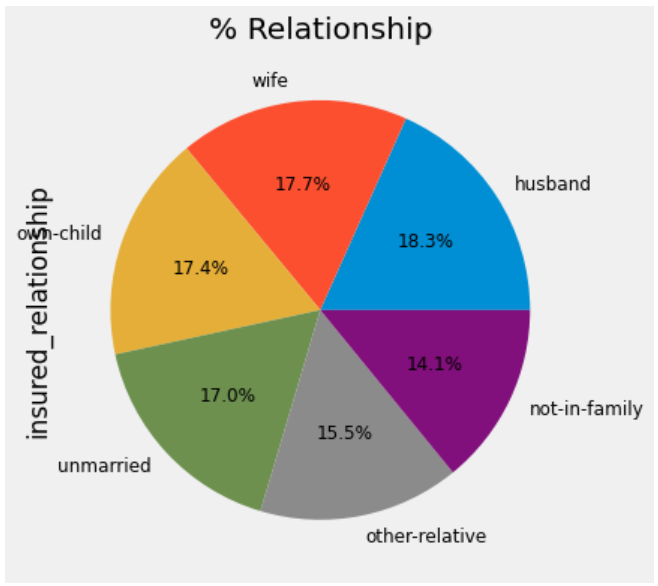
ax **=** (df['insured\_relationship']**.**value\_counts()**\***100.0 **/**len(df))\

**.**plot**.**pie(autopct**=**'%.1f%%', labels **=** ['husband', 'wife', 'own-child', 'unmarried', 'other-relative', 'not-in-family'],

fontsize**=**12)

ax**.**set\_title('% Relationship')

plt**.**show()



Relationship involved in the insurance

table**=**pd**.**crosstab(df**.**insured\_relationship, df**.**fraud\_reported)

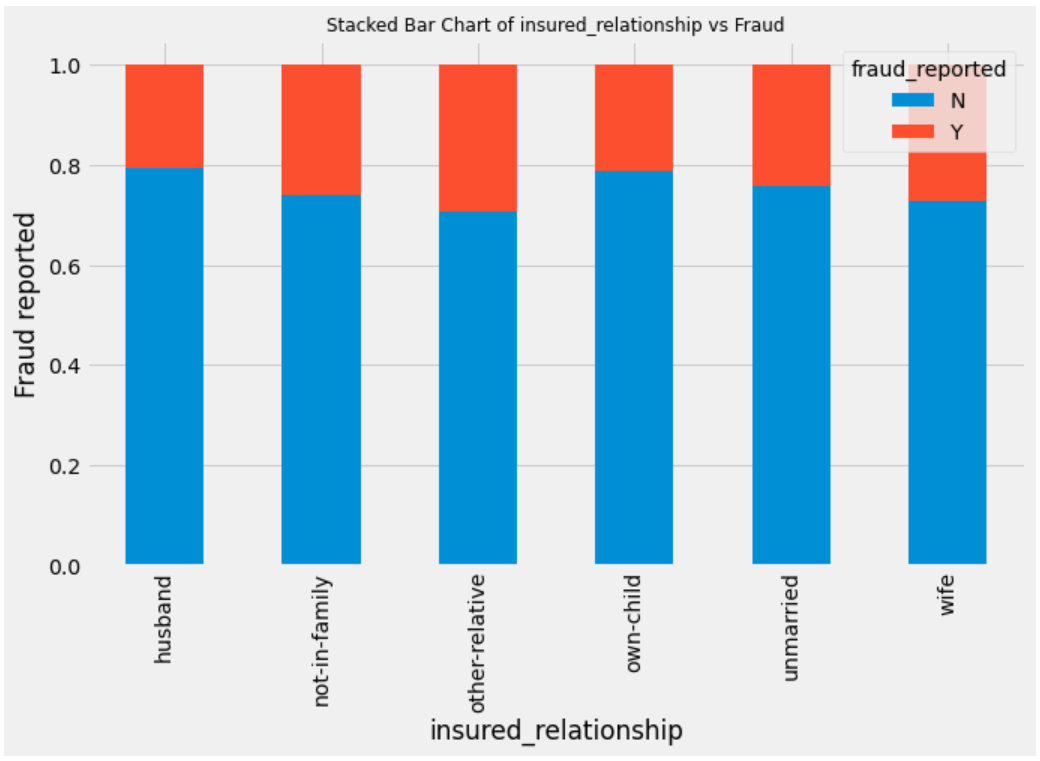
table**.**div(table**.**sum(1)**.**astype(float), axis**=**0)**.**plot(kind**=**'bar', stacked**=True**)

plt**.**title('Stacked Bar Chart of insured\_relationship vs Fraud', fontsize**=**12)

plt**.**xlabel('insured\_relationship')

plt**.**ylabel('Fraud reported')

plt**.**show()



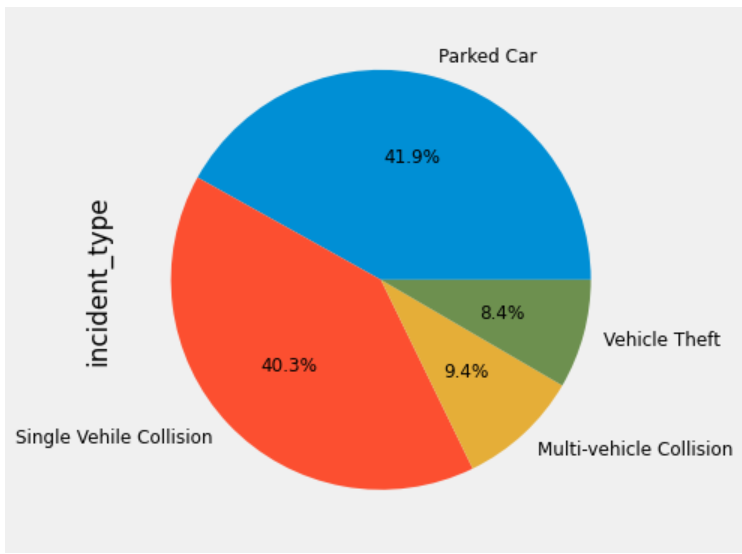
Fraud reported value by the different insured relationship

fig **=** plt**.**figure(figsize**=**(10,6))

ax **=** (df['incident\_type']**.**value\_counts()**\***100.0 **/**len(df))\

**.**plot**.**pie(autopct**=**'%.1f%%', labels **=** ['Parked Car', 'Single Vehile Collision', 'Multi-vehicle Collision', 'Vehicle Theft'],

fontsize**=**12)



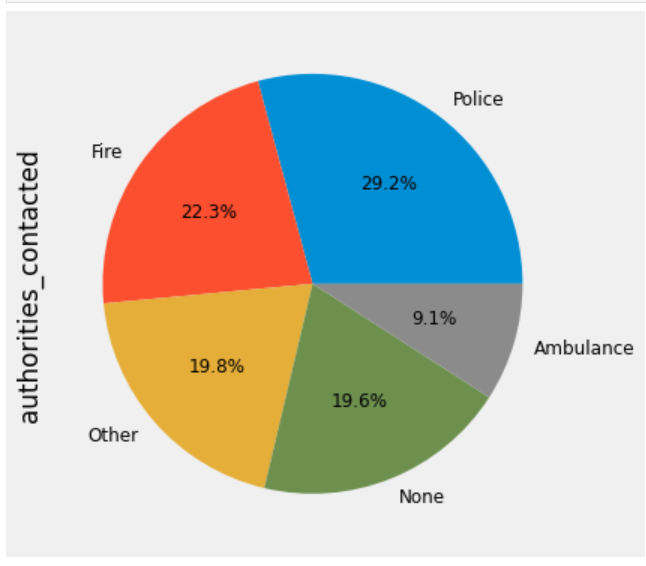
Insured incident type percentage details is shown in the above plot of Pie chart

fig **=** plt**.**figure(figsize**=**(10,6))

ax **=** (df['authorities\_contacted']**.**value\_counts()**\***100.0 **/**len(df))\

**.**plot**.**pie(autopct**=**'%.1f%%', labels **=** ['Police', 'Fire', 'Other', 'None', 'Ambulance'],

fontsize**=**12)



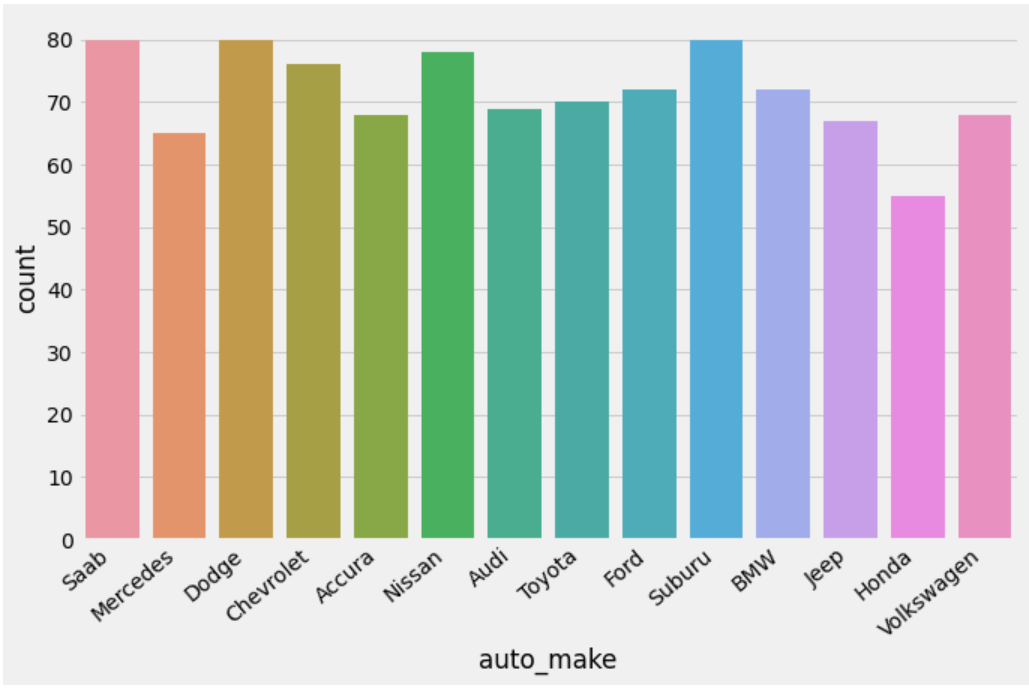
Different authorities reported percentage value at the incident is shown in the above Pie Chart.

fig **=** plt**.**figure(figsize**=**(10,6))

ax **=** sns**.**countplot(x**=**'auto\_make', data**=**df)

ax**.**set\_xticklabels(ax**.**get\_xticklabels(), rotation**=**40, ha**=**"right")

plt**.**show()



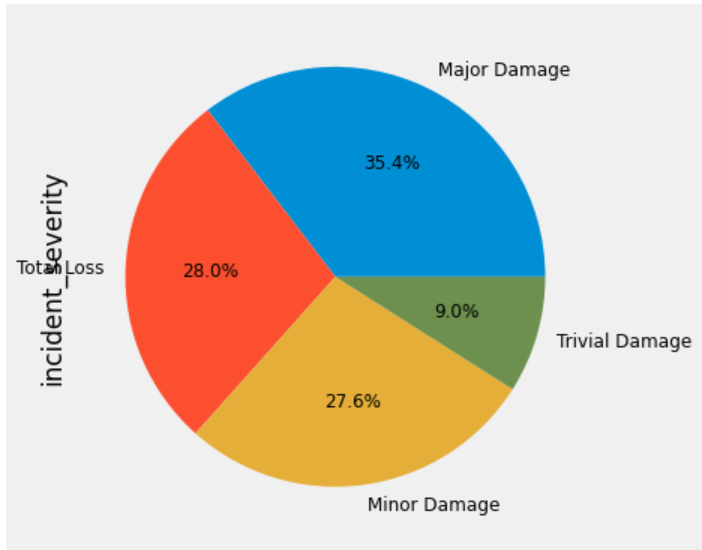
Automobile make value is shown in the above Bar chart

fig **=** plt**.**figure(figsize**=**(10,6))

ax **=** (df['incident\_severity']**.**value\_counts()**\***100.0 **/**len(df))\

**.**plot**.**pie(autopct**=**'%.1f%%', labels **=** ['Major Damage', 'Total Loss', 'Minor Damage', 'Trivial Damage'],

fontsize**=**12)



Severity of the incident reported value is explained in the above Pie Chart

plt**.**style**.**use('fivethirtyeight')

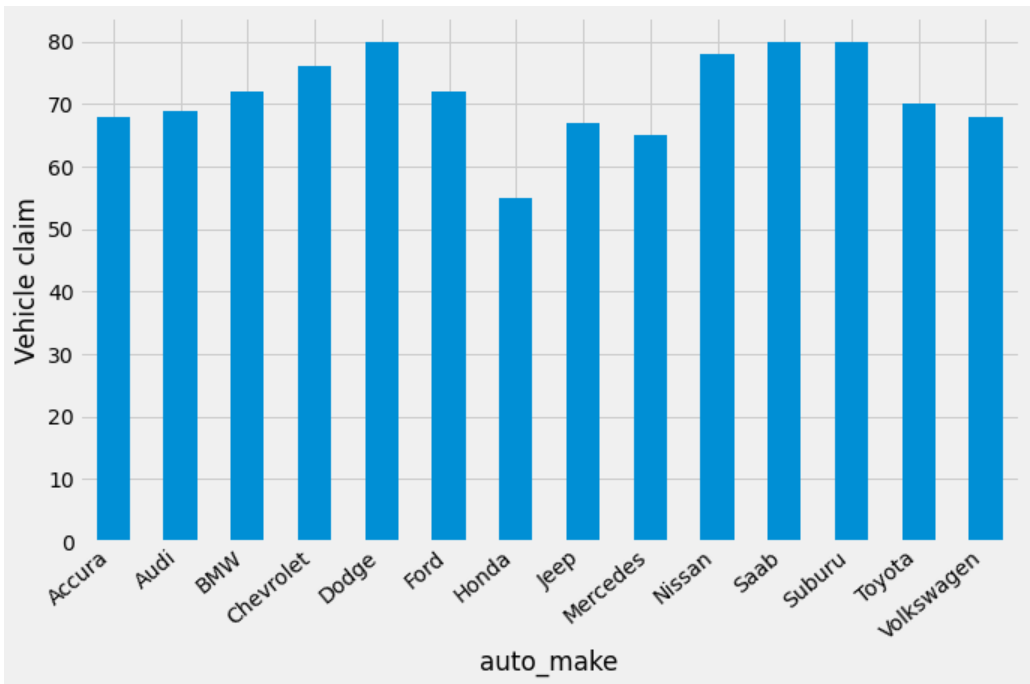
fig **=** plt**.**figure(figsize**=**(10,6))

ax**=** df**.**groupby('auto\_make')**.**vehicle\_claim**.**count()**.**plot**.**bar(ylim**=**0)

ax**.**set\_ylabel('Vehicle claim')

ax**.**set\_xticklabels(ax**.**get\_xticklabels(), rotation**=**40, ha**=**"right")

plt**.**show()



Vehicle claim vs the auto make is explained in the above bar chart.

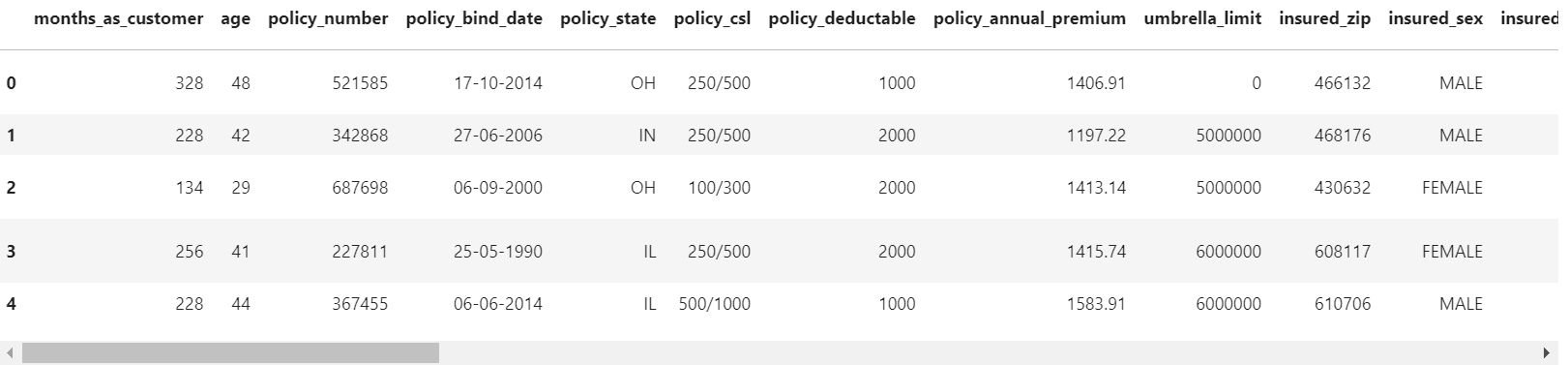
**Data Processing**

Cleaning up the data and prepare it for machine learning model.

df['fraud\_reported']**.**replace(to\_replace**=**'Y', value**=**1, inplace**=True**)

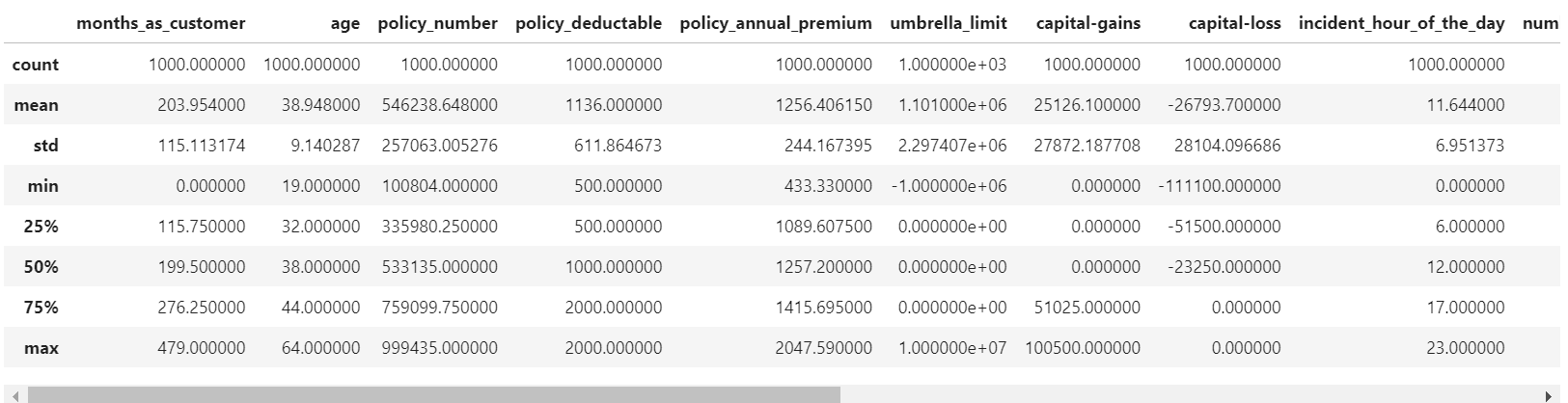
df['fraud\_reported']**.**replace(to\_replace**=**'N', value**=**0, inplace**=True**)

df**.**head()



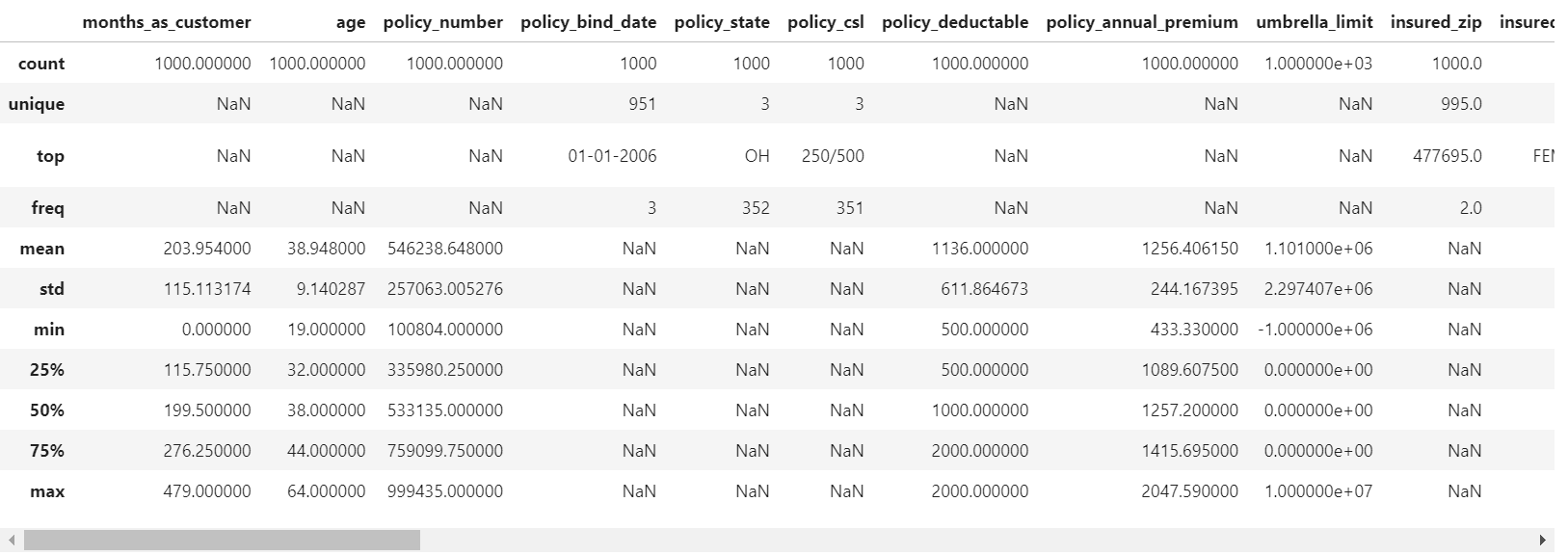
df[['insured\_zip']] **=** df[['insured\_zip']]**.**astype(object)

df**.**describe()



Some variables such as 'policy\_bind\_date', 'incident\_date',incident\_location' and 'insured\_zip' contain very high number of level. We will remove these columns for our purposes.

df.describe(include='all')



Some values in the table are shown here as “NaN”. We will need to deal with these missing values.

plt**.**style**.**use('fivethirtyeight')

fig **=** plt**.**figure(figsize**=**(10,6))

table**=**pd**.**crosstab(df**.**policy\_csl, df**.**fraud\_reported)

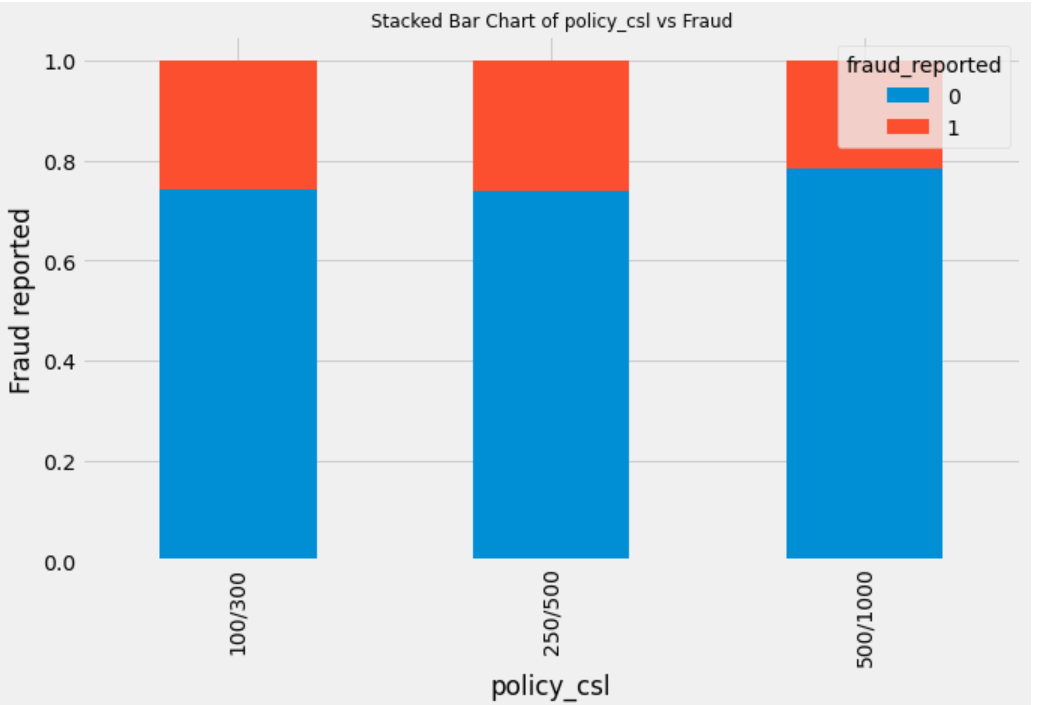
table**.**div(table**.**sum(1)**.**astype(float), axis**=**0)**.**plot(kind**=**'bar', stacked**=True**)

plt**.**title('Stacked Bar Chart of policy\_csl vs Fraud', fontsize**=**12)

plt**.**xlabel('policy\_csl')

plt**.**ylabel('Fraud reported')

plt**.**show()



Policy \_Csl vs the fraud report is shown in the above bar chart

df**.**auto\_year**.**value\_counts()

To check the spread of years to decide on further action we use above code. And we see that

auto\_year has 21 levels, and the number of records for each of the levels are quite significant considering datasize is not so large. We will do some feature engineering using this variable considering, the year of manufacturing of automobile indicates the age of the vehicle and may contain valuable information for insurance premium or fraud is concerned.

df['vehicle\_age'] **=** 2018 **-** df['auto\_year'] *# Deriving the age of the vehicle based on the year value*

df['vehicle\_age']**.**head(10)

on Using above code we will get to know the Vehicle age count.

|  |  |
| --- | --- |
| Vehicle age in years | Count |
| 0 | 14 |
| 1 | 11 |
| 2 | 11 |
| 3 | 4 |
| 4 | 9 |
| 5 | 15 |
| 6 | 6 |
| 7 | 3 |
| 8 | 6 |
| 9 | 22 |

bins **=** [**-**1, 3, 6, 9, 12, 17, 20, 24] *# Factorize according to the time period of the day.*

names **=** ["past\_midnight", "early\_morning", "morning", 'fore-noon', 'afternoon', 'evening', 'night']

df['incident\_period\_of\_day'] **=** pd**.**cut(df**.**incident\_hour\_of\_the\_day, bins, labels**=**names)**.**astype(object)

df[['incident\_hour\_of\_the\_day', 'incident\_period\_of\_day']]**.**head(20)

Now lets see incident hour and the period that has taken place by using the above code. And we have below result;

|  |  |  |
| --- | --- | --- |
| Sr.no | **incident\_hour\_of\_the\_day** | **incident\_period\_of\_day** |
| **0** | 5 | early\_morning |
| **1** | 8 | morning |
| **2** | 7 | morning |
| **3** | 5 | early\_morning |
| **4** | 20 | evening |
| **5** | 19 | evening |
| **6** | 0 | past\_midnight |
| **7** | 23 | night |
| **8** | 21 | night |
| **9** | 14 | afternoon |
| **10** | 22 | night |
| **11** | 21 | night |
| **12** | 9 | morning |
| **13** | 5 | early\_morning |
| **14** | 12 | fore-noon |
| **15** | 12 | fore-noon |
| **16** | 0 | past\_midnight |
| **17** | 9 | morning |
| **18** | 19 | evening |
| **19** | 8 | morning |

Now to Check on categorical variables we use below code

df**.**select\_dtypes(include**=**['object'])**.**columns

identify variables with '?' values

unknowns **=** {}

**for** i **in** list(df**.**columns):

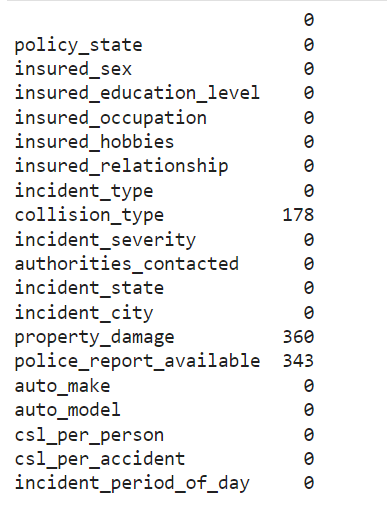
**if** (df[i])**.**dtype **==** object:

j **=** np**.**sum(df[i] **==** "?")

unknowns[i] **=** j

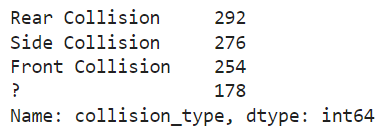
unknowns **=** pd**.**DataFrame**.**from\_dict(unknowns, orient **=** 'index')

print(unknowns)



collision\_type, property\_damage, police\_report\_available contain many missing values. So, first isolate these variables, inspect these individually for spread of category values

df**.**collision\_type**.**value\_counts()



plt**.**style**.**use('fivethirtyeight')

fig **=** plt**.**figure(figsize**=**(10,6))

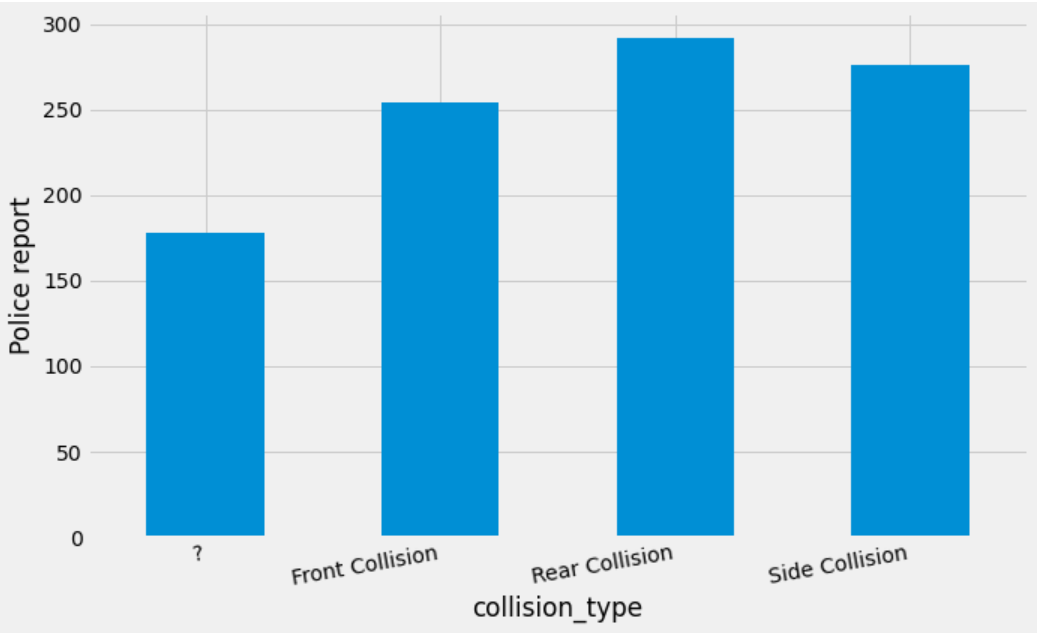
ax**=** df**.**groupby('collision\_type')**.**police\_report\_available**.**count()**.**plot**.**bar(ylim**=**0)

ax**.**set\_ylabel('Police report')

ax**.**set\_xticklabels(ax**.**get\_xticklabels(), rotation**=**10, ha**=**"right")

plt**.**show()

Now we will plot Collision report vs the Police



Now lets check the property damage against the police report;

plt**.**style**.**use('fivethirtyeight')

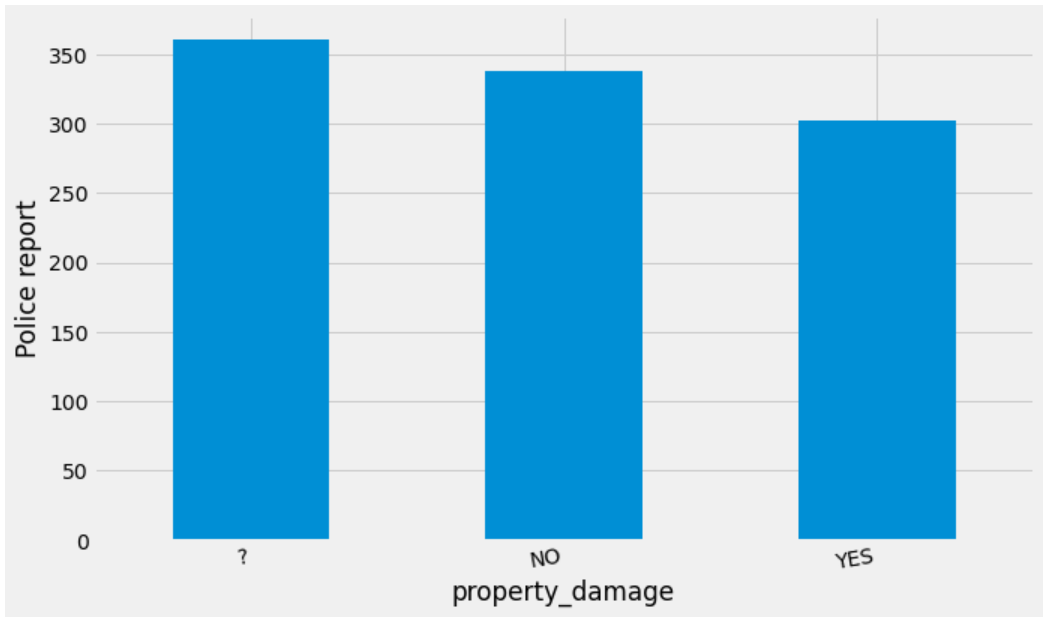
fig **=** plt**.**figure(figsize**=**(10,6))

ax**=** df**.**groupby('property\_damage')**.**police\_report\_available**.**count()**.**plot**.**bar(ylim**=**0)

ax**.**set\_ylabel('Police report')

ax**.**set\_xticklabels(ax**.**get\_xticklabels(), rotation**=**10, ha**=**"right")

plt**.**show()



### Label encoding

Label Encoding

Using Label encoding for converting categorical variable into numeric value of 0 & 1.

**from** sklearn.preprocessing **import** LabelEncoder

X['collision\_en'] **=** LabelEncoder()**.**fit\_transform(dummies['collision\_type'])

X[['collision\_type', 'collision\_en']]

After importing we need to replace, and below is the code for the same.

X['property\_damage']**.**replace(to\_replace**=**'YES', value**=**1, inplace**=True**)

X['property\_damage']**.**replace(to\_replace**=**'NO', value**=**0, inplace**=True**)

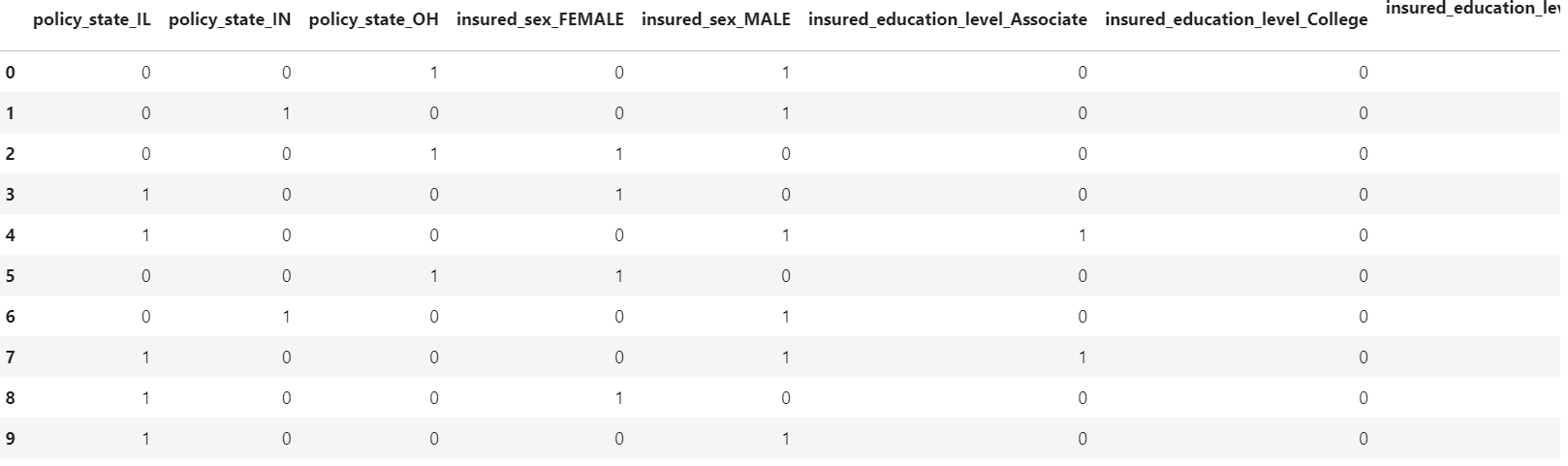
X['property\_damage']**.**replace(to\_replace**=**'?', value**=**0, inplace**=True**)

X['police\_report\_available']**.**replace(to\_replace**=**'YES', value**=**1, inplace**=True**)

X['police\_report\_available']**.**replace(to\_replace**=**'NO', value**=**0, inplace**=True**)

X['police\_report\_available']**.**replace(to\_replace**=**'?', value**=**0, inplace**=True**)

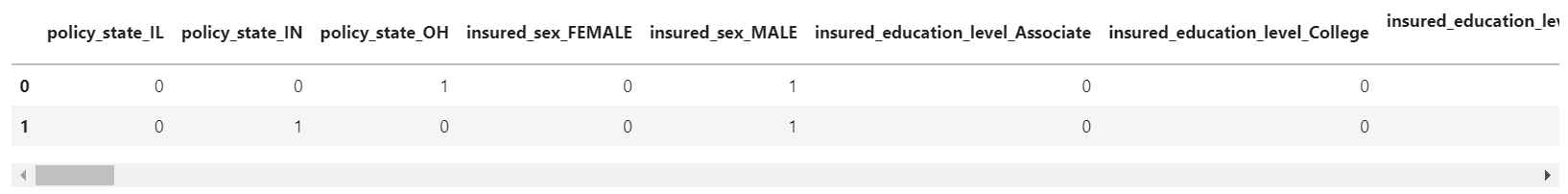
X**.**head(10)



Next we need to drop the column with collision type

X **=** X**.**drop(columns **=** ['collision\_type'])

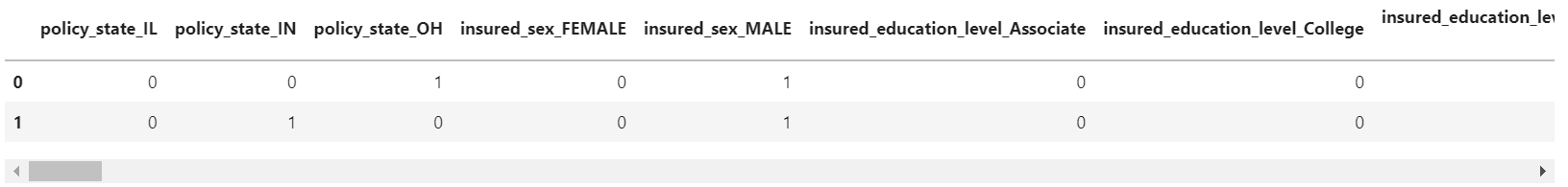
X**.**head(2)



To join the numeric columns we have below code;

X **=** pd**.**concat([X, df**.**\_get\_numeric\_data()], axis**=**1) *# joining numeric columns*

X**.**head(2)



### We now have a dataset that we could use to evaluate an algorithm sensitive to missing values like LDA

**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis

**from** sklearn.model\_selection **import** KFold

**from** sklearn.model\_selection **import** cross\_val\_score

Random Forest Classification

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.metrics **import** accuracy\_score, recall\_score, classification\_report, cohen\_kappa\_score

**from** sklearn **import** metrics

**from** sklearn.metrics **import** accuracy\_score,confusion\_matrix,classification\_report

*# Baseline Random forest based Model*

rfc **=** RandomForestClassifier(n\_estimators**=**200)

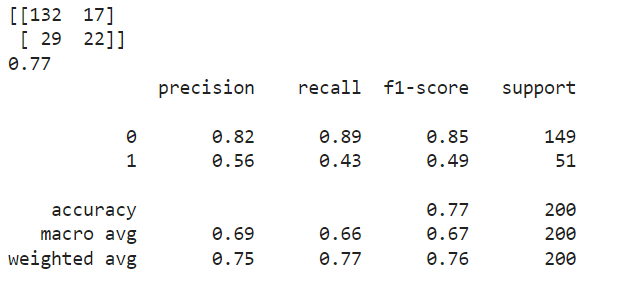
rfc**.**fit(X\_train,y\_train)

predrfc**=**rfc**.**predict(X\_test)

print(confusion\_matrix(y\_test,predrfc))

print(accuracy\_score(y\_test,predrfc))

print(classification\_report(y\_test,predrfc))



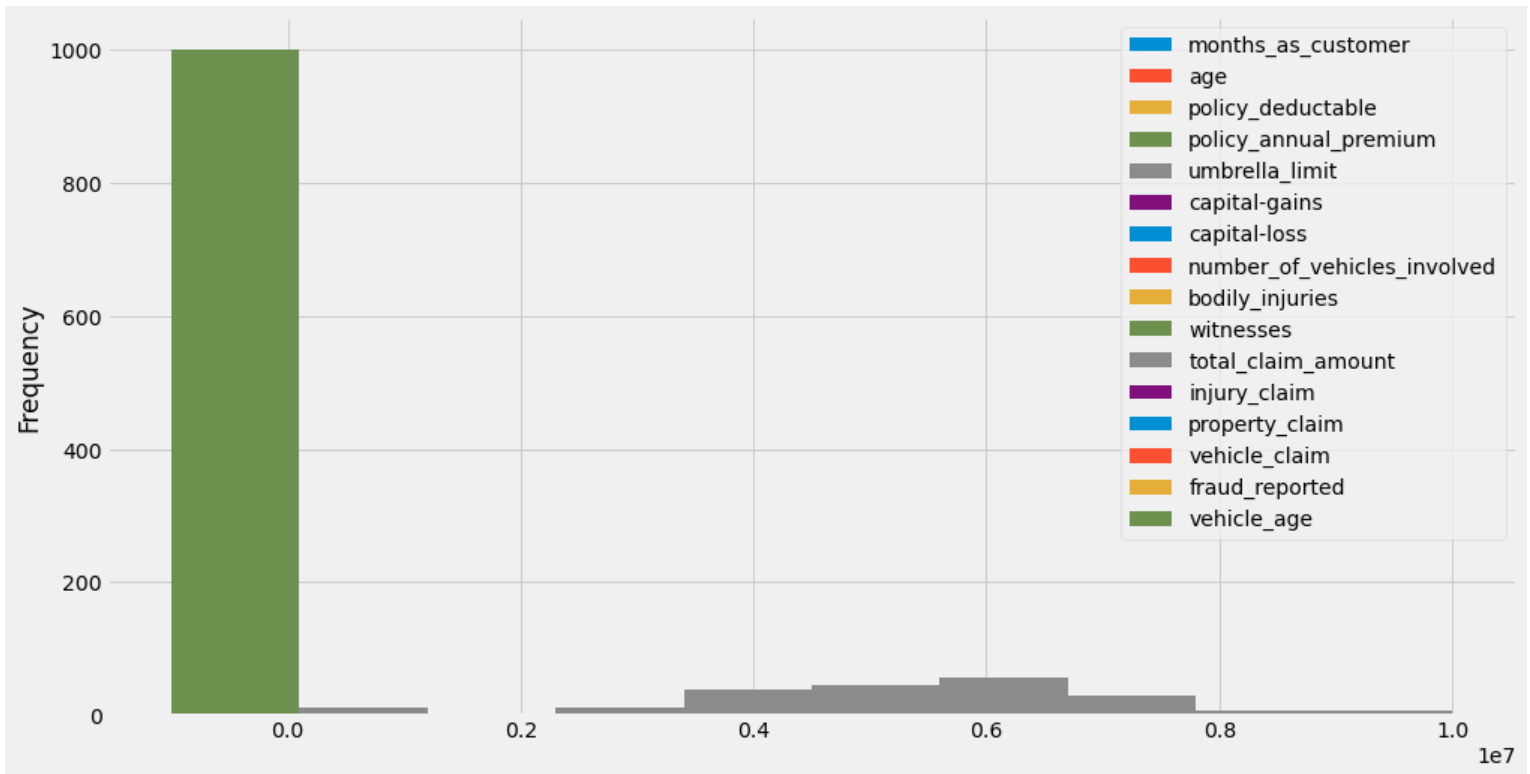
Generate a Histogram plot for anomaly detection

plt**.**style**.**use('fivethirtyeight')

plt**.**rcParams['figure.figsize'] **=** [15, 8]

df**.**plot(kind**=**'hist')

plt**.**show()



### The green bar standing tall and away from all, signifies anomalies in either of policy\_annual\_premium, witnesses or vehicle\_age. Let's draw box-and-whisker plot on each to check the presence of outliers.

plt**.**rcParams['figure.figsize'] **=** [5, 5]

sns**.**boxplot(x**=**X**.**policy\_annual\_premium)

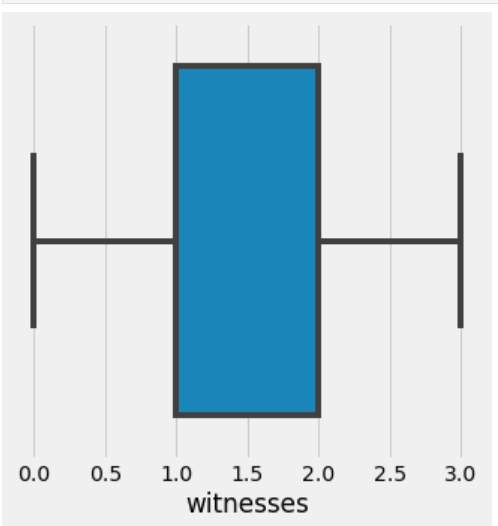
plt**.**show()

### 

Outliers are visible from the above plot from both Q1 and Q3 quartiles above the whiskers.

sns**.**boxplot(x**=**X**.**witnesses)

plt**.**show()



Standardizing the data and recheck the data distribution.

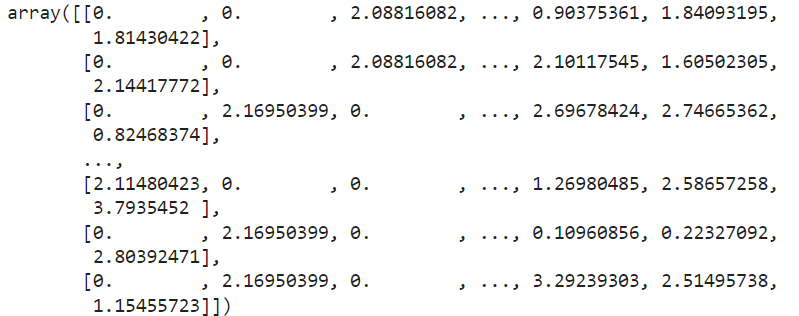
**from** sklearn.preprocessing **import** StandardScaler

scaler **=** StandardScaler(with\_mean**=False**)

X\_train\_scaled **=** scaler**.**fit\_transform(X\_train)

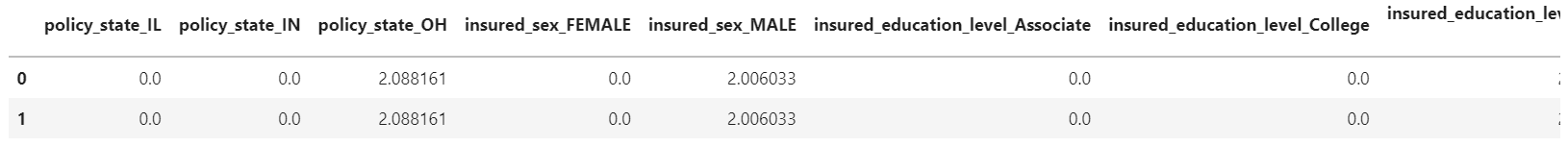
X\_test\_scaled **=** scaler**.**transform(X\_test)

X\_train\_scaled



X\_train\_scaled **=** pd**.**DataFrame(X\_train\_scaled, columns **=** X\_train**.**columns) *# retaining columns names*

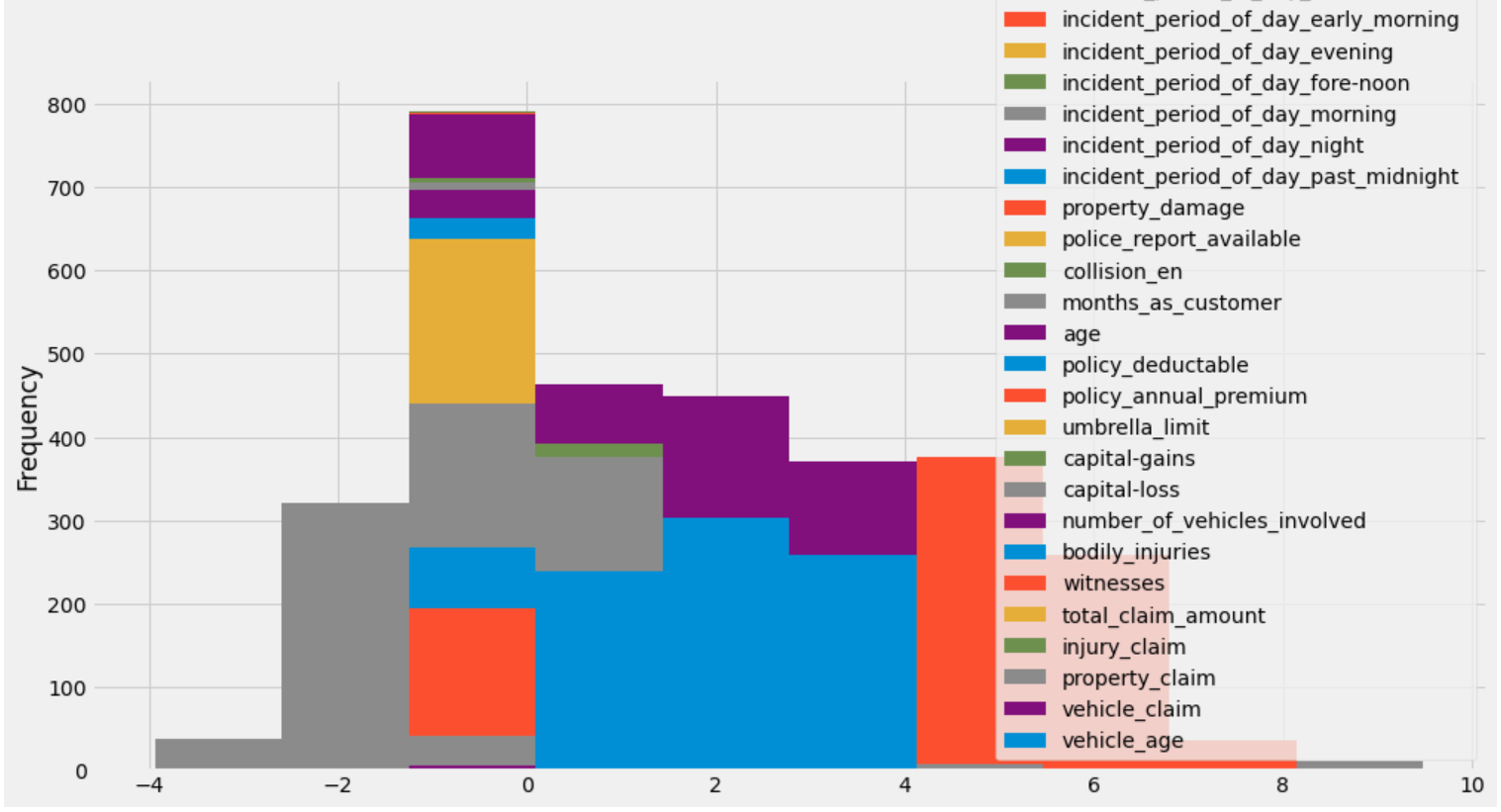
X\_train\_scaled**.**head(2)



Generate a Histogram plot on scaled data to check anomalies

plt**.**rcParams['figure.figsize'] **=** [15, 8]

X\_train\_scaled**.**plot(kind**=**'hist')



data is distributed and the anomalies are gone after standardization. The 10-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithms is evaluated in precisely the same way.

**from** sklearn.ensemble **import** AdaBoostClassifier, VotingClassifier

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn **import** model\_selection

**from** sklearn.model\_selection **import** KFold, cross\_val\_score

**from** sklearn.linear\_model **import** LogisticRegressionCV

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.svm **import** SVC

logreg**=** LogisticRegressionCV(solver**=**'lbfgs', cv**=**10)

knn **=** KNeighborsClassifier(5)

svcl **=** SVC()

adb **=** AdaBoostClassifier()

dt **=** DecisionTreeClassifier(max\_depth**=**5)

rf **=** RandomForestClassifier()

lda **=** LinearDiscriminantAnalysis()

gnb **=** GaussianNB()

Cross Validation:-

seed **=** 7

*# prepare models*

models **=** []

models**.**append(('LR', LogisticRegressionCV(solver**=**'lbfgs', max\_iter**=**5000, cv**=**10)))

models**.**append(('KNN', KNeighborsClassifier()))

models**.**append(('DT', DecisionTreeClassifier()))

models**.**append(('SVM', SVC(gamma**=**'auto')))

models**.**append(('RF', RandomForestClassifier(n\_estimators**=**200)))

models**.**append(('ADA', AdaBoostClassifier(n\_estimators**=**200)))

models**.**append(('LDA', LinearDiscriminantAnalysis()))

models**.**append(('GNB', GaussianNB()))

Model Evaluation :-

results **=** []

names **=** []

scoring **=** 'accuracy'

**for** name, model **in** models:

cv\_results **=** model\_selection**.**cross\_val\_score(model, x\_train\_scaled, y\_train, scoring**=**scoring)

results**.**append(cv\_results)

names**.**append(name)

msg **=** "%s: %f (%f)" **%** (name, cv\_results**.**mean(), cv\_results**.**std())

print(msg)

*# boxplot algorithm comparison*

plt**.**rcParams['figure.figsize'] **=** [15, 8]

fig **=** plt**.**figure()

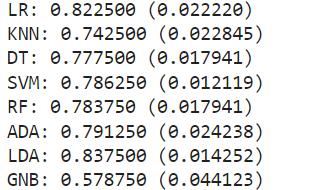
fig**.**suptitle('Algorithm Comparison')

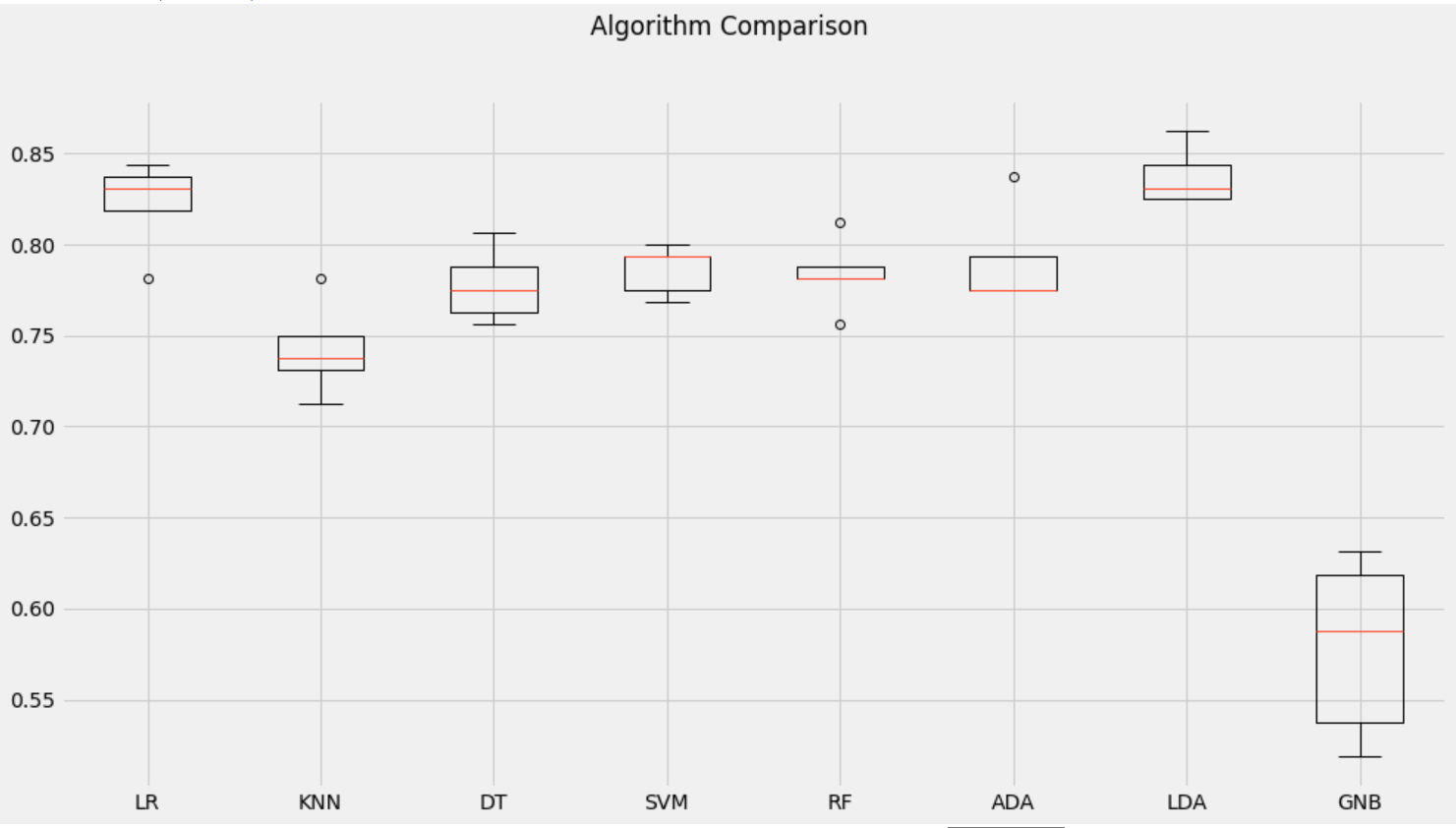
ax **=** fig**.**add\_subplot(111)

plt**.**boxplot(results)

ax**.**set\_xticklabels(names)

plt**.**show()





Above a list of each algorithm, the mean accuracy and the standard deviation accuracy and a box & whisker plot showing the spread of the accuracy scores across each cross validation fold for each algorithm. It is clear that the inear Discriminant Analysis (82%) is leading the list. Logistics regression and XGB are almost close (82.62% and 82.87% respectively). We could see some noise / outlier in data in case of XGB. The LR box-plot is skewd one side with longer tail.

clf1**=** LogisticRegressionCV(solver**=**'lbfgs', max\_iter**=**5000, cv**=**10)

clf **=** [('LR', clf1)]

checking accuracy of the model using the Logistic regression method;

eclf**=** VotingClassifier(estimators**=**[('LR', clf1)],voting**=**'hard')

**for** clf, label **in** zip([clf1],['LogisticRegression']):

scores **=** cross\_val\_score(clf, x\_train\_scaled, y\_train, cv**=**10, scoring**=**'accuracy')

print("Accuracy: %0.2f (+/- %0.2f) [%s]" **%** (scores**.**mean(), scores**.**std(), label))

The accuracy score is 83%.