# **Poonam Pandhare**

# **Nov 2, 2022**

# ****Insurance claims — Fraud detection using machine learning****

**Problem Definition:**

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

The goal of this project is to build a model that can working with some auto insurance fraud. create a predictive model that predicts if an insurance claim is fraudulent or not.

**Data analysis:**

Reading the Csv File. And setting the display option to show all columns and rows.

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

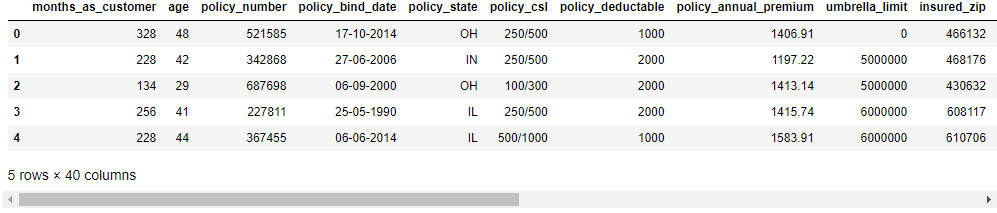
pd.set\_option("display.max\_rows",None)

pd.set\_option("display.max\_columns",None)

**Code:**

df.head()

**Output:**

****

**Code:**

df.shape

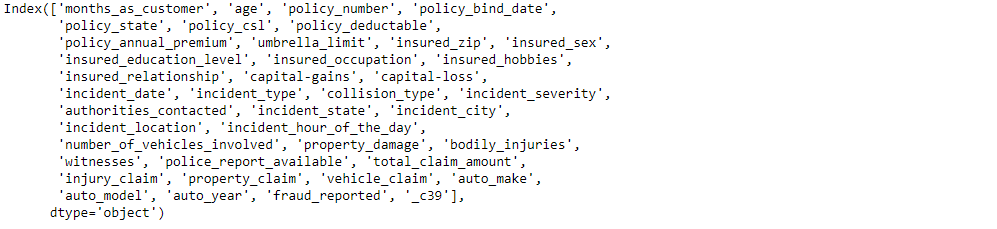
**Output:**

(1000, 40)

**Code:**

df.columns

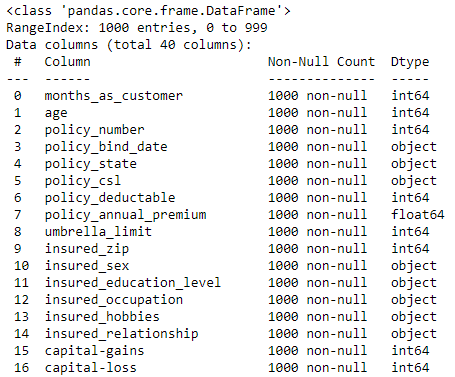
**Output:**

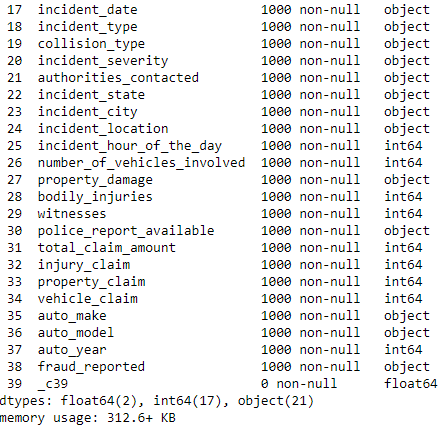
****

**Code:**

df.info()

**Output:**



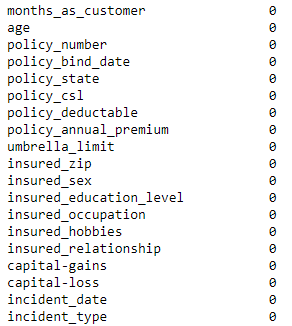


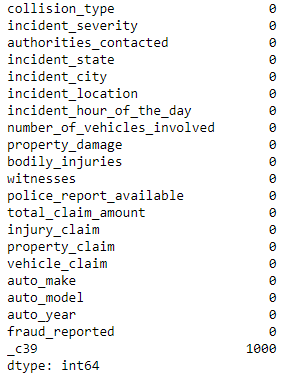
**Observation:** We have integer,float,and object data in dataset

**Code:**

df.isnull().sum()

**Output:**





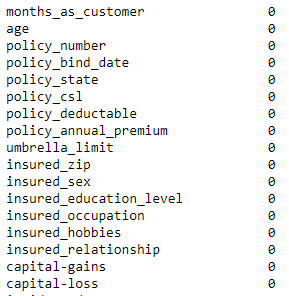
**Null value handling**

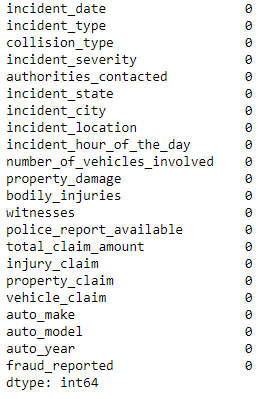
df.drop(["\_c39"],inplace=True,axis=1)

**Code:**

df.isnull().sum()

**Output:**





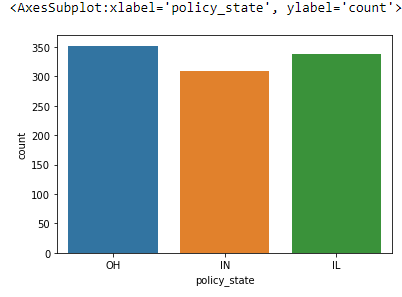
**Observation:** Null values are removed

# Exploratory data analysis (Univariate analysis)

**Code:**

sns.countplot(df ['policy\_state'])

**Output:**

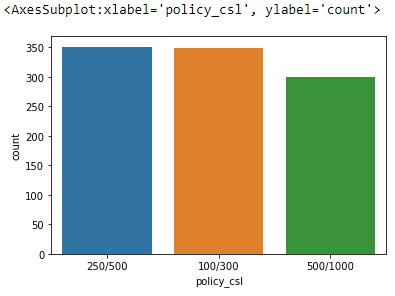
****

**Observation:** OH,IN,IL have equal amount of data

**Code:**

sns.countplot(df['policy\_csl'])

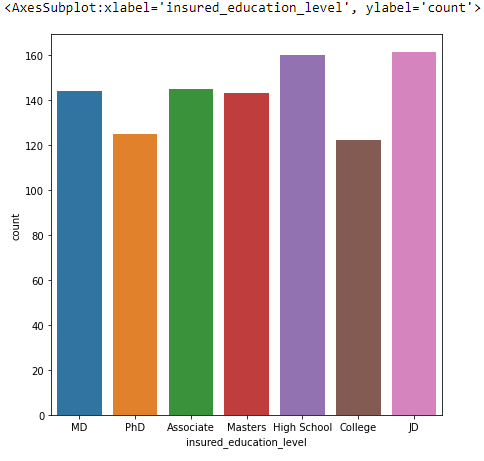
**Output:**

****

**Code:**

sns.countplot(df[ 'insured\_education\_level'])

**Output:**

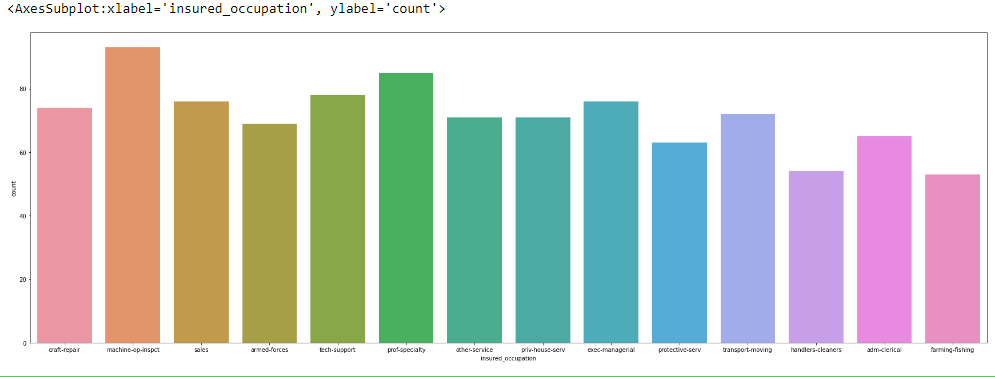
****

**Observation:** High School is higher compared to other education level

**Code:**

sns.countplot(df['insured\_occupation'])

**Output:**

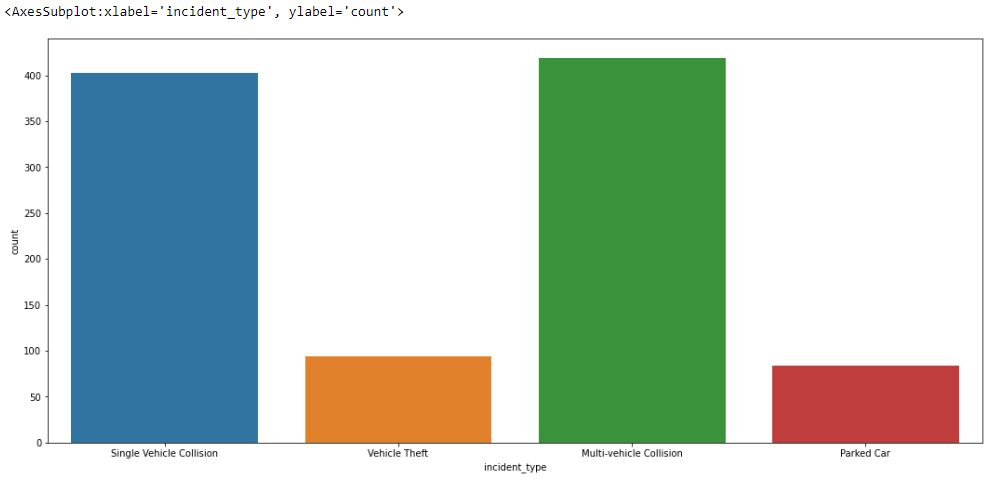
****

**Observation:** Machine-op-inspct occupation is higher compared to others

**Code:**

sns.countplot(df['incident\_type'])

**Output:**

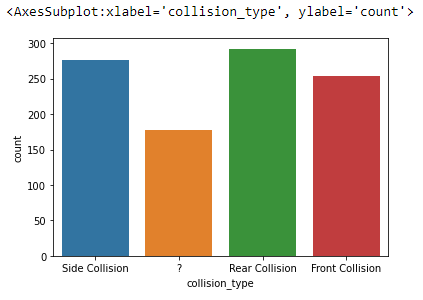
****

**Observation:** MultiVechile Collision is high compared to other incidents

**Code:**

sns.countplot(df['collision\_type'])

**Output:**

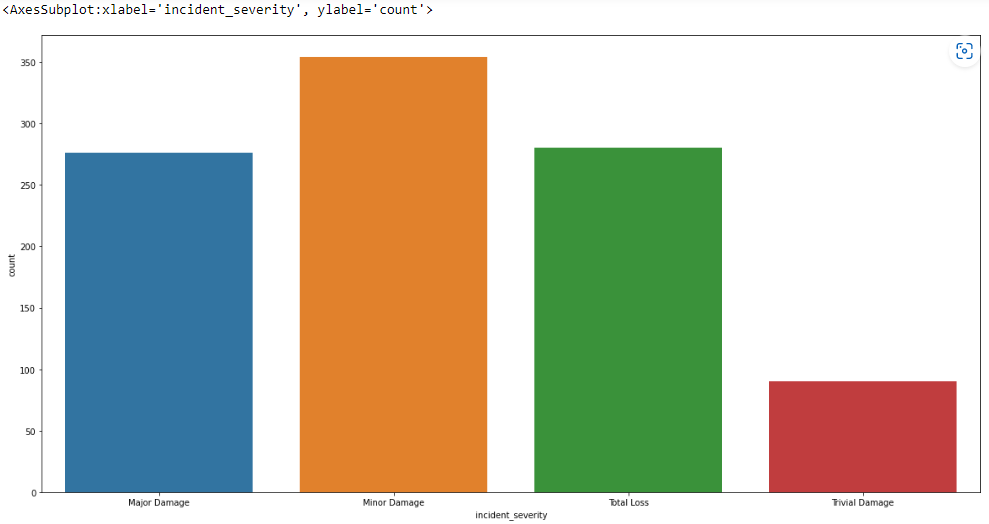
****

**Observation:** Rear Collision is high compared to other collisions

**Code:**

sns.countplot(df['incident\_severity'])

**Output:**

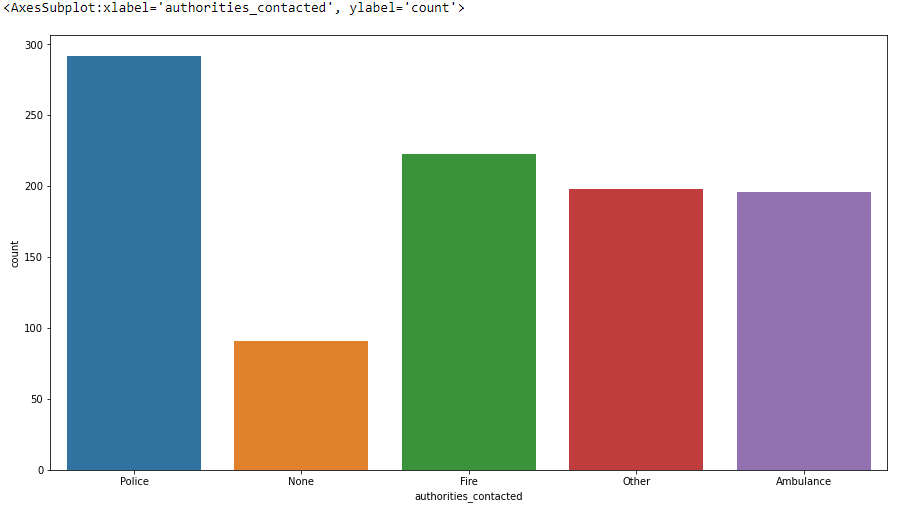
****

**Observation:** Minor Damage in incident\_severity is high copared to others

**Code:**

sns.countplot(df['authorities\_contacted'])

**Output:**

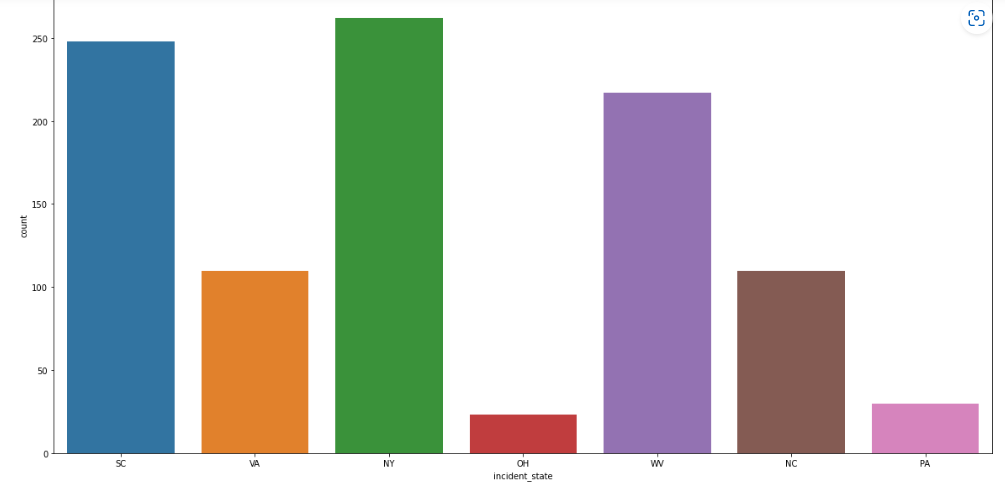
****

**Observation:** Police is contacted most in authorities contacted

**Code:**

sns.countplot(df['incident\_state'])

**Output:**

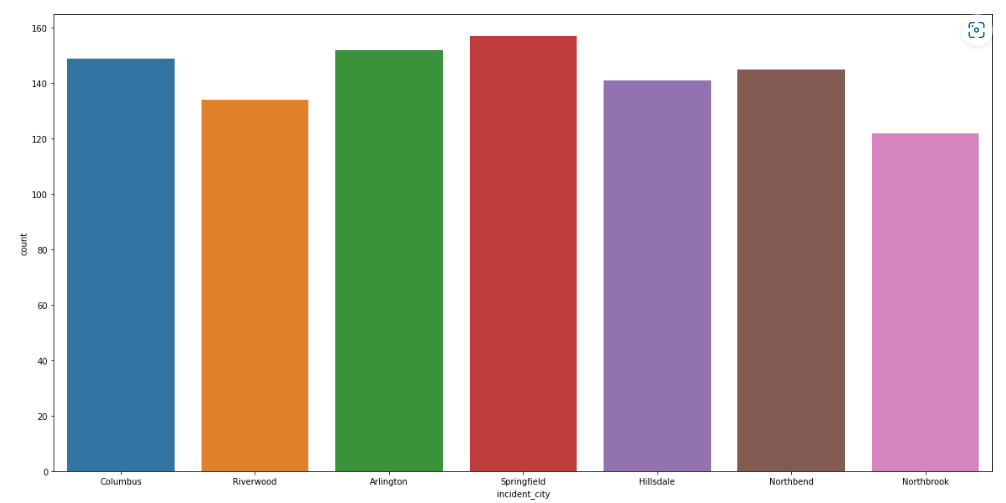
****

**Observation:** NY is higher incident state compared to others

**Code:**

sns.countplot(df['incident\_city'])

**Output:**

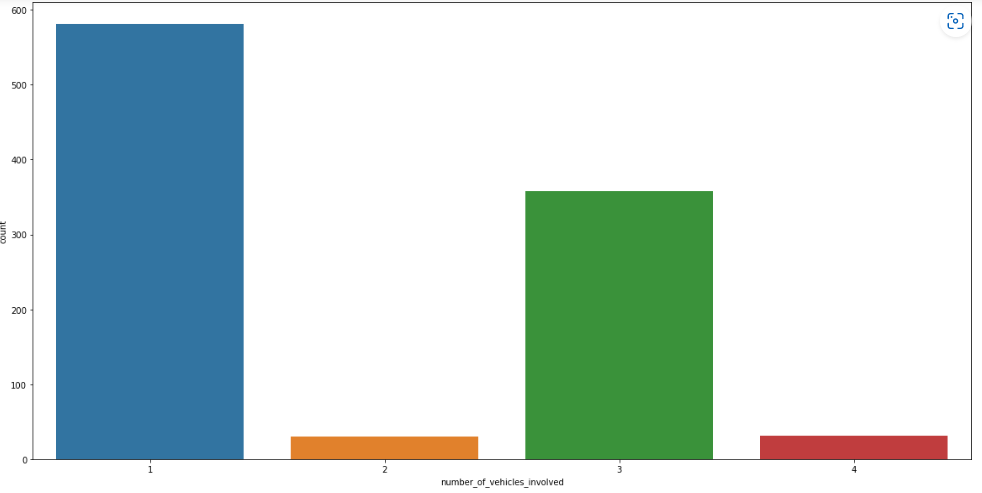


**Observation:** Springfield incident city is higher compared to other city

**Code:**

sns.countplot(df['number\_of\_vehicles\_involved'])

**Output:**

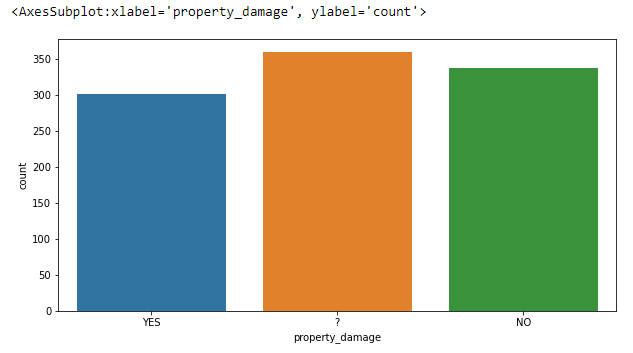


**Observation:** Vehicle involved in collision is 1 most

**Code:**

sns.countplot(df['property\_damage'])

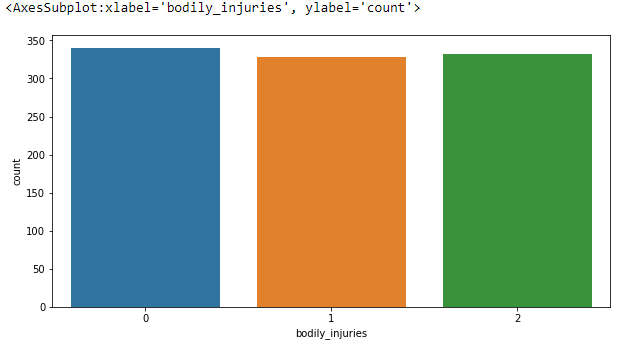
**Output:**



**Code:**

sns.countplot(df['bodily\_injuries'])

**Output:**

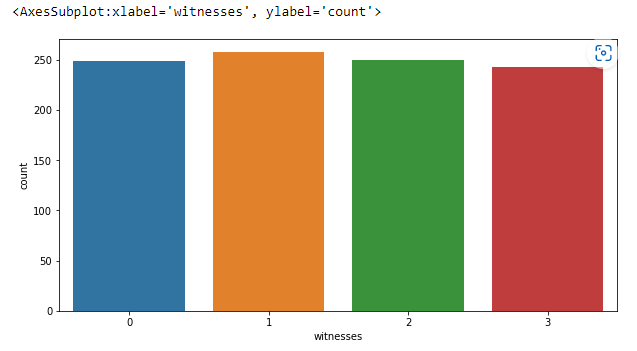


**Observation:** Body injuries is 0 most

**Code:**

sns.countplot(df['witnesses'])

**Output:**

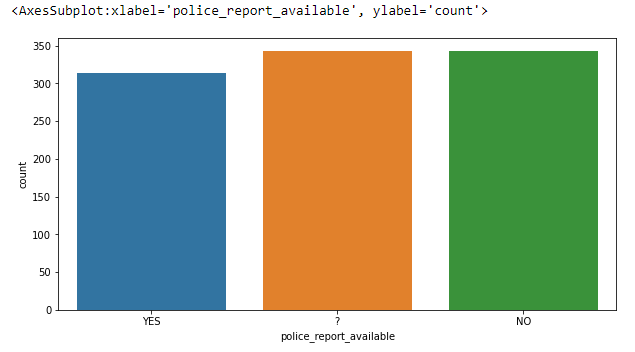


**Observation:** 'witnesses' are mostly 1

**Code:**

sns.countplot(df['police\_report\_available'])

**Output:**

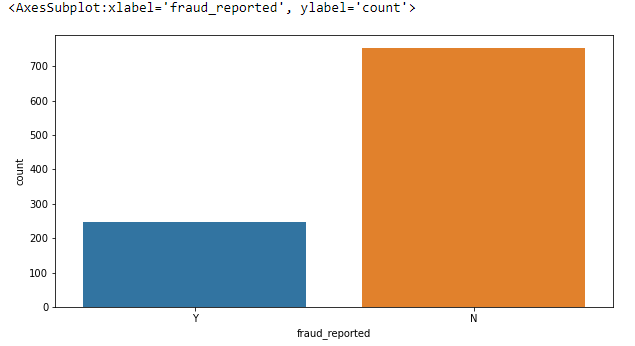


**Observation:** No police report available is no moslty

**Code:**

sns.countplot(df[ 'fraud\_reported'])

**Output:**



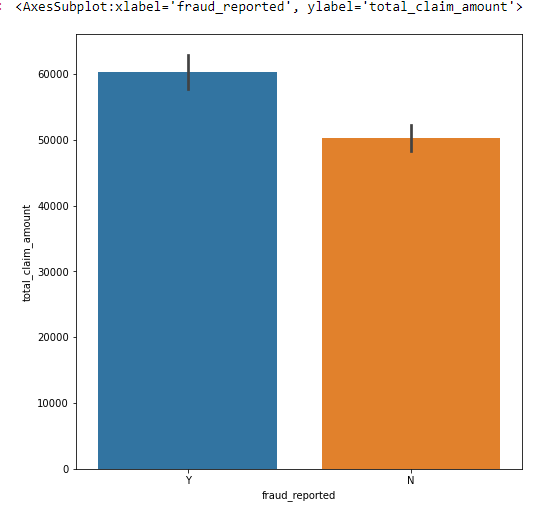
**Observation:** No fraud Reported in most of the cases

# Bivariate Analysis

**Code:**

sns.barplot(df[ 'fraud\_reported'],df[ 'total\_claim\_amount'])

**Output:**

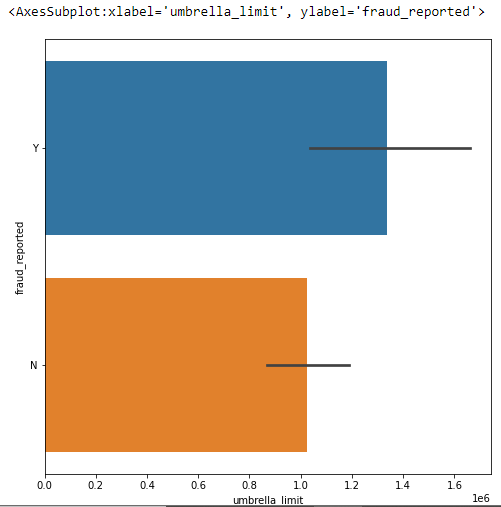
****

**Observation:** Claim amount is higher for fraud cases

**Code:**

sns.barplot(df['umbrella\_limit'],df[ 'fraud\_reported'])

**Output:**

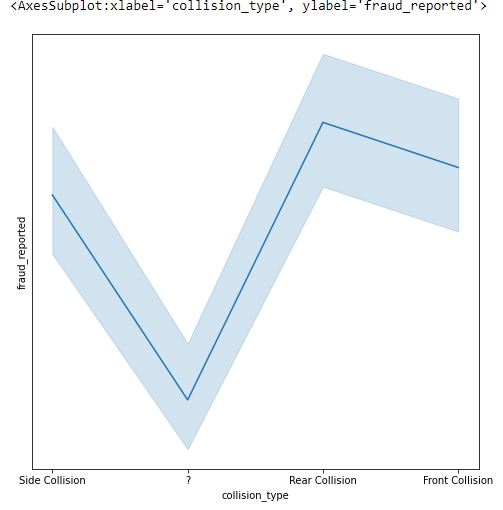
****

**Observation:** Fraud cases have higher umbrella limit

**Code:**

sns.lineplot(df[ 'collision\_type'],df[ 'fraud\_reported'])

**Output:**

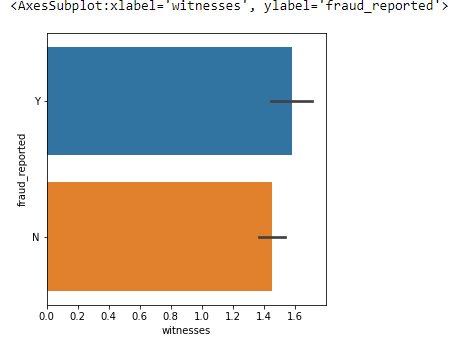
****

**Observation:** Rear collision have higher fraud claims

**Code:**

sns.barplot(df[ 'witnesses'],df[ 'fraud\_reported'])

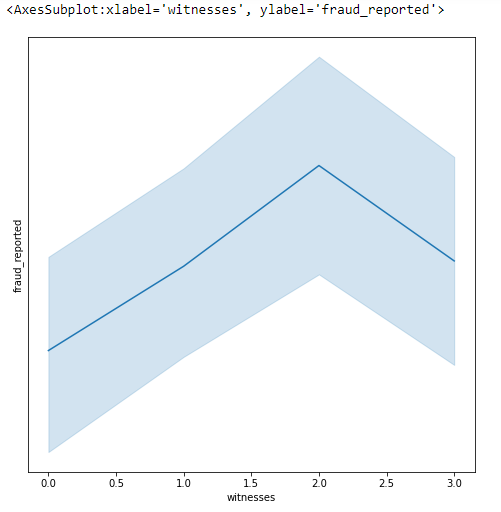
**Output:**

****

**Code:**

sns.lineplot(df[ 'witnesses'],df[ 'fraud\_reported'])

**Output:**

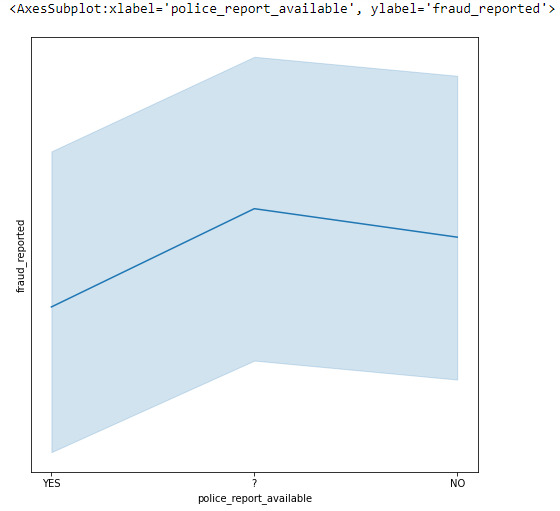
****

**Observation:** Fraud reported is higher with increase in witness

**Code:**

sns.lineplot(df['police\_report\_available'],df[ 'fraud\_reported'])

**Output:**

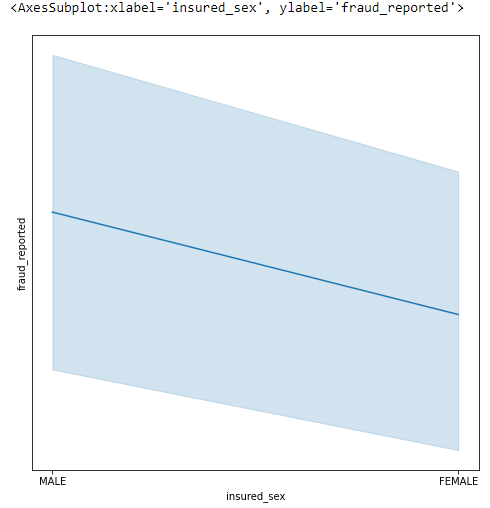
****

**Observation:** Fraud report is higher without police report

**Code:**

sns.lineplot(df['insured\_sex'],df[ 'fraud\_reported'])

**Output:**

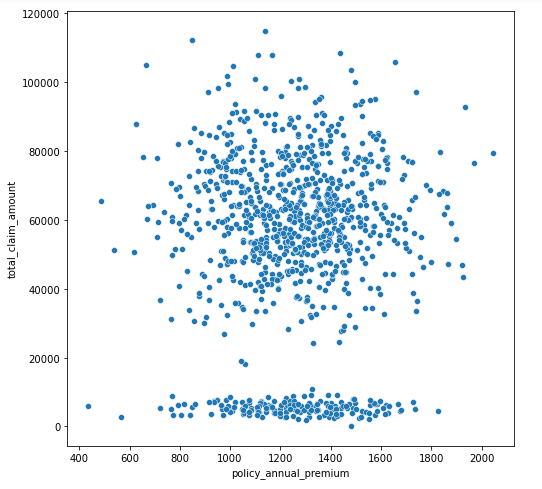
****

**Observation:** Male have higher fraud claims

**Code:**

sns.scatterplot(df['policy\_annual\_premium'],df['total\_claim\_amount'])

**Output:**

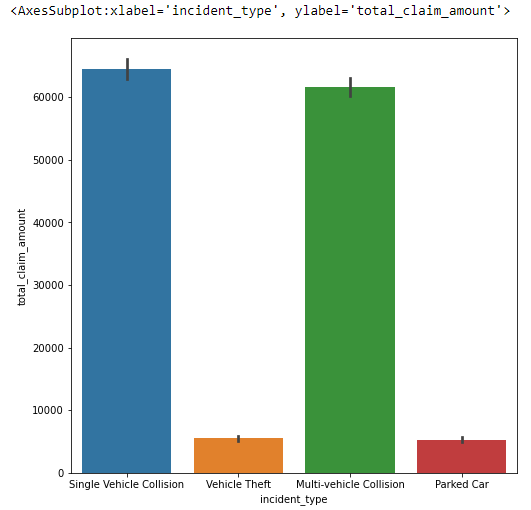
****

**Observation:** Totoal Claim amount is positively correlated with policy annual premium

**Code:**

sns.barplot(df['incident\_type'],df['total\_claim\_amount'])

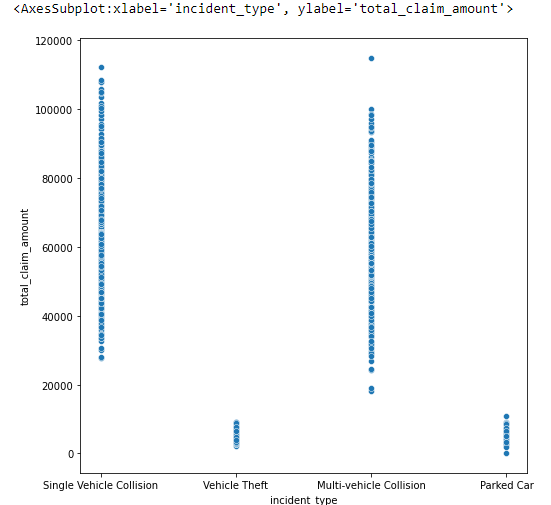
**Output:**

****

**Code:**

sns.scatterplot(df['incident\_type'],df['total\_claim\_amount'])

**Output:**

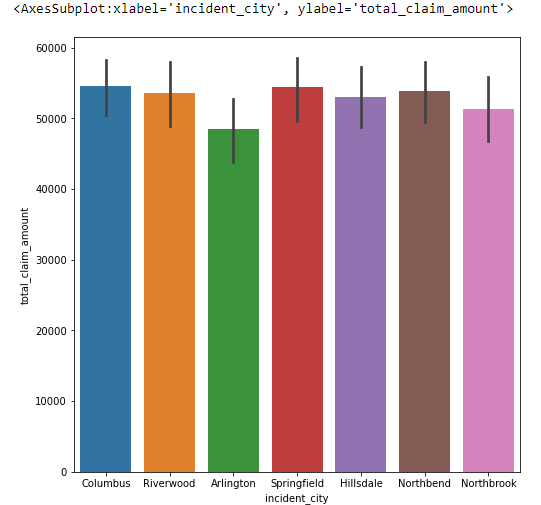
****

**Observation:** For Single vehicle collison total claim amount is higher

**Code:**

sns.barplot(df['incident\_city'],df[ 'total\_claim\_amount'])

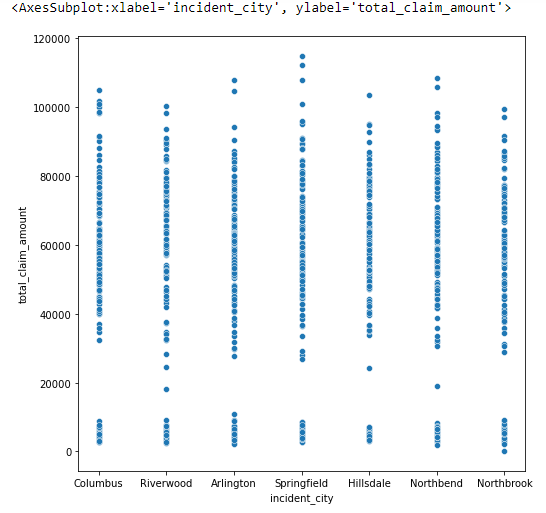
**Output:**

****

**Code:**

sns.scatterplot(df['incident\_city'],df[ 'total\_claim\_amount'])

**Output:**

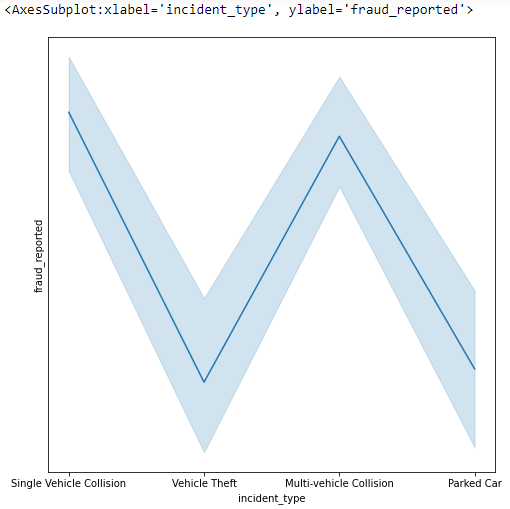
****

**Observation:** SpringFieldIncident\_city has higer claim amount

**Code:**

sns.lineplot(df[ 'incident\_type'],df[ 'fraud\_reported'])

**Output:**

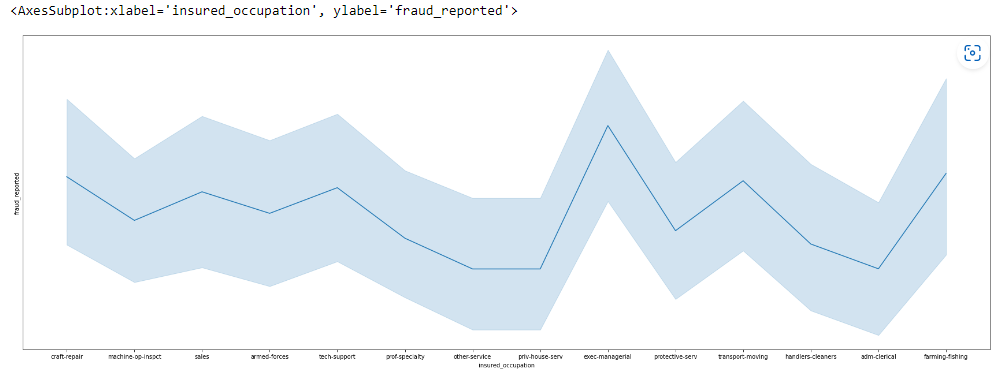
****

**Observation:** Single Vehicle Collison and Mult-Vehicle collision has higher fraud reported

**Code:**

sns.lineplot(df['insured\_occupation'],df['fraud\_reported'])

**Output:**

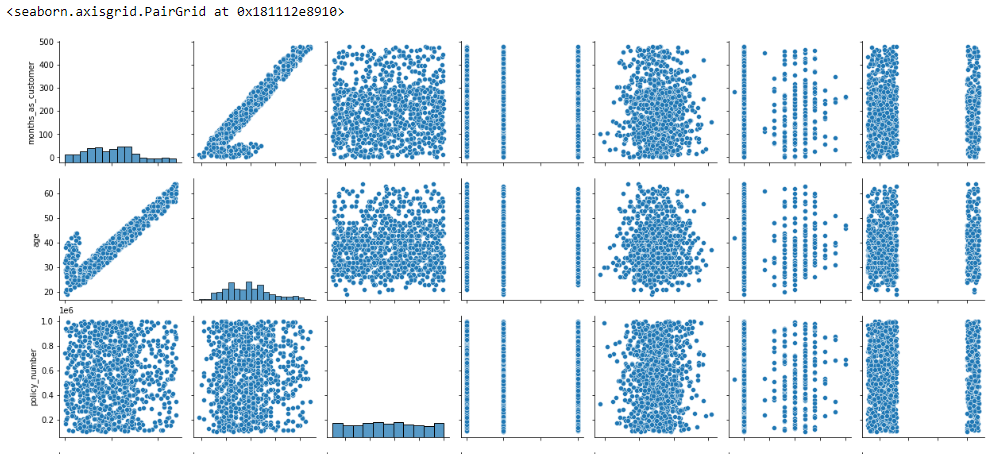
****

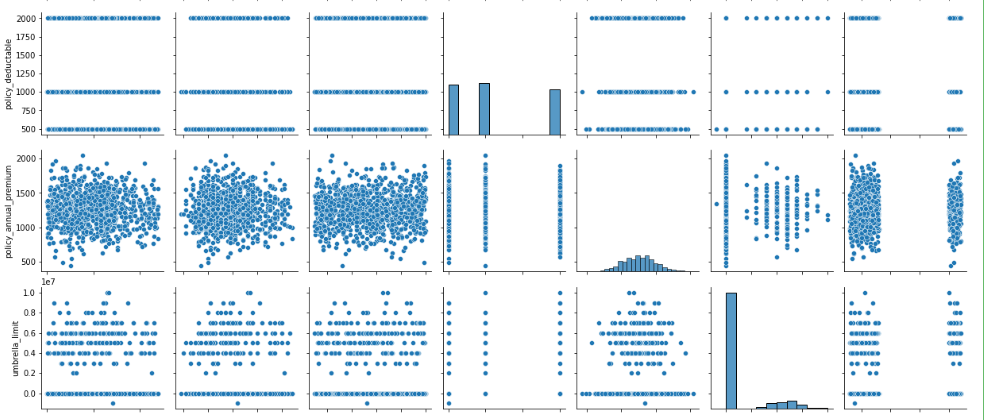
**Observation:** exec-manageral occupation has higher fruad reported

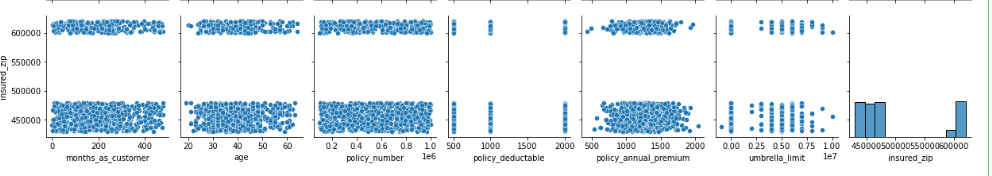
**Code:**

sns.pairplot(df.iloc[:,0:15])

**Output:**

****

****

****

**Observation:**

Total claim amount and vehicle claim are positively correalted

Vehicle claim and property claim are positively correalted

vehicle claim and total amount claim are positively correlated

months as customer age are positively correlated

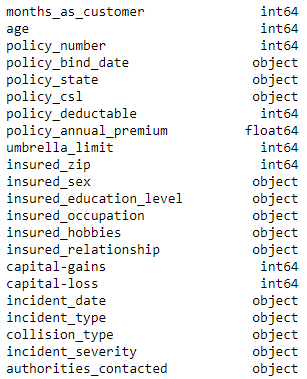
## **Pre-processing Pipeline**

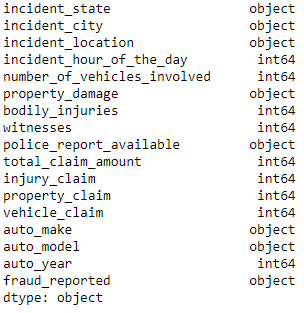
# Encoding

**Code:**

df.dtypes

**Output:**

****

****

from sklearn.preprocessing import LabelEncoder

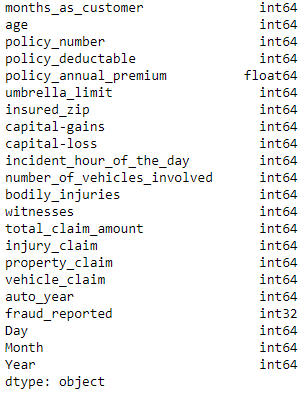
from sklearn.preprocessing import OrdinalEncoder

****

**Code:**

df.dtypes

**Output:**

****

**Observation:** we have succesfully converted all object data to numeric data.

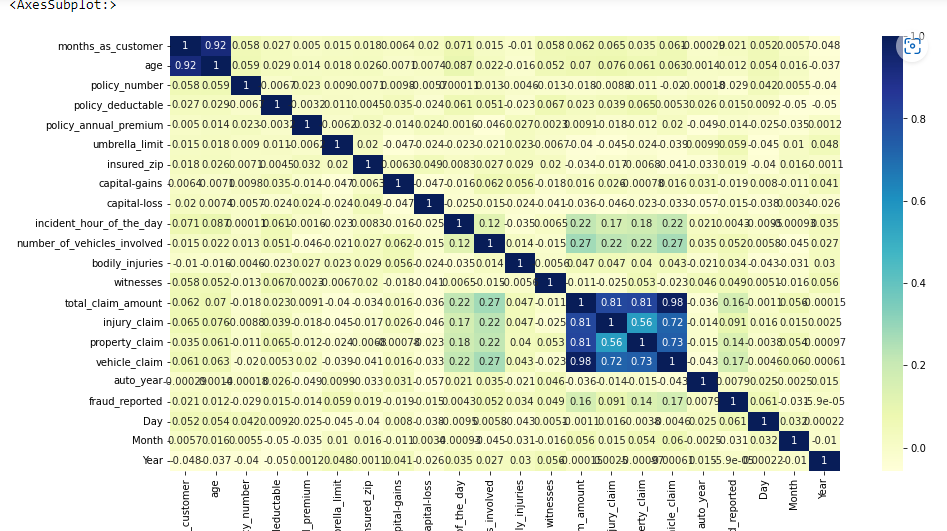
**Correlation:**

**Code:**

plt.figure(figsize=(15,8))

sns.heatmap(df.corr(),annot=True,cmap="YlGnBu")

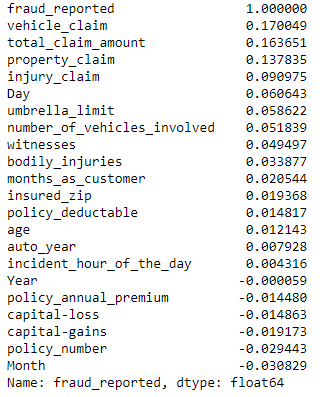
**Output:**

****

**Code:**

df.corr()['fraud\_reported'].sort\_values(ascending=False)

**Output:**

****

**Observation:**

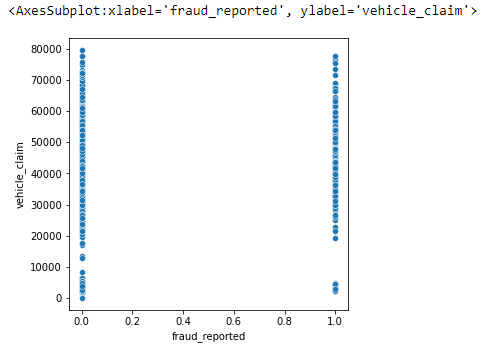
Vehicle\_claim,Total\_claim\_amount,property claim are positively correlated with Fraud\_reported

Other feautures are very less correlated with Fraud\_reported

**Code:**

sns.scatterplot(df[ 'fraud\_reported'],df[ 'vehicle\_claim'])

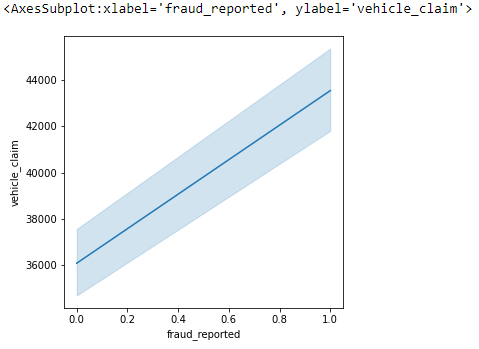
**Output:**

****

**Code:**

sns.lineplot(df[ 'fraud\_reported'],df[ 'vehicle\_claim'])

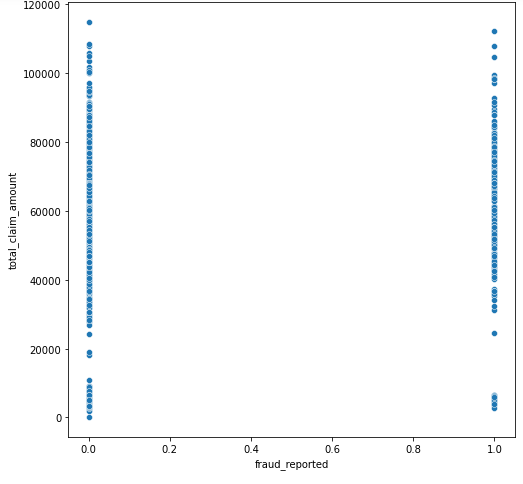
**Output:**

****

**Code:**

sns.scatterplot(df[ 'fraud\_reported'],df[ 'total\_claim\_amount'])

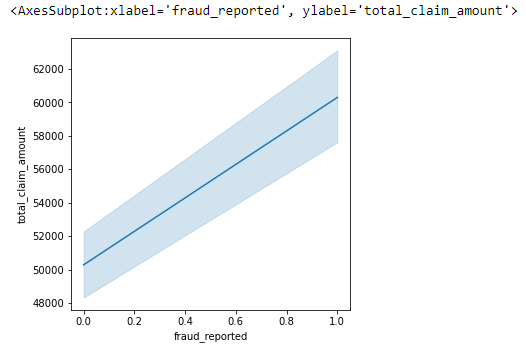
**Output:**

****

**Code:**

sns.lineplot(df[ 'fraud\_reported'],df[ 'total\_claim\_amount'])

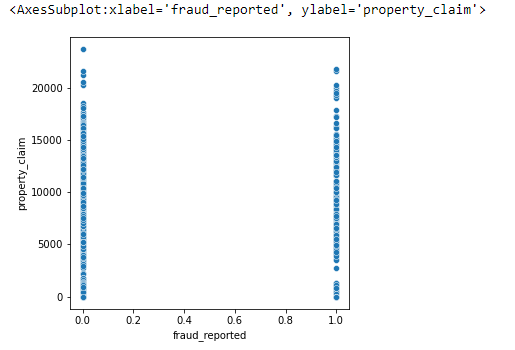
**Output:**

****

**Code:**

sns.scatterplot(df[ 'fraud\_reported'],df[ 'property\_claim'])

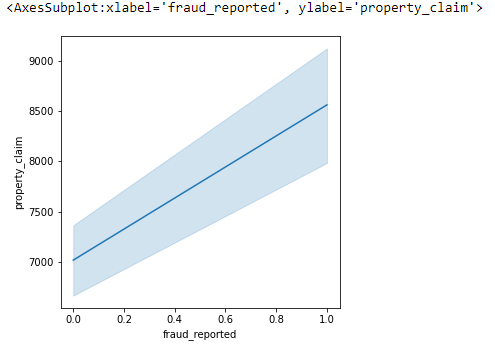
**Output:**

****

**Code:**

sns.lineplot(df[ 'fraud\_reported'],df[ 'property\_claim'])

**Output:**

****

# Skewness

**Code:**

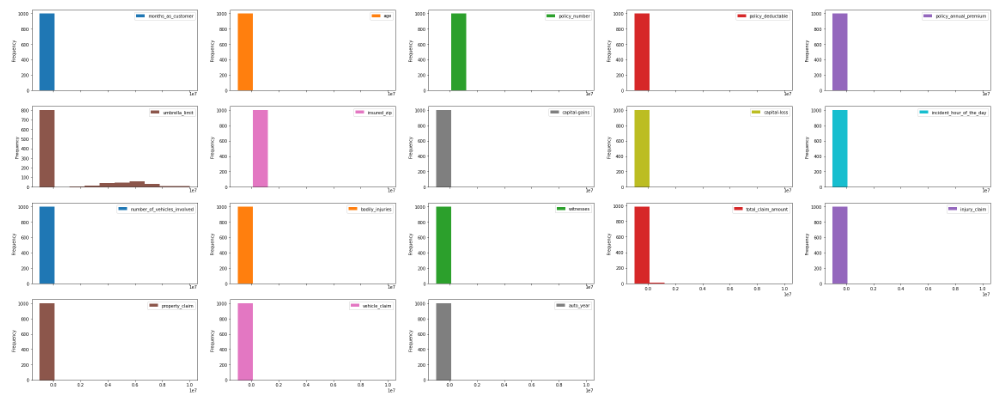
dc1 = df.drop(['fraud\_reported',"Day","Month","Year"],axis=1)

**Observation:** I am dropping Target Variable and creating a new data frame for checking skewness

**Code:**

dc1.plot(kind="hist",subplots=True,layout=(5,5),figsize=(40,20))

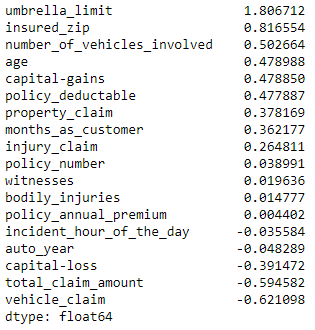
**Output:**



**Code:**

dc1.skew().sort\_values(ascending=False)

**Output:**



**Observation:** Many features have skewness proceeding with power transform for skewness removal.

**Code:**

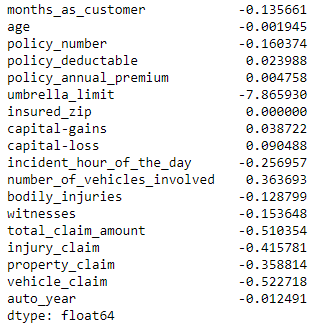
from sklearn.preprocessing import power\_transform

dc2 = power\_transform(dc1)

dc2 = pd.DataFrame(dc2,columns=dc1.columns)

dc2.skew()

**Output:**



**Observation:** we have removed skewness

# Outlier removal

**Code:**

dc2.plot(kind="box",subplots=True,figsize=(30,30))

from scipy.stats import zscore

z = np.abs(zscore(dc2))

np.where(z>3)

**Output:**



**Observation:** The outliers are removed

# Creating categorical data dataframe

**Code:**

dfd = pd.DataFrame()

dfd['fraud\_reported'] = df['fraud\_reported']

dfd["Day"] = df["Day"]

dfd["Month"] = df["Month"]

dfd["Year"] = df["Year"]

dfd = dfd.join(a)

dfd.shape

**Output:**



**Observation:** Removing the outlier removed rows in categorical dataframe

**Code:**

dfd.drop([229, 248, 290, 763],axis=0,inplace=True)

dfd.shape

**Output:**

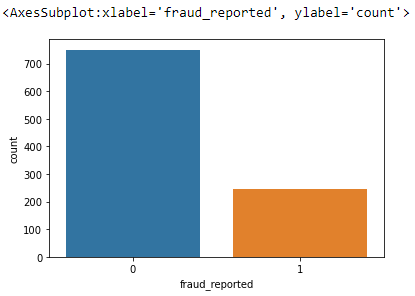


# Data Balancing

**Code:**

sns.countplot(dfd.iloc[:,0])

**Output:**



**Observation:**  Data is imbalanced so we need to balance it so we use smote to balance them

**Code:**

from imblearn.over\_sampling import SMOTE

sm = SMOTE()

dfb = df1.join(dfd)

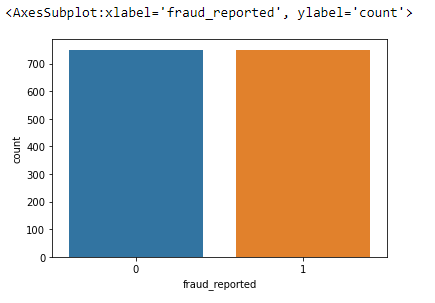
dx = dfb.drop("fraud\_reported",axis=1)

dy = dfd.iloc[:,0]

dft,y = sm.fit\_resample(dx,dy)

sns.countplot(y)

**Output:**



**Observation:** The Data is now balanced

# Standard Scaler

**Code:**

sc1=dft.iloc[:,0:18]

from sklearn.preprocessing import StandardScaler

sc= StandardScaler()

x1=sc.fit\_transform(sc1)

sc\_final = pd.DataFrame(x1,columns=sc1.columns)

**Observation:** We have Scaled the Data Which is not categorical and stored it in dataframe

# PCA

**Code:**

from sklearn.decomposition import PCA

x\_input = sc\_final.join(dft.iloc[:,18:])

x\_input.shape

**Output:**

****

**Code:**

pca = PCA().fit(dft)

fig, ax = plt.subplots(figsize=(20,10))

xi = np.arange(1, 1229, step=1)

yi = np.cumsum(pca.explained\_variance\_ratio\_)

plt.ylim(0.0,1.1)

plt.plot(xi, yi, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Components')

plt.xticks(np.arange(0, 1229, step=1)) #change from 0-based array index to 1-based human-readable label

plt.ylabel('Cumulative variance (%)')

plt.title('The number of components needed to explain variance')

plt.axhline(y=1, color='r', linestyle='-')

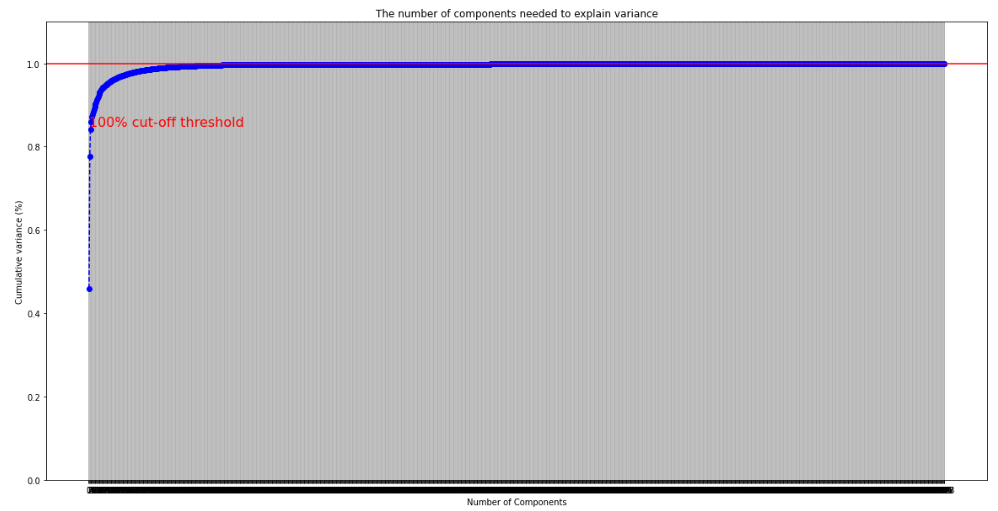
plt.text(0.5, 0.85, '100% cut-off threshold', color = 'red', fontsize=16)

ax.grid(axis='x')

plt.figure(figsize=(100,20))

plt.show()

**Output:**

****

**Observation:**

# we can see for 1110 columns i wont loose any data so i set threshold to 1110 for PCA

**Code:**

pca = PCA(n\_components=1110)

p\_final=pca.fit\_transform(x\_input)

x\_final = pd.DataFrame(p\_final)

x\_final.shape

**Output:**

****

**Observation:**

We have converted them to 1110 Columns. PCA will handle Multicollinearity so no need to check for them.

**EDA Concluding Remarks:**

**We have processed the data checked for null values, encoded categorical data, with outliers removed skewness, checked for outliers, balanced the target variables and finally reduced the columns using PCA.**

**BUILDING MACHINE LEARNING MODEL:**

For building machine learning models there are several models present inside the Sklearn module.

Sklearn provides two types of models i.e. regression and classification. Our dataset’s target variable is to predict whether fraud is reported or not. So for this kind of problem we use classification models.

**Code:**

from sklearn.model\_selection import train\_test\_split,cross\_val\_score

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.linear\_model import LogisticRegression

**Observation:** Importing all Libraries and functions required for Model Training and testing.

# I have write a code block that tests data with models and rank them according to cross val score and accuracy score

**Code:**

algo = [DecisionTreeClassifier(),KNeighborsClassifier(),RandomForestClassifier(),AdaBoostClassifier(),LogisticRegression()]

result = pd.DataFrame(columns=["Algorithm Name","Accuracy Score","Cross Validation Score"])

dtc=[]

knc=[]

rfc=[]

abc=[]

lr=[]

fl = [dtc,knc,rfc,abc,lr]

oo=0

for i in algo:

rand=0

acc=0

for ii in range(10):

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_final,y,test\_size=.20,random\_state=ii)

cf = i

cf.fit(train\_x,train\_y)

pred = cf.predict(test\_x)

ac = accuracy\_score(test\_y,pred)

if ac>acc:

acc=ac

rand=ii

print(f' the best random state is {rand} and accuracy score is {acc} for algorithm {i}')

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_final,y,test\_size=.20,random\_state=rand)

cf = i

cf.fit(train\_x,train\_y)

pred = cf.predict(test\_x)

ac = accuracy\_score(test\_y,pred)

cv = cross\_val\_score(cf,x\_final,y,cv=5).mean()

fl[oo].insert(0,i)

fl[oo].insert(1,ac)

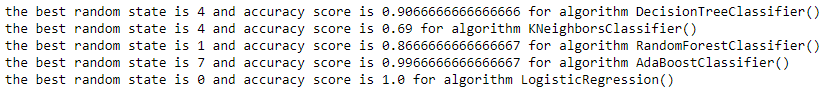
fl[oo].insert(2,cv)

result.loc[oo] = fl[oo]

oo +=1

final\_result = result.sort\_values(by=["Accuracy Score","Cross Validation Score"],ascending=False)

**Output:**

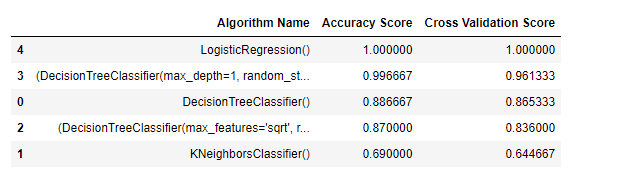
****

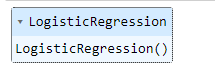
**Code:**

final\_result

final\_result.loc[4]["Algorithm Name"]

**Output:**

****

****

**Observation:**

This code prints the best random state for algorithm and ranks them in Data Frame. As we can see Support Vector Classifier performed best compared to all.

# Support vector classifier

**Code:**

svc = SVC()

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_final,y,test\_size=.20,random\_state=10)

svc.fit(train\_x,train\_y)

pred = svc.predict(test\_x)

ac = accuracy\_score(test\_y,pred)

cv = cross\_val\_score(svc,x\_final,y,cv=5).mean()

print(f' the accuracy socre is {ac}, the cross validation score is {cv} for SVC' )

**Output:**

****

**Observation:**

from all the classifier models Logistic Regression performed best with accuracy of 100 % and cross val score of 100 %

# HyperParameter Tunining

In the GridSearchCV approach, the machine learning model is evaluated for a range of hyper parameter values. This approach is called GridSearchCV, because it searches for best set of hyper parameters from a grid of hyper parameters values.

**Code:**

from sklearn.model\_selection import GridSearchCV

lor = LogisticRegression()

parameters = {'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],

'penalty': ['l1', 'l2', 'elasticnet', 'none'],

'dual': [True,False],

'fit\_intercept': [True,False],

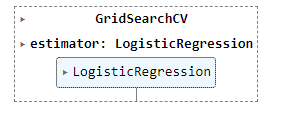
"max\_iter":[100,150]}

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_final,y,test\_size=.20,random\_state=1)

gsv = GridSearchCV(lor,parameters)

gsv.fit(train\_x,train\_y)

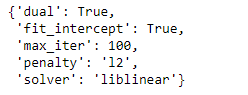
**Output:**

****

**Code:**

gsv.best\_params\_

**Output:**

****

**Observation:** Now we got best parameters using Grid Search CV

**Model Training:**

**Code:**

lor = LogisticRegression(dual=False,fit\_intercept=True,max\_iter=150,penalty='none',solver='saga')

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_final,y,test\_size=.20,random\_state=15)

lor.fit(train\_x,train\_y)

pred = lor.predict(test\_x)

ac = accuracy\_score(test\_y,pred)

cv = cross\_val\_score(lor,x\_final,y,cv=5).mean()

print(f' the accuracy socre is {ac}, the cross validation score is {cv} ' )

**Output:**

****

**Observation:** the accuracy score is 100% and cross validation score is 100%

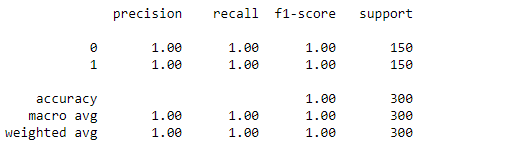
**Code:**

print(confusion\_matrix(test\_y,pred))

print(classification\_report(test\_y,pred))

**Output:**

****

****

**Observation:** the accuracy score is 100% and cross validation score is 100%

# ROC\_AUC\_CURVE

# ****ROC curve:****Itis a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.

**Code:**

from sklearn.metrics import plot\_roc\_curve,roc\_auc\_score

print(f'The Roc Auc Score is {roc\_auc\_score(test\_y,pred)}')

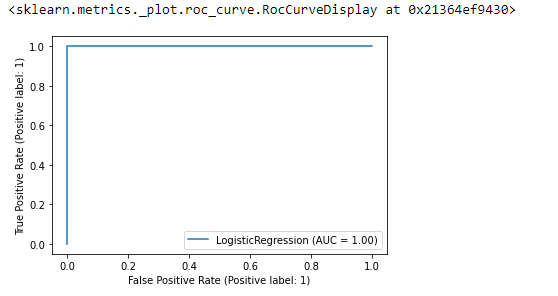
**Output:**

****

**Code:**

plot\_roc\_curve(lor,test\_x,test\_y)

**Output:**

****

**Observation:** We got Good AUC – ROC score of 100%

# Saving the model

**Code:**

import joblib

joblib.dump(lor,"Insurance Claims.pkl")

**Output:**

# 

**CONCLUSION:**

We have conducted the Data Analysis on Insurance Claims- Fraud Detection Dataset. Built the best machine learning model using hyper parameter tuning.

This project has built a model that can detect auto insurance fraud. In doing so, the model can reduce losses for insurance companies. The challenge behind fraud detection in machine learning is that frauds are far less common as compared to legit insurance claims.

Five different classifiers were used in this project: logistic regression, K-nearest neighbors, Random forest, Decision tree, GaussianNB. Four different ways of handling imbalance classes were tested out with these five classifiers: model with class weighting, oversampling with SMOTE, hyper parameter tuning, and plotting roc curve of the models.