MACHINE LEARNING MODEL TO DETECT FRAUD INSURANCE CLAIM.

1) PROBLEM DEFINITION:

Insurance fraud is a huge problem in the industry of Automobiles. It's difficult to identify fraud claims. Machine Learning is the best technique to help the Auto Insurance industry with this problem.

Today, there are many people who submit false information or report to the insurance company for getting compensation for the incident which never happened in reality. Such people misuse the law which is built for their benefit for personal gain. So, this machine learning model which we have built will try to eradicate this ongoing crisis in the Automobile Industry.

Also, many people try to create a fake incident to gain compensation or insurance from the insurance company. The major issue in detecting a false claim is that sometimes officers don't have proper background history of the applicant. However, we have taken the data from a reputed source, and it contains the personal history of the applicant.

Next, we will study about the data and the information we were able to extract from the dataset through visual representation like graphs, pie charts, bar plots, etc. Also, we have tested the machine learning model, and it is a combination of robustness, and the model will give accurate answer to whether or not the insurance claim is fraudulent or not?

2) DATA ANALYSIS:

We will use the dataset provide to us from a reputed source and do some research on the data like maximum and minimum compensation amount claimed, the region from where most insurance claim fraud is likely to happen, etc. First, we will import the dataset necessary libraries and do learn some basic essential information about the dataset.

Importing the necessary libraries

```
[1]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import plotly.express as px

[2]: import matplotlib.pyplot as plt
     %matplotlib inline
```

Importing the dataset

```
[3]: df = pd.read_csv('Automobile_insurance_fraud.csv', header=None)
     df
```

1000 rows × 39 columns

We see that the dataset contains 1000 samples and there are 38 features for training the machine learning model plus one target variable column.

```
[9]: df.info()
<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               1000 non-null   object
 20  incident_severity            1000 non-null   object
 21  authorities_contacted        909 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              1000 non-null   object
 28  bodily_injuries              1000 non-null   int64
 29  witnesses                    1000 non-null   int64
 30  police_report_available      1000 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
```

There are 20 columns in the dataset that are categorical column, and the rest of the columns are of integer and floating data type. Also, we substituted the "?" values in the dataset as null values for easier filling of them. We found out that there are total 972 null values in the dataset from 4 columns namely collision type, authorities contacted, property damage and police_report_available.

```
[16]: df.duplicated().sum()
[16]: 0

[22]: df.isnull().sum().sum()
[22]: 972
```

```
[19]: df.skew(numeric_only=True)
[19]: months_as_customer            0.362177
      age                           0.478988
      policy_number                 0.038991
      policy_deductable             0.477887
      policy_annual_premium         0.004402
      umbrella_limit                1.806712
      insured_zip                   0.816554
      capital-gains                 0.478850
      capital-loss                 -0.391472
      incident_hour_of_the_day     -0.035584
      number_of_vehicles_involved   0.502664
      bodily_injuries               0.014777
      witnesses                     0.019636
      total_claim_amount           -0.594582
      injury_claim                  0.264811
      property_claim                0.378169
      vehicle_claim                -0.621098
      auto_year                    -0.048289
      dtype: float64
```

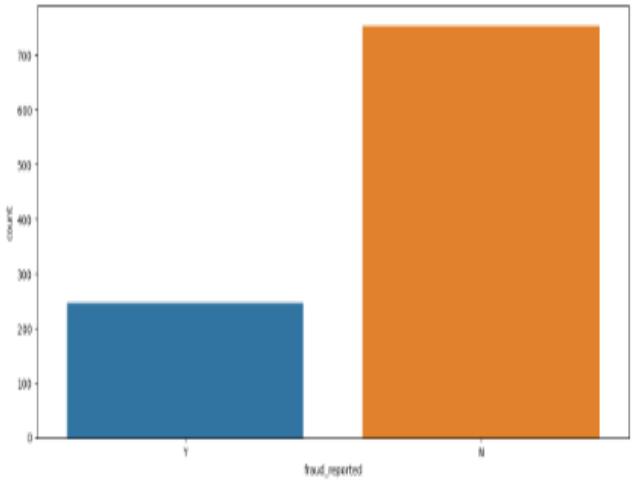
We checked that the dataset has any duplicate records or not and the answer was No, the dataset was not having any duplicate records. So, we moved further by checking the skewness of the dataset. The only column whose k=skewness was above the permissible limit was "umbrella_limit" and we will remove that during preprocessing phase.

Observation from the pandas describe method applied on the data frame.

1) The mean policy_annual_premium is 1256.40 units and the mean total_claim_amount is 52761.94 units.
2) All the vehicles involved in accidents are from the year 1995 onwards.
3) The policy number column has no say in the target variable at all.
4) On Average the injured victims in one accident is 0.99 and there are 1.48 average witnesses of the accident.

Exploratory Data Analysis (EDA)

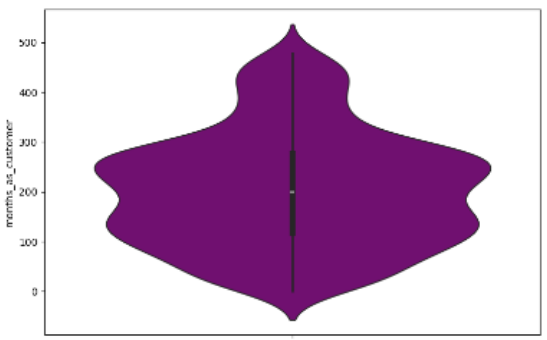
Exploratory data analysis (EDA) is used by data scientists to analyse and investigate data sets and summarize their main characteristics, often employing data visualization methods like bar plots, histogram plots, box plots pie charts and count plots.

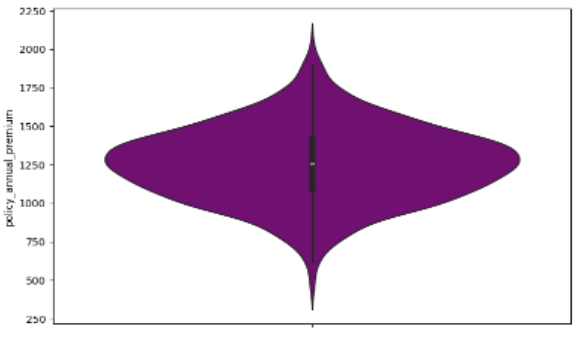


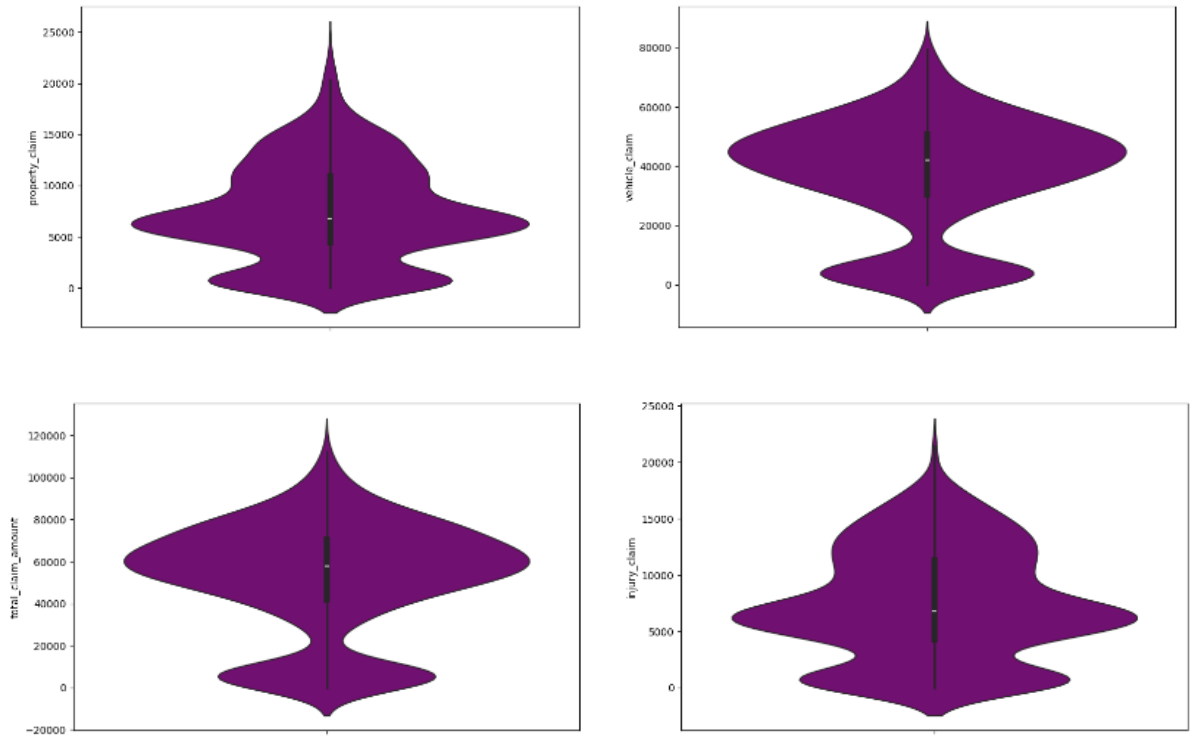

```
[34]: df['fraud_reported'].value_counts()

[34]: fraud_reported
      N    753
      Y    247
      Name: count, dtype: int64
```

Around 753 or 75.3 incidents or application are valid insurance claim samples whereas 247 or 24.7 samples or applications have done insurance fraud with the insurance company.

Now let's take a look at various violin plots of numerical columns to see how the data in those columns is distributed.



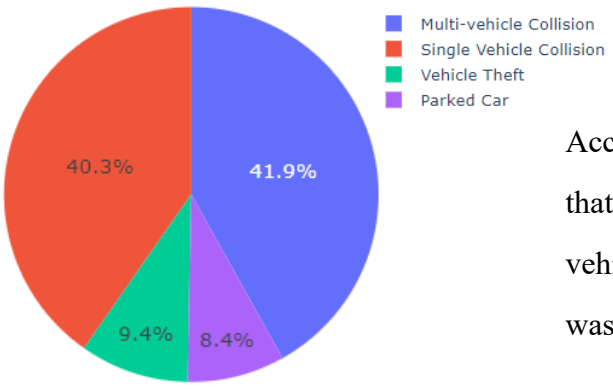




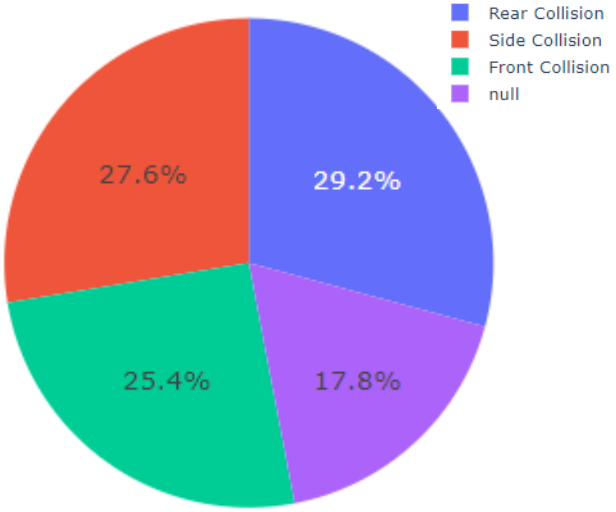
Key Insights from Above Violin Plots:

- Most policy holders have been subscribed to the policy for 100 to 300 months which means around 8 years to 25 years.
- The annual policy premium price or instalments is evenly distributed and there are almost equal policy holders in every price segment.
- But the insurance claim amount KDE plot is not evenly distributed as some applicants demand astronomical figure claim and this trend continues in Injury Claim data as well.
- Most people claim around 30,000 to 60,000 amounts for vehicle insurance claim and around 3,000 to 12,000 for property insurance claim.

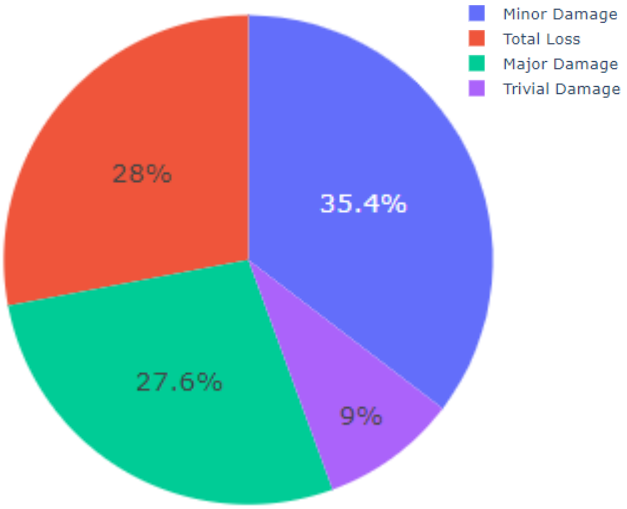
Pie Charts for different features ↓



According to the left side pie chart, we get to that 9.4% of the insurance claim applicant's vehicle was stolen and 8.4% applicant's vehicle was involved in the accident when it was parked.

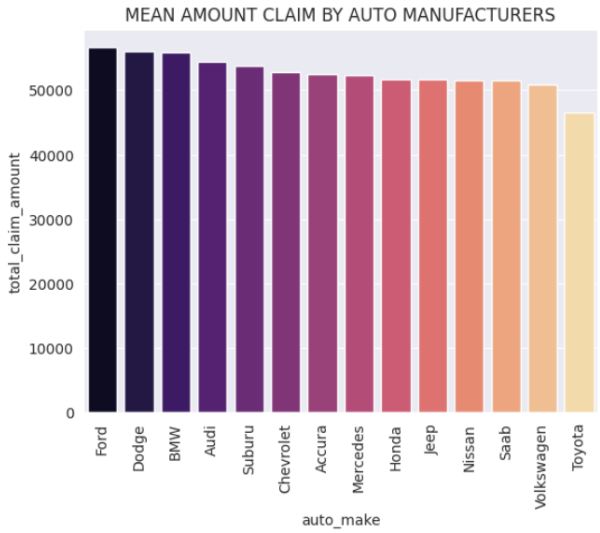


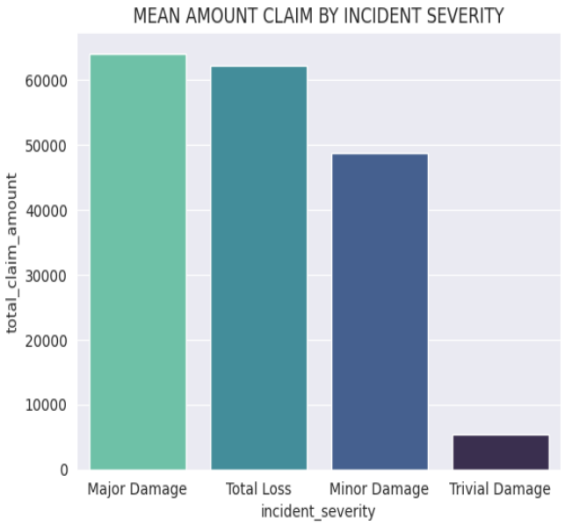
This pie chart shows that around 29.2% accidents' collision type is rear collision, and 27.6% collision type is side and finally 25.4% accidents are of front collision. So, one thing we can assume that the vehicle's driver had no fault in the collision has been the affected the most.



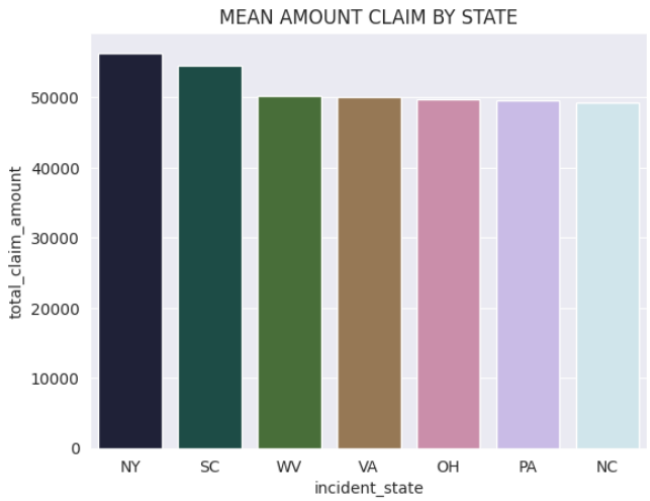
This chart shows that in majority of accidents i.e. 35% there is only minor damage to the vehicle and 28% vehicles are beyond repair after the accident while around 27.6% vehicles suffer major damage.

Bar Plots of average claim amount by different factors

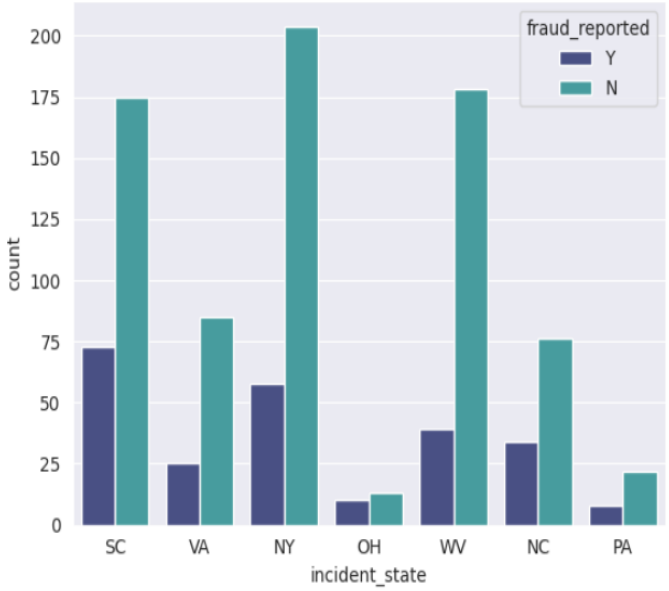




The above bar plot shows that people who drive Ford cars generally claim more insurance amount of around 56000 on average and more than any other applicant. Also, the people who driver Toyota which is generally a budget car vehicle as expected demand less insurance amount and people who drive expensive cars like Dodge, BMW and Audi demand a high level of insurance amount. Then, moving to the next bar plot, we see that if the accident is major one, the vehicle's owner demands around 63000 amounts whereas if there is trivial damage which means a minor one, they obviously demand less compensation and the mean amount in that cases is 5000.



This bar plot shows that in some states like New York and South Carolina which are pretty expensive to live in the average insurance claim amount is high like 55,000 and 53,000 respectively.



fraud_reported	N	Y
incident_state		
NC	76	34
NY	204	58
OH	13	10
PA	22	8
SC	175	73
VA	85	25
WV	178	39

The above bar plot and cross tab show that Ohio state has almost the equal share of genuine and fraudulent insurance claims whereas Wyoming has the least percentage of fraudulent insurance claims. And as we discussed earlier, New York

and South Carolina are the worst states that have high amounts of fraudulent insurance claims.

Cross Tabs data:

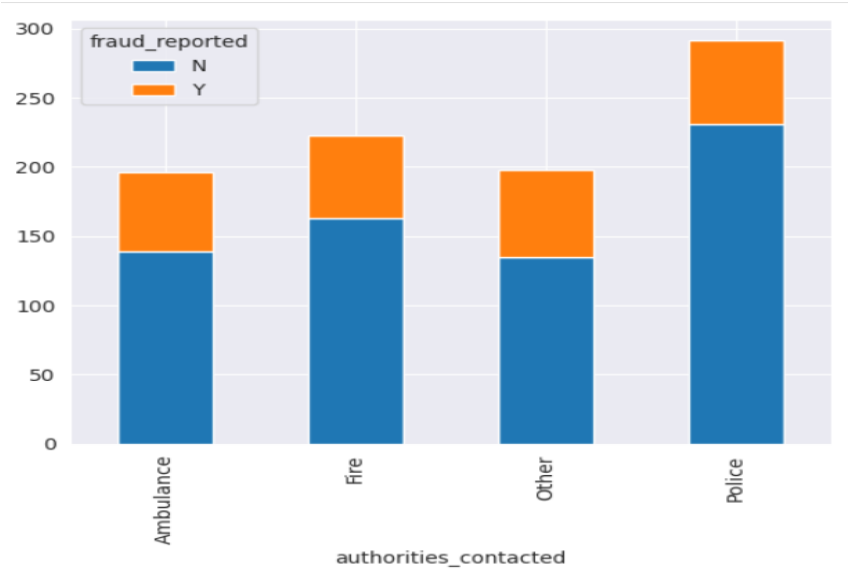
```
pd.crosstab(index=df['fraud_reported'], columns=df['insured_occupation'])
```

insured_occupation	adm-clerical	armed-forces	craft-repair	exec-managerial	farming-fishing	handlers-cleaners	machine-op-inspct	other-service	priv-house-serv	prof-specialty	protective-serv	sales	tech-support	transport-moving
fraud_reported														
N	54	52	52	48	37	43	71	59	59	67	49	55	56	51
Y	11	17	22	28	16	11	22	12	12	18	14	21	22	21

The above cross tab represents the number of fraudulent claims vs genuine claims by different occupation of the applicants. We find out that executive managerial occupation has the highest percentage of fraud convictions in terms of insurance claims.

fraud_reported	N	Y
authorities_contacted		
Ambulance	139	57
Fire	163	60
Other	135	63
Police	231	61

This cross tab represents the number of fraudulent claims vs genuine claims by different authorities they contacted for seeking help by the applicants. The important observation is that people who did not contact ambulance, fire and police have the highest number of fraudulent claims whereas the rest of the data is more or less the same, so we cannot make any further assumptions based on this cross tab. The below bar plot gives more information about this cross tab.

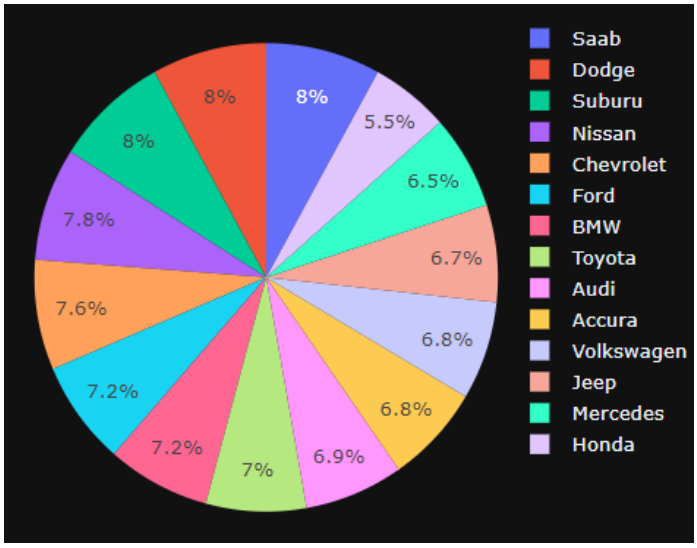


3) EDA Concluding Remarks:

fraud_reported	N	Y
police_report_available		
NO	257	86
YES	242	72

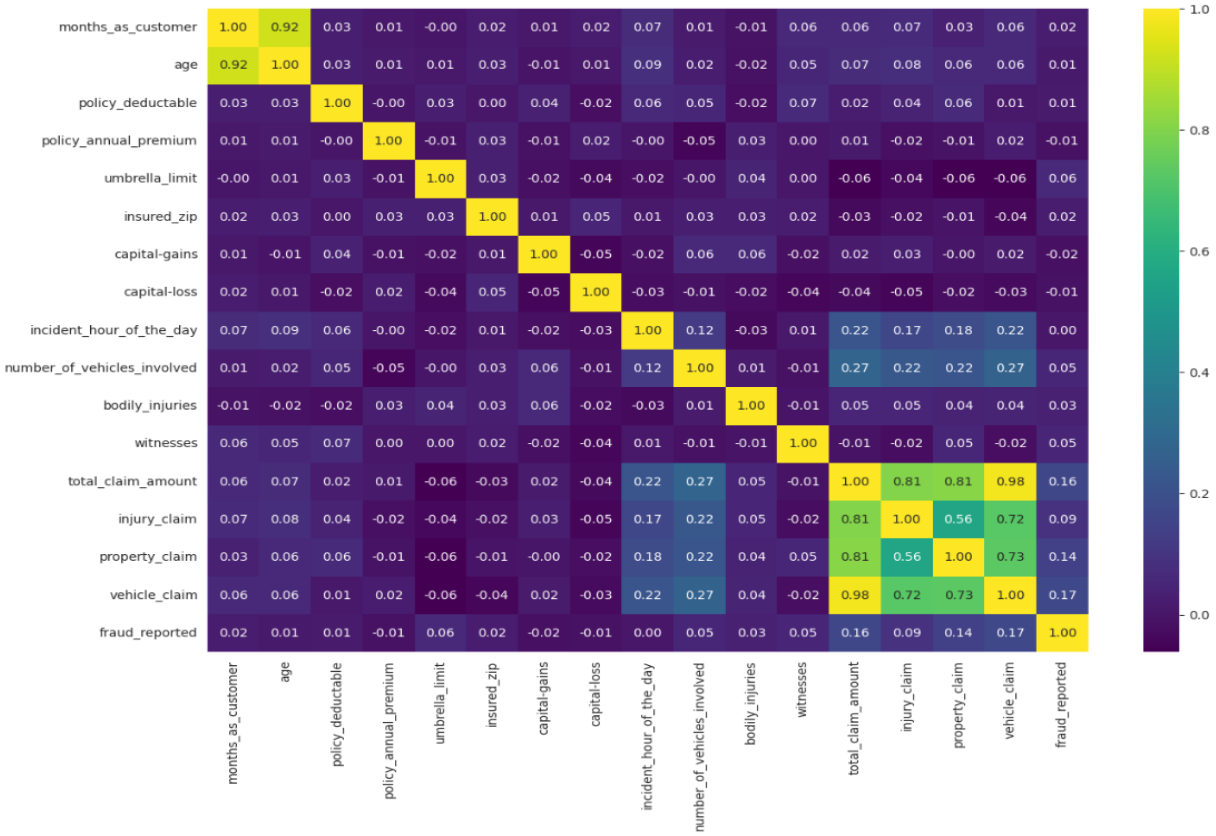
fraud_reported	N	Y
property_damage		
NO	272	66
YES	224	78

The two cross tabs above give some more information related to the fraudulent insurance claims like if there is no police report available in such cases, there is a high risk of insurance fraud case.



This bar plot tells us that Subru, Dodge and Saab are the most common vehicle company cars involved in the insurance claim applications.

Overall, we have learned enough about the data that we understand that the important features that determine the outcome of the model like Police report available, the authorities that were contacted when the incident happened, the region in which the incident occurred are all in some way or the other have an influence on the fraud reported variable. Below is the heatmap for all the numerical columns data and the important observation is that the total claim amount, the property claim amount, the injury claim amount and the vehicle claim amount are co-related with each other.



4) Pre-processing Pipeline:

Now, we will do feature engineering on the data first and then scale the data using Standard Scaler library for training the data on some machine learning models.

a) Filling all the null values:

[79]:
```
from sklearn.impute import SimpleImputer
```

[80]:
```
si = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
si
```

[80]: SimpleImputer(strategy='most_frequent')

[81]:
```
cols_to_be_filled_for_nullvalues = ['police_report_available','collision_type','authorities_contacted' , 'property_da
cols_to_be_filled_for_nullvalues
```

[81]:
```
['police_report_available',
 'collision_type',
 'authorities_contacted',
 'property_damage']
```

[82]:
```
for x in cols_to_be_filled_for_nullvalues:
    df[[x]] = si.fit_transform(df[[x]])
df
```

We have now filled all the null values in the dataset using Simple Imputer library which fills the null values with the mean value of that particular column or feature.

b) Dropping irrelevant features:

```
[85]: df.drop(['policy_number', 'policy_bind_date', 'incident_date', 'incident_location','auto_model', 'auto_year'], axis=1
      df
```

We have dropped the columns which have no effect on fraud reported target variable.

c) Label Encoding on Target Variable:

```
[86]: from sklearn.preprocessing import LabelEncoder

[87]: enc = LabelEncoder()

[89]: df['fraud_reported'] = enc.fit_transform(df['fraud_reported'])
      df
```

As the target variable contains 'Y' and 'N' values we need to transform them into binary values 1 and 0 respectively so that our model can understand them.

d) Encoding Categorical Columns:

```
[96]: df = pd.get_dummies(data=df, columns=cat_cols, dtype=int, drop_first=True)
      df
```

We converted all categorical columns using the built in One Hot Encoding Technique in the pandas' library so that our model can learn about categorical data along with the numerical data to detect insurance fraud claim.

e) Removing Skewness from the dataset:

```
[98]: from sklearn.preprocessing import PowerTransformer

[101… powertransformer = PowerTransformer(method='yeo-johnson',standardize=True,copy=True)

[104… df[['umbrella_limit']] = powertransformer.fit_transform(df[['umbrella_limit']])
      df
```

Now, we removed skewness from the only column we could remove which is umbrella limit column using Power Transformer technique yeo-Johnson method which can deal with both positive and negative values.

f) Splitting the dataset into training data and testing data:

```
[174…] X = df.drop('fraud_reported', axis=1)
       y = df['fraud_reported']

[175…] from sklearn.model_selection import train_test_split

[176…] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)

[177…] X_train.shape

[177…] (800, 102)

[178…] X_test.shape

[178…] (200, 102)
```

SMOTE OVERSAMPLING TECHNIQUE

```
[179…] y_train.value_counts()

[179…] fraud_reported
       0    593
       1    207
       Name: count, dtype: int64
```

- We need some data to train the model and some data to evaluate the performance of the machine learning model. Hence, we split the data using train test split library which splits the data into two parts, and you can choose the percentage of distribution of the data for the two sets. We have chosen 80:20 ratio i.e. 80% data for training data and the rest 20% of data for testing purposes.
- Another issue which arises here is that the dataset is imbalanced. So, we need to balance it out so that our model learns about both the classes equally.

g) Handling Imbalanced Data Using SMOTE:

```
[180…] from imblearn.over_sampling import SMOTE

[181…] smote = SMOTE()

[182…] X_train, y_train = smote.fit_resample(X_train, y_train)
```

We must do oversampling of the dataset because no fraud values are 74% and fraud yes values are 26% of the total samples. So, Synthetic Minority Over Sampling Technique is used to solve this problem.

```
[186…] y_train.value_counts()

[186…] fraud_reported
       0    593
       1    593
       Name: count, dtype: int64
```

We see that after applying SMOTE, there are equal number of samples of yes and no values. So, our model will now learn about both the classes equally.

h) Scaling of data using Standard Scaler:

Scaling the data is important because our model doesn't know about the unit of the data. So, it is necessary step we scaled the data using Standard Scaler library.

```
[187…] from sklearn.preprocessing import StandardScaler
```

```
[188…] scaler = StandardScaler()
```

```
[189…] X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
       X_train
```

[189…]

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-loss	incident_hour_of_the_day
0	0.814913	1.010429	-0.247184	1.182278	-0.428048	1.659664	-0.881647	-1.003860	0.069390
1	2.002889	2.245628	-1.102180	0.362531	1.935982	-0.391885	-0.881647	0.953699	-0.550547
2	0.016883	-0.337062	1.462806	-0.147995	-0.428048	-0.690317	-0.881647	0.953699	0.844311
3	0.007815	-0.337062	-1.102180	-0.442992	-0.428048	-0.966075	-0.881647	0.953699	0.689327
4	-1.152956	-1.459971	-0.247184	0.356792	-0.428048	-0.811791	1.467910	0.953699	0.069390
…	…	…	…	…	…	…	…	…	…
1181	0.570063	0.561265	1.462806	-0.711132	0.878370	-0.430788	-0.881647	0.953699	-0.705531
1182	-0.735804	-1.010807	1.055828	0.834796	1.645609	1.553076	1.121078	0.953699	-1.635437
1183	-0.699530	-0.786226	1.365337	-2.022956	-0.428048	-0.809190	-0.881647	-0.450912	-1.635437
1184	-0.527228	-0.673935	-1.102180	-0.163369	0.463383	-0.440712	1.156212	-1.415627	0.379359
1185	-0.717667	-0.561644	-1.102180	1.361218	1.043812	1.465502	-0.881647	0.953699	-1.480453

1186 rows × 102 columns

```
[190…] X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
       X_test
```

[190…]

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-loss	incident_hour_of_the_day
0	-0.690462	-1.123098	-0.247184	-1.182469	-0.428048	-0.300112	1.507918	0.953699	-1.015500
1	-0.119145	-0.449353	-1.102180	0.665885	-0.428048	-0.839583	-0.881647	0.953699	-0.705531
2	-1.089477	-0.898516	-0.247184	-0.778166	-0.428048	-0.703125	-0.881647	-0.746192	0.689327
3	-1.796821	-1.684553	1.462806	0.194143	-0.428048	-0.391602	0.998726	-0.238014	-0.860516
4	-1.216436	-1.459971	1.462806	-0.283888	1.040979	-0.455725	0.184019	0.953699	-0.395563
…	…	…	…	…	…	…	…	…	…
195	-0.717667	-1.010807	-0.247184	-0.831933	-0.428048	-0.317726	0.693211	-1.415412	-0.550547
196	0.288939	-0.000189	-1.102180	0.583969	-0.428048	-0.830408	-0.881647	0.953699	-1.790421
197	-1.007860	-1.123098	-0.247184	0.068547	1.347857	1.660202	0.845969	-1.147008	0.534343
198	-0.753941	-0.786226	-0.247184	0.220519	-0.428048	-0.440712	0.184019	0.953699	1.774217
199	0.751433	0.898138	-0.247184	0.569241	-0.428048	-0.397639	-0.881647	-1.243634	1.309264

200 rows × 102 columns

The data preprocessing part is now complete, and our training data is ready to be sent to the machine learning model.

5) Building Machine Learning Models:

We will build multiple Machine Learning models basically Supervised learning binary classification Machine Learning models to predict fraud 'Yes' /1 or 'No'/0 using many classification algorithms.

First, we will build Logistic Regression model which is the most popular model for classification purposes.

```
[192…] from sklearn.linear_model import LogisticRegression

[193…] log_model = LogisticRegression()

[194…] log_model.fit(X_train, y_train)

[194…] LogisticRegression()

[195…] log_pred = log_model.predict(X_test)
       log_pred

[198…] print('ACCURACY SCORE : ', accuracy_score(y_test, log_pred))
       print("Training accuracy :", log_model.score(X_train, y_train))
       print("Test accuracy :", log_model.score(X_test, y_test))
       print('ROC AUC SCORE : ', roc_auc_score(y_test, log_pred))
       print('Cohen\'s Kappa Score :', cohen_kappa_score(y_test, log_pred))
       print('\n\n \n-------------------Confusion Matrix--------------\n')
       print(confusion_matrix(y_test, log_pred))
       print('\n\n-------------------Clasification Report--------\n')
       print(classification_report(y_test, log_pred))
```

The above code was used to build the model and the model gave an accuracy of 0.82 which is really a good score. The few more metrics of this model are given below.

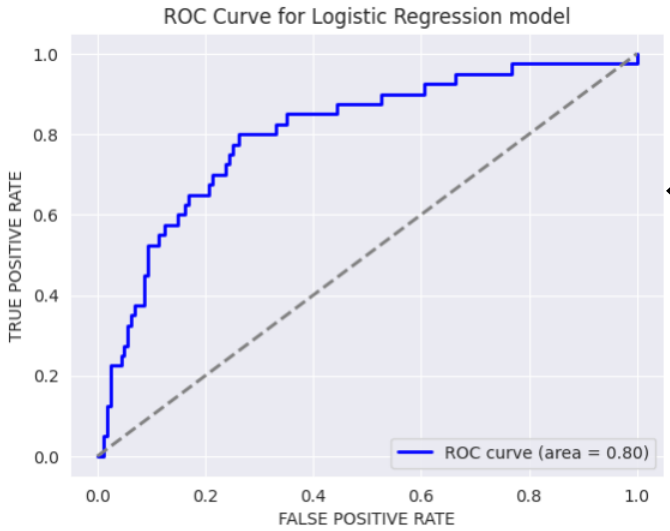
```
ACCURACY SCORE :  0.82
Training accuracy : 0.9258010118043845
Test accuracy : 0.82
ROC AUC SCORE :  0.709375
Cohen's Kappa Score : 0.4267515923566879
```

```
-------------------Confusion Matrix--------------

[[143  17]
 [ 19  21]]
```

↑↓

```
-------------------Clasification Report--------

              precision    recall  f1-score   support

           0       0.88      0.89      0.89       160
           1       0.55      0.53      0.54        40

    accuracy                           0.82       200
   macro avg       0.72      0.71      0.71       200
weighted avg       0.82      0.82      0.82       200
```

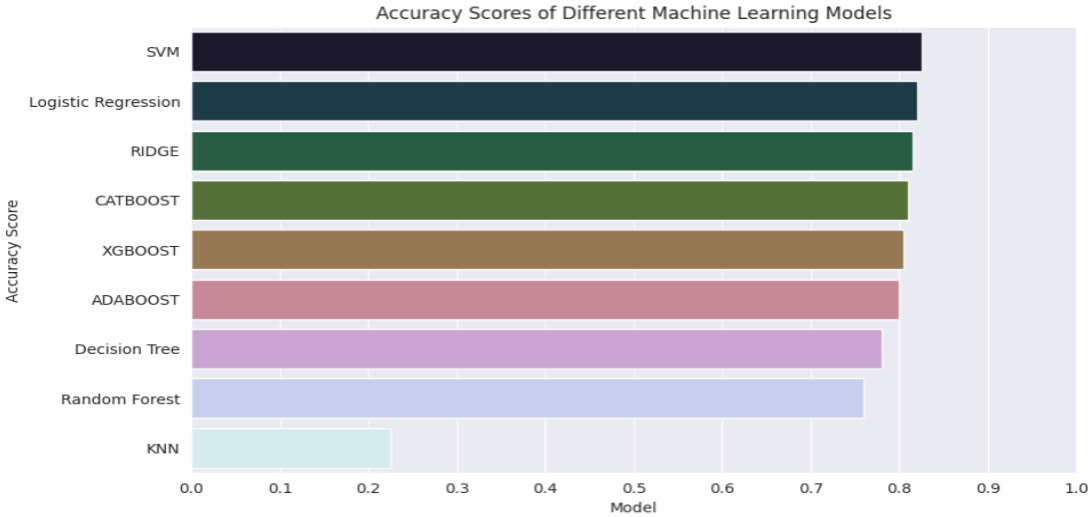
Our model gives roc auc score of 0.70 which is good as we balanced our dataset using SMOTE technique. There is also no issue of overfitting and underfitting as our testing accuracy and training accuracy are close to each other.



As you can see here on the left side, we have roc curve and area under curve (AUC) is 0.80 which is decent.

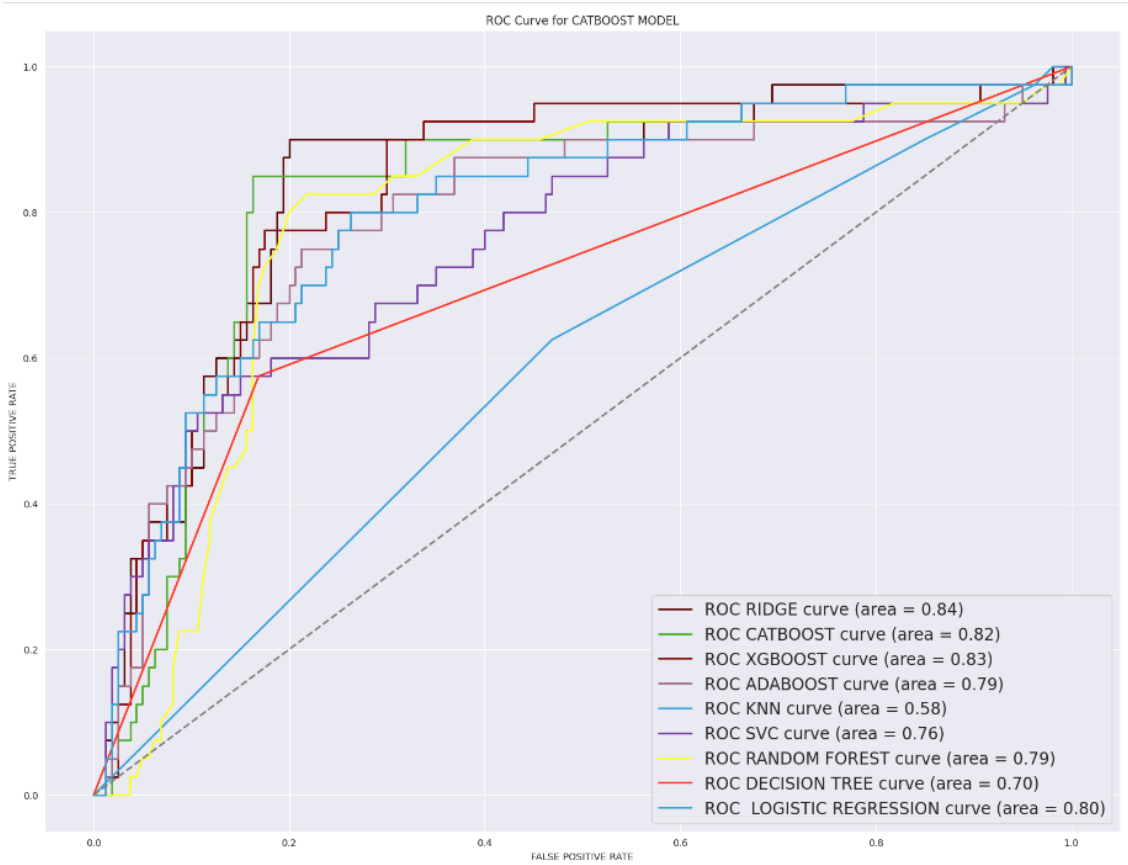
Now, we build multiple models like Decision Tree, Random Forest, SVM, K Nearest Neighbour, XGBOOST, CATBOOST to check which model will give the overall improved performance considering factors like accuracy, roc-auc-score and area under curve (AUC).

Machine Learning Models Evaluation:




```
Logistic Regression ROC AUC SCORE IS 0.709375
Decision Tree ROC AUC SCORE IS 0.703125
Random Forest ROC AUC SCORE IS 0.5593750000000001
SVM ROC AUC SCORE IS 0.609375
KNN ROC AUC SCORE IS 0.50625
ADABOOST ROC AUC SCORE IS 0.6875
XGBOOST ROC AUC SCORE IS 0.728125
CATBOOST ROC AUC SCORE IS 0.7218749999999999
RIDGE ROC AUC SCORE IS 0.725
```

As you can see the Support Vector Machine Learning model gives the best accuracy however XGBOOST gives the best roc auc score. Let's take a look at various roc curve and AUC scores.



Now, Ridge Classifier model gives the best auc score and curve of 0.84 which is impressive, but we will consider the most important three accuracy metrics to decide the model we want to select for further training.

The key insights from the accuracy metrics are as follows:

- Now, we have trained and tested all models and the best performing model is SUPPORT VECTOR MACHINE CLASSIFIER (SVC) which give 82% accuracy and 0.60 roc auc score.
- We will not select this model because if we consider other metrics as well, then CATBOOST, XGBOOST AND RIDGE CLASSIFIER these 3 models are locked in a three-horse race and the difference in Accuracy, ROC AUC SCORE AND Area Under Curve of these 3 models is almost negligible as you can see in the above graph and output above.
- We will select Ridge Classifier as it gives the best all round performance. It gives accuracy of almost 82% accuracy and roc auc score of 0.73 and area under the curve (AUC) is 0.84.

- Now, we will try to hyperparameter tune our RIDGE CLASSIFIER model to see if we can improve even further.

Hyperparameter Tuning:

```
[363…] from sklearn.model_selection import GridSearchCV

[369…] param_grid = {
           'alpha': [0.1, 1, 10, 100, 150, 200, 250, 300],
           'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
       }

[370…] grid = GridSearchCV(ridge, param_grid=param_grid, scoring='accuracy',n_jobs=-1,cv=5,verbose=4)

[371…] grid.fit(X_train, y_train)

[371…] Fitting 5 folds for each of 56 candidates, totalling 280 fits
       GridSearchCV(cv=5, estimator=RidgeClassifier(), n_jobs=-1,
                    param_grid={'alpha': [0.1, 1, 10, 100, 150, 200, 250, 300],
                                'solver': ['auto', 'svd', 'cholesky', 'lsqr',
                                           'sparse_cg', 'sag', 'saga']},
                    scoring='accuracy', verbose=4)
```

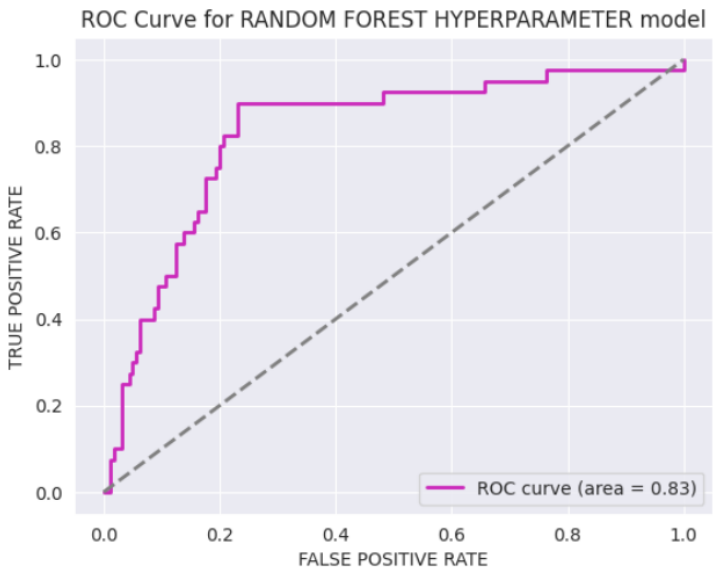
Grid Search technique was used to hyperparameter tune the Ridge Classifier model using parameters like alpha and solver and the results are given below:

```
[372…] grid.best_score_

[372…] 0.8517639967379358

[373…] grid.best_params_

[373…] {'alpha': 100, 'solver': 'lsqr'}
```

Now, the best score is 0.85 and alpha value should be 100 and ‘lsqr’ should be the solver values if we want to achieve maximum potential of the model. Let’s test out these parameters by giving them to ridge classifier model and again train and evaluate our model.

The results after hyperparameter tuning are as follows:

```
ACCURACY SCORE :  0.805
Training accuracy:  0.924114671163575
Test accuracy :  0.805
ROC AUC SCORE :  0.6906249999999999
Cohen's Kappa Score : 0.3848580441640379

---------Confusion Matrix------------------

[[141  19]
 [ 20  20]]

---------Classification Report--------------

              precision    recall  f1-score   support

           0       0.88      0.88      0.88       160
           1       0.51      0.50      0.51        40

    accuracy                           0.81       200
   macro avg       0.69      0.69      0.69       200
weighted avg       0.80      0.81      0.80       200
```



The hyperparameter tuning model is not improving our model at all as it giving accuracy 0.81 and roc auc score of 0.69 which is slightly less than Ridge Classifier model we earlier trained. Also, AUC score has decreased by 0.1 earlier it was 0.84 now it is 0.83. So, we will stop here, and the ridge classifier model will be our final model.

Saving the model using Joblib library:

```
[384…] import joblib
[385…] joblib.dump(ridge, filename='Insurance_Fraud_Claim_Detection_Model.pkl')
[385…] ['Insurance_Fraud_Claim_Detection_Model.pkl']
```

6) Concluding Remarks:

- We have successfully developed and evaluated a machine learning model to predict whether the insurance claim is fraudulent or not?
- By using various machine learning algorithms like Logistic Regression, Ridge Classifier, Decision Tree, Random Forest, XGBOOST, CATBOOST and many other algorithms, we found out that the best model giving better overall performance in various metrics like Accuracy Score, ROC AUC SCORE and AUC score was none other than Ridge Classifier.
- Ridge Classifier gave an accuracy score of 0.82 or 82%, ROC AUC SCORE of 0.725 and AUC (Area under Curve) score of 0.84. So, our model gives very accurate and unbiased results.
- We performed different feature engineering techniques like SMOTE, Scaling the data using Standard Scaler, Power Transformer using yeo-Johnson method so that the model will learn better about the data.
- This model can predict fraudulent claims based on the dataset, but it is important to note that this is a historical data and in future the trends of fraudulent claims' data may change significantly, so our model may not be able to give such high accuracy.

- So, by creating this model, we have step foot forward in the right direction and our research will help and inspire others to build more robust and more powerful models in future to tackle the fraudulent claims which evolve in a better and efficient way.
- Finally, you can view and download the code we used to build this Machine Learning model from this link.