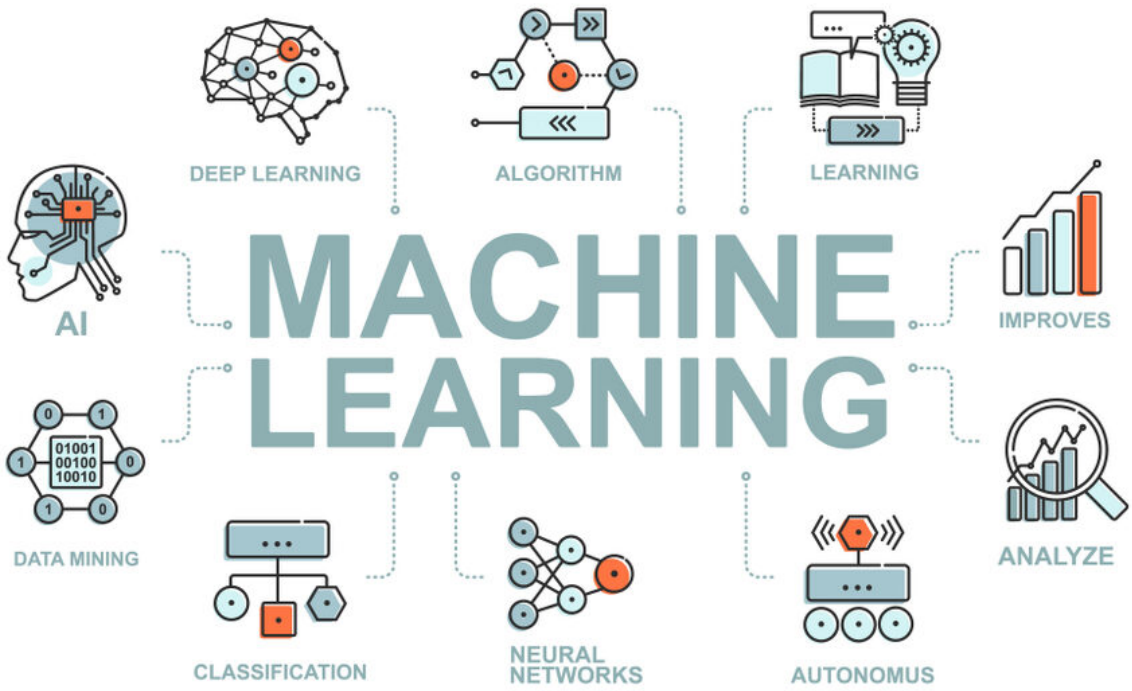
FINAL PROJECT



Name: Aishwary Shukla

Blog/Article: Project on Insurance Fraud Detection

Course: P.G. Program in Data Science

Institute: DataTrained Education

PROJECT: INSURANCE CLAIM FRAUD DETECTION

USING MACHINE LEARNING

PROBLEM IDENTIFICATION AND GOAL

Problem Statement:

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.

In this project, you are provided a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

The goal

Demonstrate how you can create a predictive model that predicts if an insurance claim is fraudulent or not.

ABSTRACT

Insurance fraud is a deliberate deception perpetrated against or by an insurance company or agent for financial gain. Fraud may be committed at different points by applicants, policyholders, third-party claimants, or professionals who provide services to claimants. Insurance agents and company employees may also commit insurance fraud. Common frauds include "padding" (inflating claims), misrepresenting facts on an insurance application, submitting claims for injuries or damage that never occurred, and staging accidents.

People who commit insurance fraud include:

- organized criminals who steal large sums through fraudulent business activities,
- professionals and technicians who inflate service costs or charge for services not rendered, and
- Ordinary people who want to cover their deductible or view filing a claim as an opportunity to make a little money.

Some insurance lines are more vulnerable to fraud than others. Healthcare, workers' compensation, and auto are generally considered the most affected insurance sectors.

PROCESS FOLLOWED AND STEPS

- Exploratory data analysis
- Feature Engineering
- Relation between different features/amenities of house and target column i.e. sales price.
- Correlation and identification of multicollinearity problem.
- Identifying and Selecting best features that affect sales price.
- Data pre-processing.
- Model Initialization and evaluation.
- Model Validation
- Model Testing and Pipelining.
- Prediction of test dataset values.

IMPORTING LIBRARIES:

Common libraries

- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns

Libraries for splitting training and testing data
Hyper parameter tuning

- from sklearn.model_selection import train_test_split,GridSearchCV

Importing Algorithms

- from sklearn.linear_model import LogisticRegression
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.ensemble import RandomForestClassifier
- from sklearn.metrics import confusion_matrix
- from sklearn.neighbors import KNeighborsClassifier
- from sklearn.svm import SVC
- from sklearn.naive_bayes import GaussianNB

Importing metrics

- from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
- from statsmodels.stats.outliers_influence import variance_inflation_factor

Removing warnings

```
import warnings

warnings.filterwarnings('ignore')

%matplotlib inline
```

DATASET DESCRIPTION SUMMARY AND LOADING DATASET

The dataset has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

File location:

file_loc = 'https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fraud.csv'

Reading CSV file:

df = pd.read_csv(file_loc)

FEATURE ENGINEERING

What is it and how we've used it?

Feature engineering or feature extraction or feature discovery is the process of using domain knowledge to extract features from raw data.

As the date column in the data set is mentioned in ddmmyy format so we are splitting the date into days months and year and then checking the insurance claims on the basis of these individual days months and year just check if there is any particular relation in between them and the target column.

```
# Feature Engineering:

# Date split

date_split=[]
for i in df['policy_bind_date']:
    date_split.append(i.split('-'))

day=[]
month=[]
year=[]

for i in range(0,1000):
    day.append(date_split[i][0])

for i in range(0,1000):
    month.append(date_split[i][1])

for i in range(0,1000):
    year.append(date_split[i][2])

df['Incident day'] = day

df['Incident month'] = month

df['Incident year'] = year

# Policy csl split

policy_split=[]
for j in df['policy_csl']:
    policy_split.append(j.split('/'))

policy_initial=[]
policy_ending=[]

for j in range(0,1000):
    policy_initial.append(policy_split[j][0])

for j in range(0,1000):
    policy_ending.append(policy_split[j][1])

df['policy_initial']=policy_initial
df['policy_ending']=policy_ending

df['policy_initial']= df['policy_initial'].astype('int64')
df['policy_ending']= df['policy_ending'].astype('int64')

# Replacing ? from police_report_available column:

df['police_report_available']

df.loc[df["police_report_available"] == "?", "police_report_available"] = "Unknown"
```

DATA ANALYSIS

As preliminary data analysis we are first inspecting the first and last 10 rows of the data set to check the different column values and row entries.

Using head and tail method dtypes method and isnal.sum method we are inspecting data types of columns and if null values are present in the columns.

As we can see that there are no null values in the data set that is a good sign but further we need to check the statistical description of the data set and we need to inspect the presence of outliers in the data set.

Segregating the categorical data types and numerical data types is a good idea and will help us in doing exploratory data analysis on the basis of data types of columns.

As categorical columns and numerical columns need to be inspected and analysed with different approaches.

- First and last 10 rows:

```
In [4]: print('First 10 rows')
        df.head(10)
```

First 10 rows

Out[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	17-10-2014	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	27-06-2006	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	06-09-2000	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	06-06-2014	IL	500/1000	1000	1583.91	6000000	610706
5	256	39	104594	12-10-2006	OH	250/500	1000	1351.10	0	478456
6	137	34	413978	04-06-2000	IN	250/500	1000	1333.35	0	441716
7	165	37	429027	03-02-1990	IL	100/300	1000	1137.03	0	603195
8	27	33	485665	05-02-1997	IL	100/300	500	1442.99	0	601734
9	212	42	636550	25-07-2011	IL	100/300	500	1315.68	0	600983

10 rows × 40 columns

```
In [5]: print('Last 10 rows')
        df.tail(10)
```

Last 10 rows

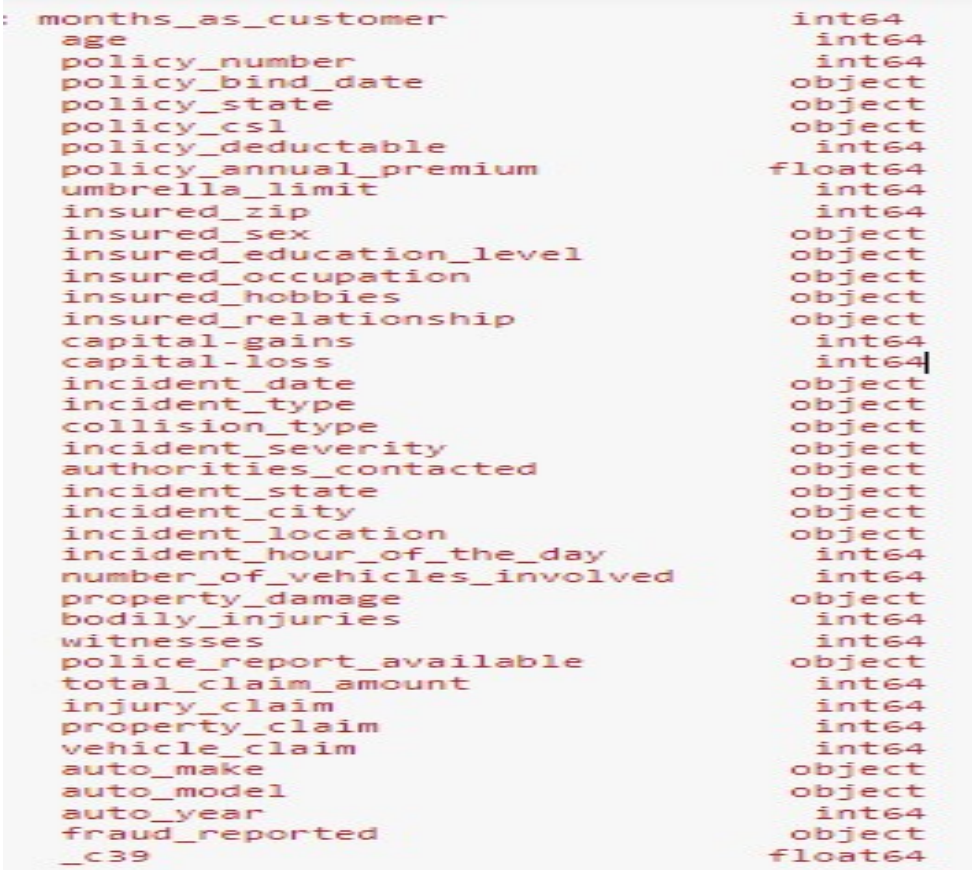
Out[5]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
990	286	43	663190	05-02-1994	IL	100/300	500	1564.43	3000000	477644
991	257	44	109392	12-07-2006	OH	100/300	1000	1260.88	0	43398
992	94	26	215278	24-10-2007	IN	100/300	500	722.66	0	43389
993	124	28	674570	08-12-2001	OH	250/500	1000	1235.14	0	44356
994	141	30	681486	24-03-2007	IN	500/1000	1000	1347.04	0	43066
995	3	38	941851	16-07-1991	OH	500/1000	1000	1310.80	0	43128
996	285	41	186934	05-01-2014	IL	100/300	1000	1436.79	0	60817
997	130	34	918516	17-02-2003	OH	250/500	500	1383.49	3000000	44279
998	458	62	533940	18-11-2011	IL	500/1000	2000	1356.92	5000000	44171
999	456	60	556080	11-11-1996	OH	250/500	1000	766.19	0	61226

- Rows and columns in dataset: (1000, 40)
- Column names:

  ['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', '_c39']

  - **Dtypes of columns**:



```
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
```

- **Null values in each column:**

```
In [53]: null_data = df.isnull().sum()
         null_data.sort_values(ascending=0)
```

```
Out[53]: months_as_customer              0
         age                             0
         incident_location               0
         incident_hour_of_the_day        0
         number_of_vehicles_involved     0
         property_damage                 0
         bodily_injuries                 0
         witnesses                       0
         police_report_available         0
         total_claim_amount              0
         injury_claim                    0
         property_claim                  0
         vehicle_claim                   0
         auto_make                       0
         auto_model                      0
         auto_year                       0
         fraud_reported                  0
         Incident day                    0
         Incident month                  0
         Incident year                   0
         policy_initial                  0
         incident_city                   0
         incident_state                  0
         authorities_contacted           0
         insured_sex                     0
         policy_number                   0
         policy_bind_date                0
         policy_state                    0
         policy_csl                      0
         policy_deductable               0
         policy_annual_premium           0
         umbrella_limit                  0
         insured_zip                     0
         insured_education_level         0
         incident_severity               0
         insured_occupation              0
         insured_hobbies                 0
         insured_relationship            0
         capital-gains                   0
         capital-loss                    0
         incident_date                   0
         incident_type                   0
         collision_type                  0
         policy_ending                   0
         dtype: int64
```

EXPLORATORY DATA ANALYSIS

- **Summary statistics of numerical columns:**

```
In [55]: desc = df.describe().T
         desc['range']=desc['max']-desc['min']
         desc
```

Out[55]:

	count	mean	std	min	25%	50%	75%	max	range
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00	45.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00	898631.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00	1500.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59	1614.26
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00	11000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00	190858.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00	111100.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00	3.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00	3.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00	114820.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00	79490.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00	20.00
policy_initial	1000.0	2.726500e+02	1.616032e+02	100.00	100.0000	250.0	500.000	500.00	400.00
policy_ending	1000.0	5.802000e+02	2.874205e+02	300.00	300.0000	500.0	1000.000	1000.00	700.00

- **Summary statistics of categorical columns:**

```
df.describe(include='object').T
```

	count	unique	top	freq
policy_bind_date	1000	951	01-01-2006	3
policy_state	1000	3	OH	352
policy_csl	1000	3	250/500	351
insured_sex	1000	2	FEMALE	537
insured_education_level	1000	7	JD	161
insured_occupation	1000	14	machine-op-inspct	93
insured_hobbies	1000	20	reading	64
insured_relationship	1000	6	own-child	183
incident_date	1000	60	02-02-2015	28
incident_type	1000	4	Multi-vehicle Collision	419
collision_type	1000	4	Rear Collision	292
incident_severity	1000	4	Minor Damage	354
authorities_contacted	1000	5	Police	292
incident_state	1000	7	NY	262
incident_city	1000	7	Springfield	157
incident_location	1000	1000	9935 4th Drive	1
property_damage	1000	3	?	360
police_report_available	1000	3	Unknown	343
auto_make	1000	14	Saab	80
auto_model	1000	39	RAM	43
fraud_reported	1000	2	N	753
incident day	1000	31	28	48
incident month	1000	12	12	95
incident year	1000	26	1991	55

- Selecting categorical and numerical columns:

```
In [58]: cont_data = df.select_dtypes(include=['int64','float64'])

         cat_data= df.select_dtypes(include=['object'])

         cont_columns = cont_data.columns

         cat_columns = cat_data.columns
```

```
In [59]: cont_columns
```

```
Out[59]: Index(['months_as_customer', 'age', 'policy_number', 'policy_deductable',
                'policy_annual_premium', 'umbrella_limit', 'insured_zip',
                'capital-gains', 'capital-loss', 'incident_hour_of_the_day',
                'number_of_vehicles_involved', 'bodily_injuries', 'witnesses',
                'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_claim',
                'auto_year', 'policy_initial', 'policy_ending'],
               dtype='object')
```

```
In [60]: cat_columns
```

```
Out[60]: Index(['policy_bind_date', 'policy_state', 'policy_csl', 'insured_sex',
                'insured_education_level', 'insured_occupation', 'insured_hobbies',
                'insured_relationship', 'incident_date', 'incident_type',
                'collision_type', 'incident_severity', 'authorities_contacted',
                'incident_state', 'incident_city', 'incident_location',
                'property_damage', 'police_report_available', 'auto_make', 'auto_model',
                'fraud_reported', 'Incident day', 'Incident month', 'Incident year'],
               dtype='object')
```

Data Distribution Analysis:

Data distribution of the data set is very important to determine the frequency distribution of the numerical columns it also helps us to analyse the data distribution of columns and find the skewness if it is present in the data set.

Here we have used histograms density plots box plots distribution plots and violent plots to analyse the data distribution and frequency of the data set at large.

Skewness can lead to large error values does they need to be figured out and further they need to be treated.

In [ ]:
```
# Univariate Summary plots
```

In [ ]:
```
# Histogram plot:

for i in cont_columns:
    plt.figure(figsize=(10,6),facecolor='orange')
    plt.hist(df[i],bins=10)
    plt.xlabel(i)
    plt.show()
```

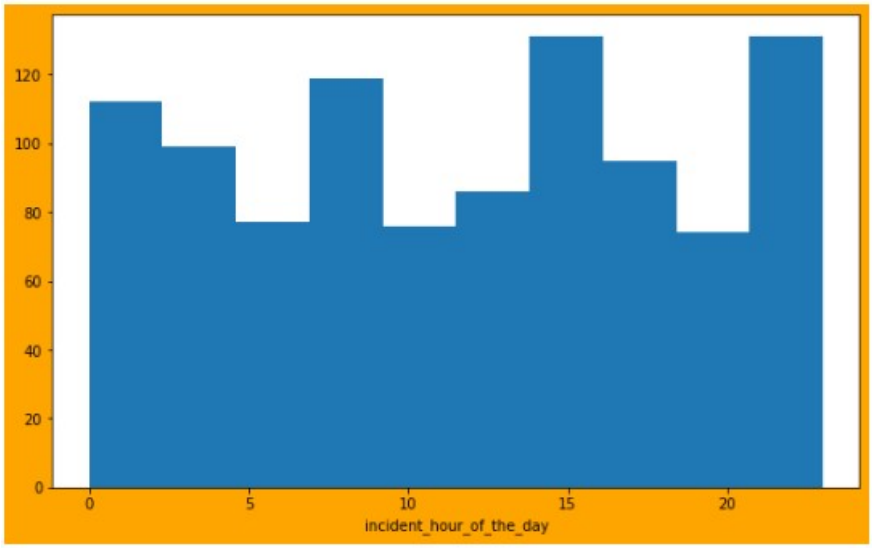
In [ ]:
```
# Densityplot

for i in cont_columns:
    plt.figure(figsize=(10,6),facecolor='orange')
    sns.kdeplot(df[i], shade=True)
    plt.xlabel(i)
    plt.show()
```

In [ ]:
```
# Boxplots

for i in cont_columns:
    plt.figure(figsize=(10,6),facecolor='orange')
    sns.boxplot(df[i])
    plt.xlabel(i)
    plt.show()
```

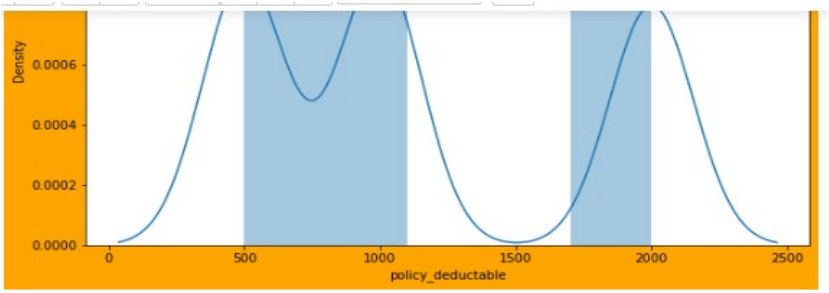
In [ ]:
```
# Distplot

for i in cont_columns:
    plt.figure(figsize=(10,6),facecolor='orange')
    sns.distplot(df[i])
    plt.xlabel(i)
    plt.show()
```

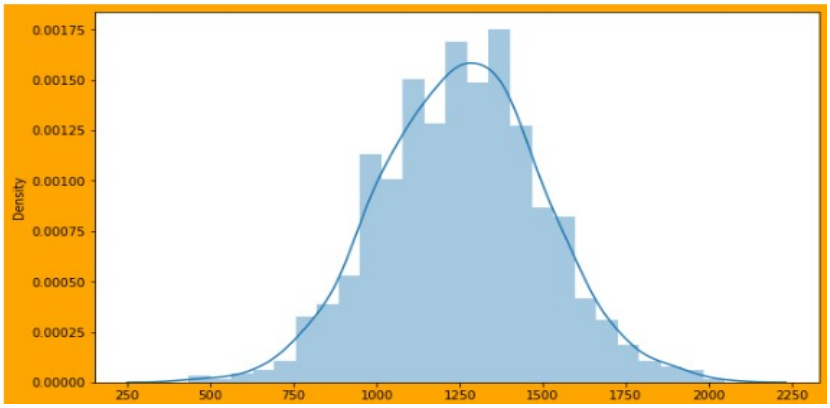
In [ ]:
```
# Violinplot

for i in cont_columns:
    plt.figure(figsize=(10,6),facecolor='skyblue')
    sns.violinplot(df[i],showmedians=True)
    plt.xlabel(i)
    plt.show()
```

1. Histogram.

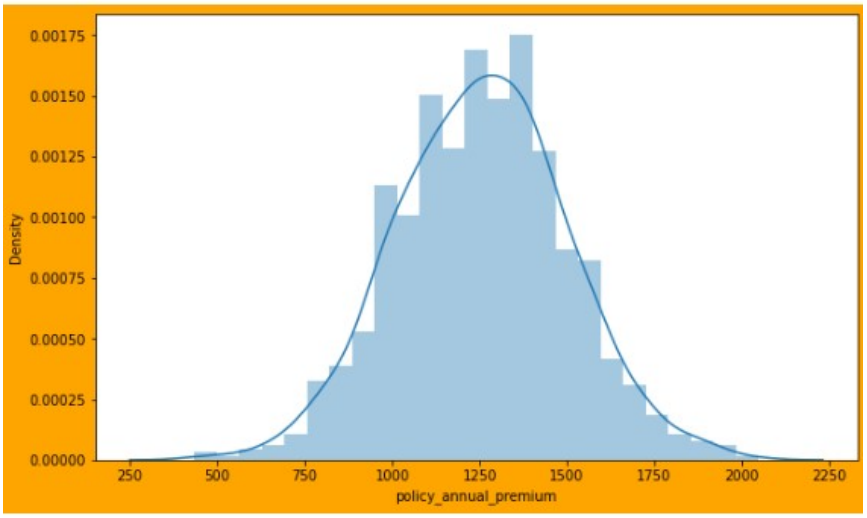


2. KdePlot

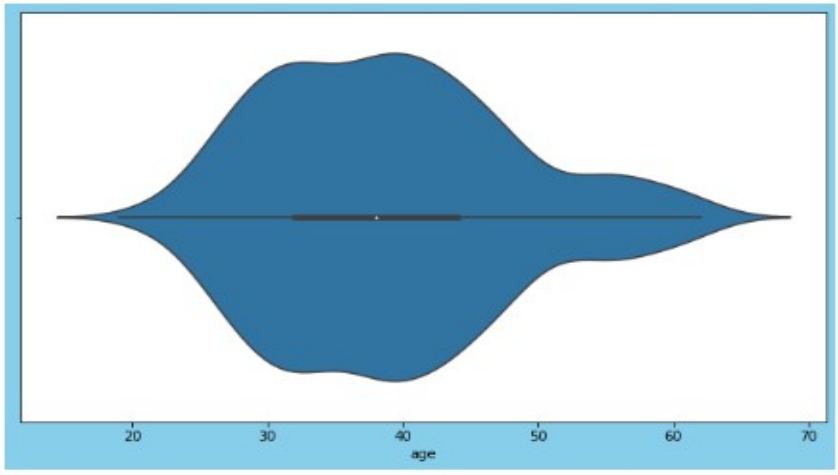


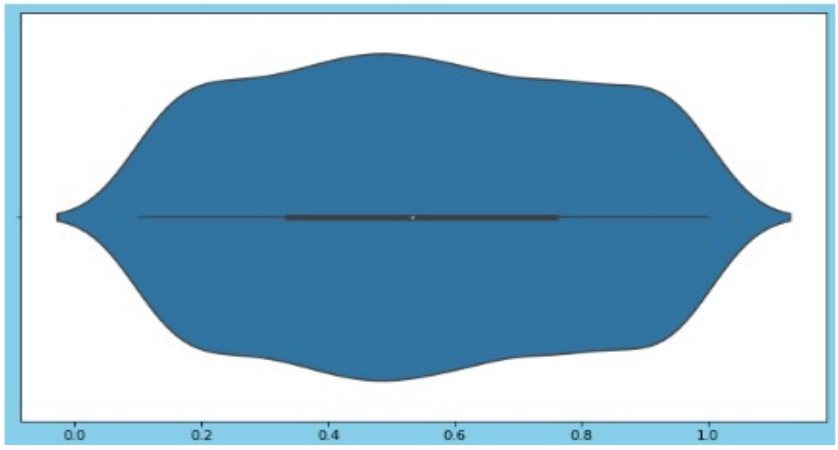


3. Distribution Plot

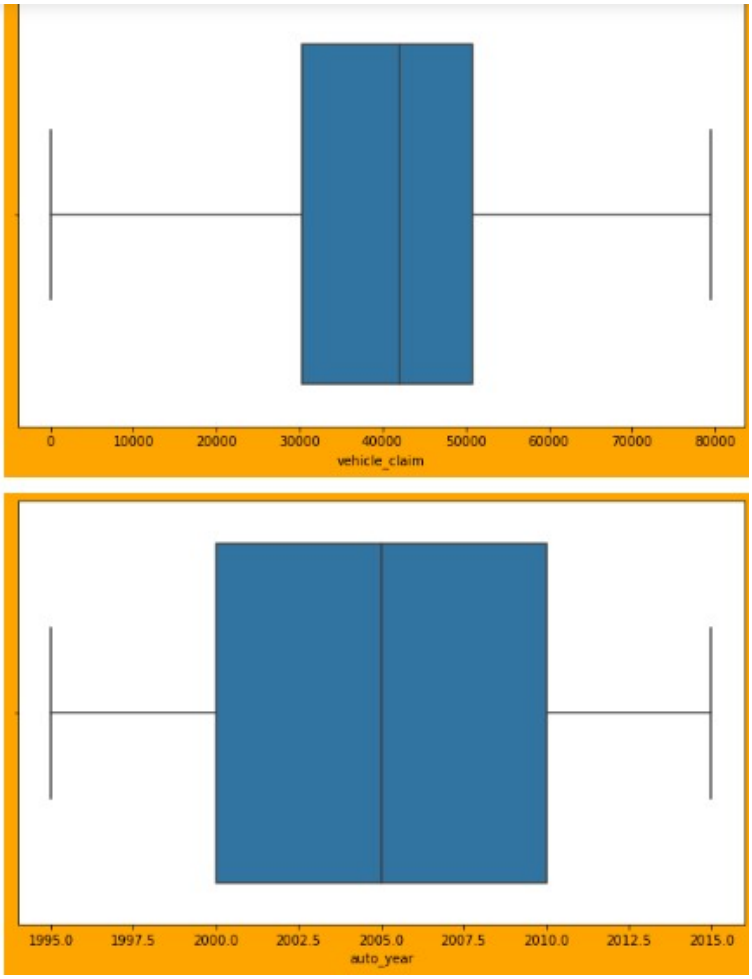


4. Violin plot.





Outlier Analysis using BoxPlot



UNIVARIATE ANALYSIS

What is it and how we've used it?

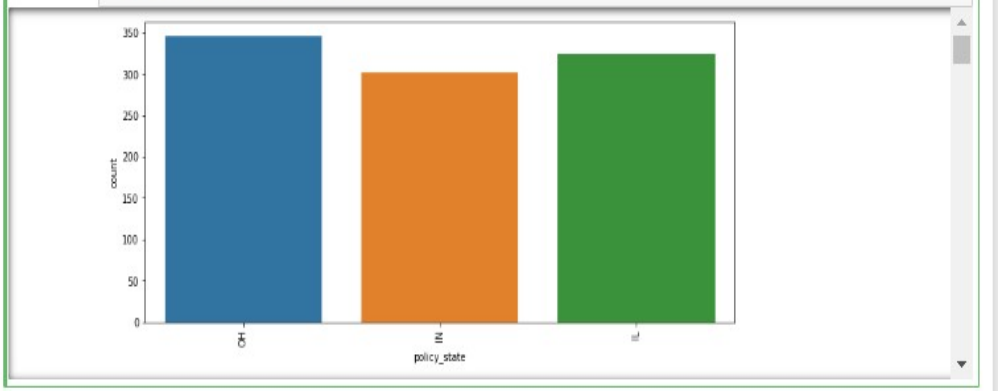
Univariate Analysis is a type of data visualization where we visualize only a single variable at a time. Univariate Analysis helps us to analyze the distribution of the variable present in the data so that we can perform further analysis.

All data distribution plots belong to analysing only one column at a time apart from them we have used count plot line plot and scatter plot strip plot and swarmplot to analyse one column at a time.

1. Count plot
2. Lineplot
3. Scatterplot

```
In [ ]: # Countplot
```

```
In [145]: for i in ['policy_state', 'policy_csl', 'insured_sex',
                    'insured_education_level', 'insured_occupation', 'insured_hobbies',
                    'insured_relationship', 'incident_date', 'incident_type',
                    'collision_type', 'incident_severity', 'authorities_contacted',
                    'incident_state', 'incident_city','property_damage', 'police_report_available', 'auto_make', 'auto_model',
                    'fraud_reported']:
              f= plt.figure(figsize=(12,5))
              ax = sns.countplot(x=df[i],data=df)
              plt.xticks(rotation=90)
```




```
In [ ]: # Scatterplot

for i in cont_columns:
    plt.figure(figsize=(12,8))
    sns.scatterplot(x = df.index, y = i, data = df)
    plt.xticks(rotation=90)
    plt.show()
```

```
In [ ]: # Lineplot

for i in cont_columns:
    plt.figure(figsize=(12,8))
    sns.lineplot(x = df.index, y = i, data = df)
    plt.xticks(rotation=90)
    plt.show()
```

```
In [ ]: # Stripplot

for i in cont_columns:
    plt.figure(figsize=(12,8))
    sns.stripplot(x = i, data = df)
    plt.xticks(rotation=90)
    plt.show()
```

```
In [ ]: # SwarmPlot

for i in cont_columns:
    plt.figure(figsize=(12,8))
    sns.swarmplot(x = i, data = df)
    plt.xticks(rotation=90)
    plt.show()
```

```
In [ ]: # Univariate Summary plots
```

BIVARIATE ANALYSIS:

What is it and how we've used it?

Bivariate analysis is the simultaneous analysis of two variables. It explores the concept of the relationship between two variable whether there exists an association and the strength of this association or whether there are differences between two variables and the significance of these differences.

For analysing two columns at a time we have used categorical plots and box plot with you value as whether fraud reported or not to check indepth of frequencies for different columns at which fraud was reported or not.

In [ ]:
```
# Bivariate analysis
```

In [ ]:
```
for i in cat_columns:
    plt.figure(figsize=(12,50))
    sns.catplot(x=i, kind="count",hue = 'fraud_reported', data=df)
    plt.xticks(rotation=90)
    plt.show()
```

In [ ]:
```
# Scatterplot

for i in ['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']:
    for j in ['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']:
            plt.figure(figsize=(12,8))
            sns.scatterplot(x = i,y=j, data = df,hue='fraud_reported')
            plt.xticks(rotation=90)
            plt.show()
```

In [ ]:
```
# Lineplot

for i in ['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']:
    for j in ['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']:
        plt.figure(figsize=(12,8))
        sns.lineplot(x = df.index, y = i, data = df,hue='fraud_reported')
        plt.xticks(rotation=90)
        plt.show()
```

OUTLIER ANALYSIS USING BOXPLOT:

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='auto_model', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='auto_make', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='police_report_available', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='property_damage', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='incident_location', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='incident_city', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='incident_state', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='authorities_contacted', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='incident_severity', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='collision_type', y = i, data = df, hue ='fraud_reported')
```

In [ ]:
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='incident_type', y = i, data = df, hue ='fraud_reported')
```

ANALYSIS USING SWARM PLOT

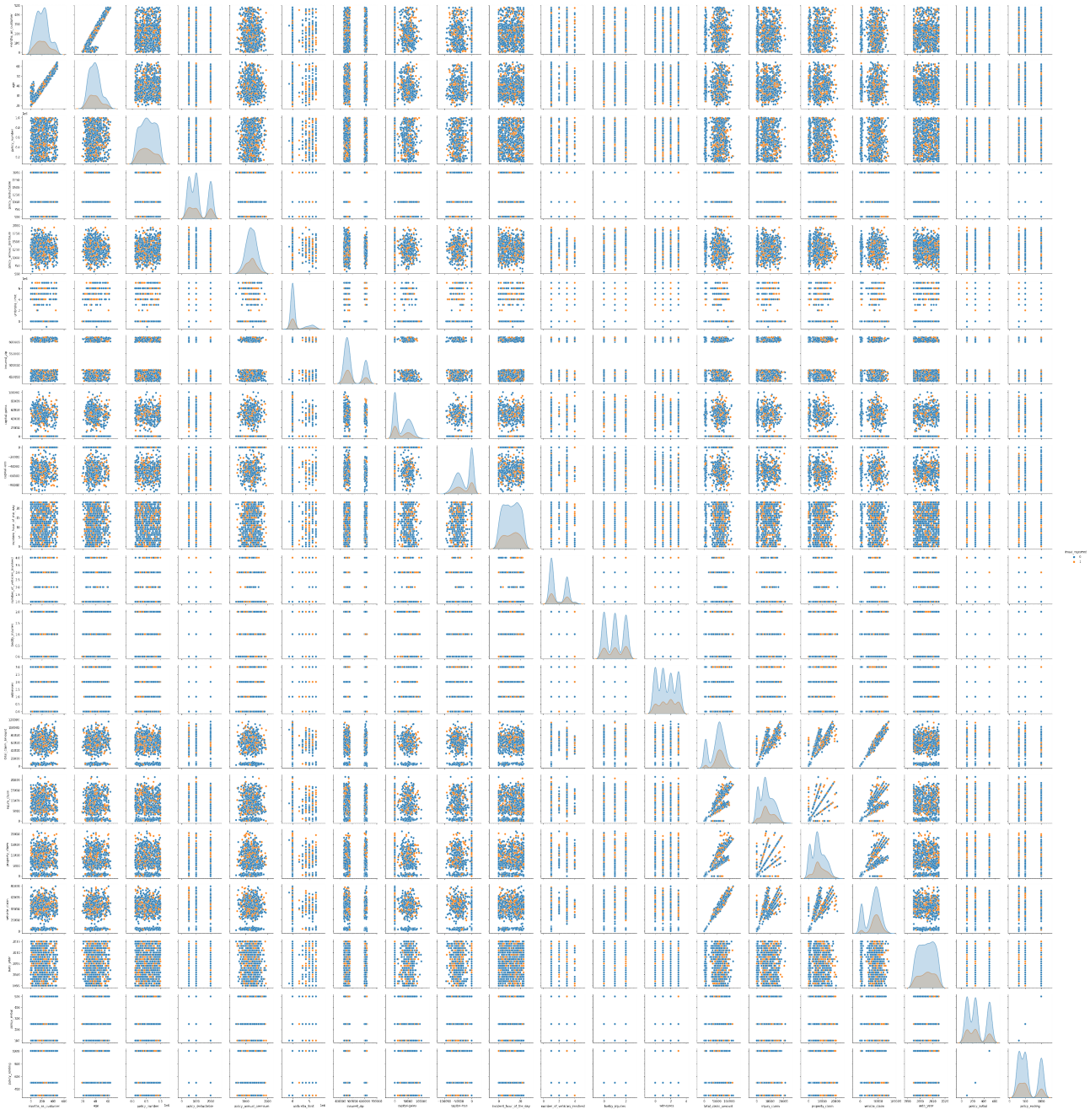
```
In [ ]: # Swarmplot
In [ ]: sns.swarmplot(x = 'age', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'policy_number', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'policy_deductable', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'policy_annual_premium', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'umbrella_limit', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'insured_zip', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'capital-gains', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'capital-loss', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'incident_hour_of_the_day', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'number_of_vehicles_involved', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'bodily_injuries', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'total_claim_amount', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'injury_claim', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'property_claim', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'vehicle_claim', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'auto_year', y = "fraud_reported", palette = "Set2", data = df)
In [ ]: sns.swarmplot(x = 'months_as_customer', y = "fraud_reported", palette = "Set2", data = df)
```

Multivariate Analysis

What is it and how we've used it?

It is an extension of bivariate analysis which means it involves multiple variables at the same time to find correlation between them. Multivariate Analysis is a set of statistical model that examine patterns in multidimensional data by considering at once, several data variable.

We have done multivariate analysis using pair plot and correlation plot such as heat maps to understand the correlation between different columns and to check multi collinearity problem as well.



Preprocessing

What is it and how we've used it?

Label Encoding refers to **converting the labels into a numeric form so as to convert them into the machine-readable form**. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

Using label encoder to encode label column:

```
: # Import label encoder
  from sklearn import preprocessing

  # label_encoder object knows how to understand word labels.
  label_encoder = preprocessing.LabelEncoder()

  # Encode labels in column 'species'.
  df['fraud_reported']= label_encoder.fit_transform(df['fraud_reported'])

  df['fraud_reported'].unique()

: array([1, 0])
```

Removing outliers values using **Z-Score method**

What is it and how we've used it?

Z-score tells how many standard deviations away a given observation is from the mean. For example, a Z score of 2.5 means that the data point is 2.5 standard deviation far from the mean. And since it is far from the center, it's flagged as an outlier/anomaly.

Here we have chosen absolute z score value equal to 2.8 so all the values lying in 2.8 times standard deviation will be removed.

```
# Using Z Statistics to check and remove any more outliers:

from scipy.stats import zscore

z_score = zscore(df[cont_columns])

abs_z_score = np.abs(z_score)

filtering_entry = (abs_z_score < 2.8).all(axis=1) # values lying in 3 times std will be removed

df = df[filtering_entry]

df.describe()
```

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-loss
count	971.000000	971.000000	971.000000	971.000000	971.000000	9.710000e+02	971.000000	971.000000	971.000000
mean	205.531411	39.018538	546085.284243	1136.972194	1258.575675	9.907312e+05	500583.408857	25319.876416	-26652.317199
std	115.387093	9.199040	257987.549458	611.136088	239.082766	2.121300e+06	71349.604532	27966.988021	27870.835407
min	0.000000	19.000000	100804.000000	500.000000	617.110000	-1.000000e+06	430104.000000	0.000000	-93600.000000
25%	119.000000	32.000000	335264.500000	500.000000	1090.175000	0.000000e+00	448451.000000	0.000000	-51250.000000
50%	202.000000	38.000000	533941.000000	1000.000000	1257.830000	0.000000e+00	466303.000000	0.000000	-24100.000000
75%	279.000000	45.000000	760439.500000	2000.000000	1416.160000	0.000000e+00	602892.000000	51200.000000	0.000000

Checking correlation and addressing Multicollinearity problem:

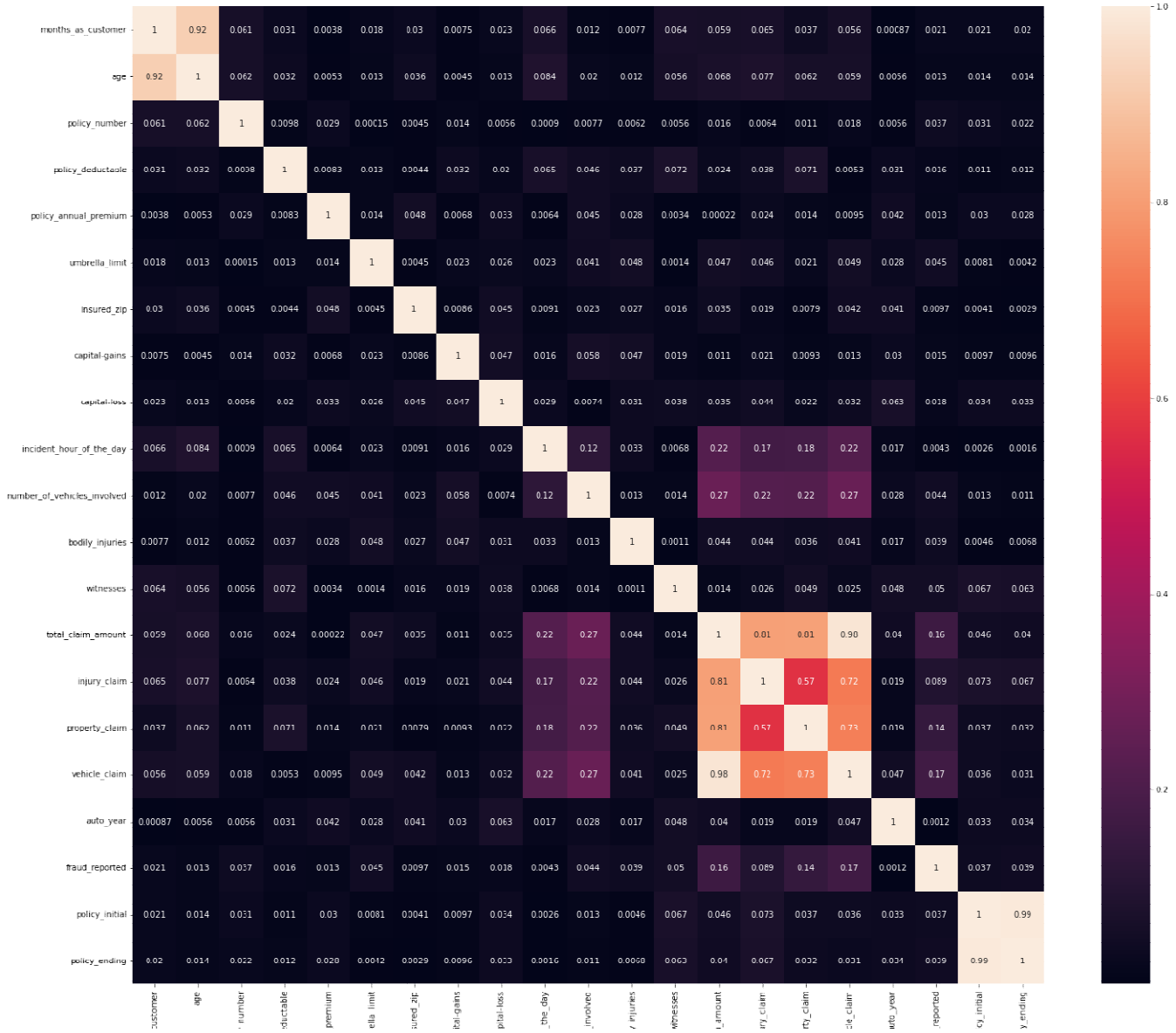
What is it and how we've used it?

Heatmap analysis:

A Heat map is a graphical representation of multivariate data that is structured as a matrix of columns and rows.

Heat maps are very useful in describing correlation among several numerical variables, visualizing patterns and anomalies.

Using heat map the correlation values of different columns show that vehicle claim property claim injury claim and total claim amount columns are highly correlated with each other thus it would be necessary to use variance inflation factor that will help us to measure the amount of multi collinearity in the data set columns.



Multicollinearity check using Variance Inflation Factor:

What is it and how we've used it?

A variance inflation factor (VIF) is a measure of the amount of multicollinearity in regression analysis**. Multicollinearity exists when there is a correlation between**

multiple independent variables in a multiple regression model. This can adversely affect the regression results.

Vif values of different columns show that the multi collinearity problem exist in the data set that needs to be corrected before further model initialization and testing.

Two eliminate the multicolineality problem in the data prep processing stage we have used power transformation and we have done principle component analysis to select the best principal components for predicting the target column that is whether fraud was reported or not in a particular insurance claim.

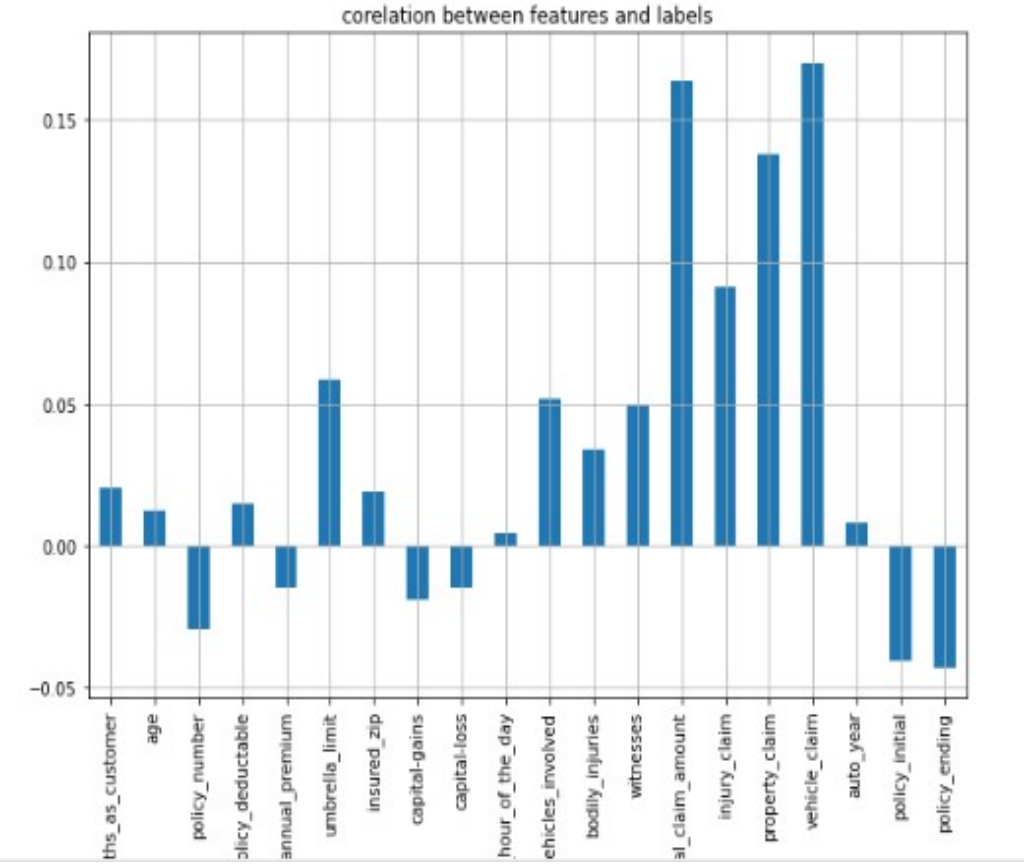
```
from statsmodels.stats.outliers_influence import variance_inflation_factor


vif = pd.DataFrame()
vif['vif'] = [variance_inflation_factor(df[cont_columns],i) for i in range(df[cont_columns].shape[1])]
vif['features'] = df[cont_columns].columns
vif.sort_values(by='vif',ascending=False)
```

	vif	features
16	inf	vehicle_claim
15	inf	property_claim
14	inf	injury_claim
13	inf	total_claim_amount
19	480.745005	policy_ending
18	364.432233	policy_initial
17	154.238734	auto_year
1	134.565858	age
6	50.676813	insured_zip
0	29.481936	months_as_customer
4	29.084805	policy_annual_premium
2	5.563358	policy_number
10	4.657641	number_of_vehicles_involved
3	4.573494	policy_deductable
9	4.112863	incident_hour_of_the_day
12	2.877312	witnesses
11	2.526997	bodily_injuries
8	1.941466	capital-loss
7	1.843121	capital-gains
5	1.231494	umbrella_limit

Using correlation function we can see that columns namely umbrella limit vehicles involved bodily injuries witnesses claim amount injury claim property claim and vehicle claim are positively correlated with the target column that is fraud reported.

Feature Importance

Feature (variable) importance indicates how much each feature contributes to the model prediction**. Basically, it determines the degree of usefulness of a specific variable for a current model and prediction.**



In building this model we have followed two approaches with variations in:
1.Outlier removal.
2. PCA selection.
3. Variation in train test split size.
4. Opting Oversampling and Undersampling techniques.

Feature Selection and Data Preprocessing

1. **Feature and Label Selection:**

2. **2. Scaling Data using Power Transformer:**

What is it and how we've used it?

In statistics, a power transform is a family of functions applied to create a monotonic transformation of data using power functions. It is a data transformation technique used to stabilize variance, make the data more normal distribution-like, improve the validity of measures of association (such as the Pearson correlation between variables), and for other data stabilization procedures.

```
# Splitting data into features and label:

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

```
X_Dums = pd.get_dummies(X)
```

```
from sklearn.preprocessing import PowerTransformer

pt = PowerTransformer()

X_trans = pt.fit_transform(X_Dums)
```

Analyzing Principal component using PCA:

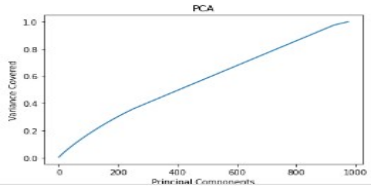
What is it and how we've used it?

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss**. It does so by creating new uncorrelated variables that successively maximize variance.**

```
# Using PCA i.e. Principal Component Analysis that is a diamensionality reduction technique:

pcaf = PCA()
pcaf.fit_transform(X_trans)
```

```
array([[-1.30971962e+00,  1.33482310e-01,  7.57761399e-01, ...,
         1.41590396e-02, -1.29393191e-02,  4.64273286e-15],
       [ 5.60776600e+00, -7.71105795e-01,  1.38516964e+00, ...,
        -1.20047229e-02, -2.38149054e-01,  4.64273286e-15],
       [-7.78243765e-02, -1.72068029e+00, -2.57087449e+00, ...,
         8.32111761e-02, -5.05516250e-02,  4.64273286e-15],
       ...,
       [-4.82355787e-01,  1.56383999e+00, -2.93418072e+00, ...,
        -1.04846125e-02,  1.26224312e-01,  4.64273286e-15],
       [-3.33320088e-02,  2.77876293e+00,  1.62522756e+00, ...,
        -1.57361505e-01, -2.39759732e-01,  4.64273286e-15],
       [ 5.67022201e+00, -1.11448066e-01,  1.64458403e+00, ...,
         9.72843297e-02, -9.01971673e-02,  4.64273286e-15]])
```

```
# Using Scree Plot to identify best components:

plt.figure()
plt.plot(np.cumsum(pcaf.explained_variance_ratio_))
plt.xlabel('Principal Components')
plt.ylabel('Variance Covered')
plt.title('PCA')
plt.show()
```




```
from sklearn.decomposition import PCA

pcaf = PCA(n_components=975)
new_pcompf = pcaf.fit_transform(X_trans)
princi_compf = pd.DataFrame(new_pcompf)
princi_compf
```

	0	1	2	3	4	5	6	7	8	9	...	965	966	967	968
0	-1.309720	0.133482	0.757761	2.399679	3.704364	3.738203	-0.312367	1.012493	-0.681989	-2.005104	...	0.231128	-0.364810	-0.046190	0.010564
1	5.607766	-0.771106	1.385170	4.620983	-2.087348	-3.752877	-0.883807	0.961549	-0.548975	0.261650	...	-0.433777	0.219293	-0.022444	-0.124805
2	-0.077824	-1.720680	-2.570874	-1.798903	-1.359451	-2.869600	1.642317	-0.973030	-2.291834	2.006154	...	-0.151102	-0.171215	0.144241	-0.019959
3	-0.122925	-0.230663	1.064716	-0.073337	2.061772	-0.735124	2.082378	-1.047627	-0.886793	-1.460831	...	3.514980	4.127619	4.182149	2.169624
4	5.907659	2.766240	1.041849	1.226948	-0.745193	1.078651	3.987781	2.738602	-3.052439	0.789989	...	0.063264	0.091602	-0.186550	-0.078033
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
570	-1.731542	3.188425	3.984575	-4.629630	1.431930	-0.653960	-2.372687	1.943429	-2.263077	-1.021289	...	0.134173	0.099529	-0.005679	0.010780
571	-4.239532	-2.408686	1.427368	0.182542	0.398235	2.811907	-0.019060	-1.894528	-0.202830	-2.263880	...	0.117859	-0.300170	-0.012801	0.035613
572	-0.482356	1.563840	-2.934181	-1.690244	1.385622	-1.374636	-1.961950	-1.705775	4.369341	0.601252	...	-0.132883	0.030039	0.038449	0.271242
573	-0.033332	2.778763	1.625228	1.840153	4.232328	0.770508	-3.533238	0.539969	-0.244279	3.438413	...	-0.022428	-0.091980	0.030037	0.147566
574	5.670222	-0.111448	1.644584	4.930580	0.124563	-3.917780	-1.738405	-0.048677	0.647385	-2.074305	...	-0.246432	0.238728	0.017992	-0.108936

975 rows × 975 columns

Splitting Features and Labels into training and testing dataset:

What is it and how we've used it?

The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train,X_test , y_train and y_test. X_train and y_train sets are used for training and fitting the model.

Further we are splitting our data set and we are keeping 30% data result for the testing part and 70% data for the training part.

```
# Splitting our data to training data and testing data
# x_train,x_test,y_train,y_test

X_train,X_test,y_train,y_test = train_test_split(princi_compf,y,test_size=0.30,random_state=1)

# Here we are keeping training data as our scalled data and testing data as our label or target.
```

Treating imbalanced class using SMOTE and Near Miss methods:

What is it and how we've used it?

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem.
It aims to balance class distribution by randomly increasing minority class examples by replicating them.
By checking the unique value counts of the target column we can see that the values in the target column are categorical in nature and does they need to be and encoded. Does we are using label and coder to encode the label column that's simply converts the categorical value is to numerical values.

By checking the value counts of each class in label we can see that there are 5004 value counts for class where no fraud is reported. But we have only 178 value counts for which fraud is reported and this weekend conclude that the data set is imbalanced and does we need to correct this imbalance data set with the help of over sampling technique. That's we are using smoke that is synthetic minority over sampling technique with the help of which we will balance the data set.

```
from imblearn.over_sampling import SMOTE

print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0)))
```

```
Before OverSampling, counts of label '1': 178
Before OverSampling, counts of label '0': 504

After OverSampling, the shape of train_X: (1008, 975)
After OverSampling, the shape of train_y: (1008,)

After OverSampling, counts of label '1': 504
After OverSampling, counts of label '0': 504
```

```
arr = np.array(y_train_res)

un, co = np.unique(arr,return_counts=True)

dict(zip(un,co))
```

```
{0: 504, 1: 504}
```

```
# Using near miss undersampling technique:

print("Before Undersampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before Undersampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# apply near miss
from imblearn.under_sampling import NearMiss
nr = NearMiss()

X_train_miss, y_train_miss = nr.fit_resample(X_train, y_train.ravel())

print('After Undersampling, the shape of train_X: {}'.format(X_train_miss.shape))
print('After Undersampling, the shape of train_y: {} \n'.format(y_train_miss.shape))

print("After Undersampling, counts of label '1': {}".format(sum(y_train_miss == 1)))
print("After Undersampling, counts of label '0': {}".format(sum(y_train_miss == 0)))
```

```
Before Undersampling, counts of label '1': 199
Before Undersampling, counts of label '0': 581
```

Building Machine Learning Models and Model Performance

Thus after splitting the data set and balancing the data set we are proceeding towards model initialization part and further we will train and test our model and we will check the accuracy of our model and then we will select the model that is giving us the best accuracy further we will also plot the aucroc curve to evaluate the performance in classification models.

In all this will be initializing logistic regression model decision tree classifier model random forest classifier model k neighbors classifier model support vector classifier model gradient boosting classifier model at a post classifier model and nav base classifier model.

We will also do hyperparameter tuning of all these models and will do the cross validation using k folds method.

Screenshots of all initialized models are indicated below:

Different modes are built to classify Insurance cases being fraudulent or genuine:

1. Logistic Regression

```
# Logistic Regression:

log_reg = LogisticRegression(random_state=1)

log_reg.fit(X_train_res,y_train_res)

pred_train = log_reg.predict(X_train_res)

y_pred = log_reg.predict(X_test)

acc = accuracy_score(y_test,y_pred)

print('Training accuracy: ', accuracy_score(y_train_res,pred_train)*100)
print('Testing accuracy: ', acc*100)

confusion_mat = confusion_matrix(y_test,y_pred)

print('Confusion matrix: \n',confusion_mat)

clr = classification_report(y_test,y_pred)

print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  81.56996587030717
Confusion matrix:
 [[230   3]
 [ 51   9]]
classification report:               precision    recall  f1-score   support

           0       0.82      0.99      0.89       233
           1       0.75      0.15      0.25        60

    accuracy                           0.82       293
   macro avg       0.78      0.57      0.57       293
weighted avg       0.80      0.82      0.76       293
```

2. Decision Tree Classifier

```
# Decision Tree Classifier:

dtc = DecisionTreeClassifier(random_state=1)
dtc.fit(X_train_res,y_train_res)
pred_train_dtc = dtc.predict(X_train_res)
y_pred = dtc.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_dtc)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  60.40955631399317
Confusion matrix:
 [[160  73]
 [ 43  17]]
classification report:                precision    recall  f1-score   support

           0       0.79      0.69      0.73       233
           1       0.19      0.28      0.23        60

    accuracy                           0.60       293
   macro avg       0.49      0.49      0.48       293
weighted avg       0.67      0.60      0.63       293
```

3. Random Forest Classifier

```
# Random forrest classifier model:

rfc_f = RandomForestClassifier()
rfc_f.fit(X_train_res,y_train_res)
pred_train_rfc_f = rfc_f.predict(X_train_res)
y_pred = rfc_f.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_rfc_f)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  79.5221843003413
Confusion matrix:
 [[232   1]
 [ 59   1]]
classification report:               precision    recall  f1-score   support

           0       0.80      1.00      0.89       233
           1       0.50      0.02      0.03        60

    accuracy                           0.80       293
   macro avg       0.65      0.51      0.46       293
weighted avg       0.74      0.80      0.71       293
```

4. K-Neighbours Classifier

```
# KNN classifier:

knn = KNeighborsClassifier()
knn.fit(X_train_res,y_train_res)
pred_train_knn = knn.predict(X_train_res)
y_pred = knn.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_knn)*100)
print('Testing accuracy: ', acc*100)
```

```
Training accuracy:  50.0
Testing accuracy:  23.46938775510204
```

5. Support Vector Classifier

```
# Using best parameters for improved score:

svc = SVC()

svc.fit(X_train_res,y_train_res)

pred_train_svc = svc.predict(X_train_res)
y_pred = svc.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_svc)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  76.53061224489795
Confusion matrix:
 [[225   0]
 [ 69   0]]
classification report:                precision    recall  f1-score   support

           0       0.77      1.00      0.87       225
           1       0.00      0.00      0.00        69

    accuracy                           0.77       294
   macro avg       0.38      0.50      0.43       294
weighted avg       0.59      0.77      0.66       294
```

6. Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

# GBC

gbdt_clf = GradientBoostingClassifier(random_state=1)
gbdt_clf.fit(X_train_res,y_train_res)
pred_train_gbdt_clf = gbdt_clf.predict(X_train_res)
y_pred = gbdt_clf.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_gbdt_clf)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  76.19047619047619
Confusion matrix:
 [[216   9]
 [ 61   8]]
classification report:               precision    recall  f1-score   support

           0       0.78      0.96      0.86       225
           1       0.47      0.12      0.19        69

    accuracy                           0.76       294
   macro avg       0.63      0.54      0.52       294
weighted avg       0.71      0.76      0.70       294
```

7. AdaBoostClassifier

```
# ADA model:
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier()
ada.fit(X_train_res,y_train_res)
yb_pred = ada.predict(X_test)
pred_train_ada = ada.predict(X_train_res)
y_pred = ada.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_ada)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  95.63106796116504
Testing accuracy:  69.38775510204081
Confusion matrix:
 [[183  42]
 [ 48  21]]
classification report:               precision    recall  f1-score   support

           0       0.79      0.81      0.80       225
           1       0.33      0.30      0.32        69

    accuracy                           0.69       294
   macro avg       0.56      0.56      0.56       294
weighted avg       0.68      0.69      0.69       294
```

8. **Naïve Bayes Classifier**

```
# training a Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB().fit(X_train_res, y_train_res)
gnb_predictions = gnb.predict(X_test)

# accuracy on X_test
accuracy = gnb.score(X_test, y_test)
print(accuracy)
```

```
0.7346938775510204
```

Hyperparameter tuning

For Cross Validation using K-Fold method:

What is it and how we've used it?

Cross-validation is usually used in machine learning for improving model prediction when we don't have enough data

k-fold cross-validation; we first shuffle our dataset so the order of the inputs and outputs are completely random. We do this step to make sure that our inputs are not biased in any way. Then, we split the dataset into *k* parts of equal sizes.

Hyperparameter tuning using:

1. GridSearchCV (**What is it and how we've used it?)**

GridSearchCV **is a technique for finding the optimal parameter values from a given set of parameters in a grid. It's essentially a cross-validation technique.**

2. Randomized Search CV (**What is it and how we've used it?)**

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model**.**

Hyperparameter tuning for Logistic Regression

```
param_grid = [
    {'penalty' : ['l1', 'l2'],
    'C' : np.logspace(-4, 4, 20),
    'max_iter' : [100, 1000,2500, 5000]
    }
]

from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(log_reg, param_grid = param_grid, cv = 3, verbose=True, n_jobs=-1)

best_clf = clf.fit(X_train_res,y_train_res)
```

```
Fitting 3 folds for each of 160 candidates, totalling 480 fits
```

```
best_clf.best_estimator_
```

```
LogisticRegression(C=1438.44988828766, random_state=1)
```

Hyperparameter tuning for Decision Tree

```
params = {'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}

grid_search1 = GridSearchCV(estimator=dtc,param_grid=params,cv=4, n_jobs=-1, verbose=1, scoring = "f1")

grid_search2 = GridSearchCV(estimator=dtc,
                            param_grid=params,
                            cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")

grid_search1.fit(X_train_res,y_train_res)
```

Hyperparameter tuning for Random Forest Classifier

```
rf_model_cv = RandomForestClassifier(oob_score=True, random_state=1, n_jobs=-1)

param_grid_rf = {'criterion' : ["gini", "entropy"],
                 'max_features': ['log2', 'sqrt', None],
                 'max_depth': [2, 4, 8, 16, 32, 64],
                 'min_samples_leaf' : [1, 5, 10, 20],
                 'min_samples_split' : [2, 4, 10, 14, 18],
                 'n_estimators': [100, 200, 300, 400, 500]}

gs_rf = RandomizedSearchCV(estimator=rf_model_cv,
                           param_distributions=param_grid_rf,
                           scoring='accuracy',
                           n_iter = 100,
                           cv=5,
                           n_jobs=-1)

gs_rf.fit(X_train_res,y_train_res)
```

Hyperparameter tuning for SVC

```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_train_res,y_train_res)
```

```
from sklearn.model_selection import KFold , cross_val_score
k_f = KFold(n_splits =3)
k_f

# Cross Validation score to check if model is overfitting:

cross_val_score(grid,X_train_res,y_train_res,cv=5) #
```

Hyperparameter tuning for Gradient Boosting Classifier

```
# HYPER PARAMETER TUNING:

# Tuning parameters using GridSearchCV:

params = {'max_depth':range(4,8),
          'min_samples_split':range(2,8,2),
          'learning_rate':np.arange(0.1,0.3)} # at which rate our model should learn

grd = GridSearchCV(gbdt_clf,param_grid=params)

grd.fit(X_train_res,y_train_res)
```

```
grd.best_params_
```

```
{'learning_rate': 0.1, 'max_depth': 6, 'min_samples_split': 4}
```

Hyperparameter tuning for AdaBoost Classifier

```
# HyperParameter tuning using Randomised Search CV :

from sklearn.model_selection import RandomizedSearchCV

params = {'n_estimators':[70,75,80,90,100,150],'learning_rate':[0.45,0.46,0.47,0.48,0.49,0.5]}

rnd = RandomizedSearchCV(ada,param_distributions=params)

rnd.fit(X_train_res,y_train_res)
```

```
RandomizedSearchCV(estimator=AdaBoostClassifier(),
                   param_distributions={'learning_rate': [0.45, 0.46, 0.47,
                                                          0.48, 0.49, 0.5],
                                        'n_estimators': [70, 75, 80, 90, 100,
                                                         150]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
rnd.best_estimator_
```

```
AdaBoostClassifier(learning_rate=0.46, n_estimators=150)
```

Model Evaluation and Final Result

After performing the hyper parameter tuning of all the models we can conclude that the model that is performing the best is the logistic regression model. In it we have defined

a Param grid with L1 and l2 penalty values and different c values with different Max iterations. Further will be using grade search CV that will be fitting three folds for each of 160 candidates totally 480 fits and the best estimators that we have got is the C value equal to 1438.449 with random state equal to 1.

Model Evaluation using Auc Roc curves:

What is it and how we've used it?

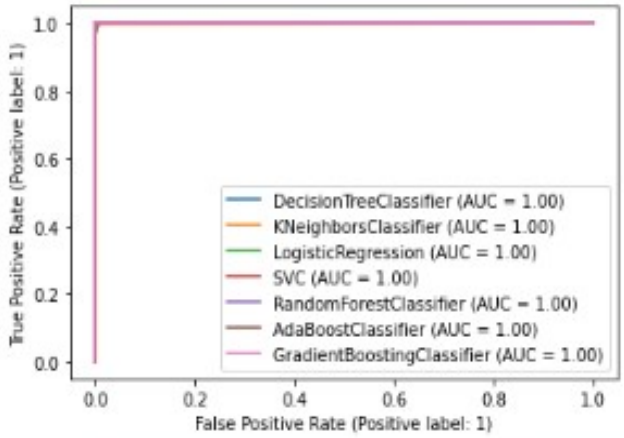
AUC-ROC is the valued metric used for evaluating the performance in classification models. The AUC-ROC metric clearly helps determine and tell us about the capability of a model in distinguishing the classes. The judging criteria being - Higher the AUC, better the model. AUC-ROC curves are frequently used to depict in a graphical way the connection and trade-off between sensitivity and specificity for every possible cut-off for a test being performed or a combination of tests being performed. The area under the ROC curve gives an idea about the benefit of using the test for the underlying question. AUC - ROC curves are also a performance measurement for the classification problems at various threshold settings.

All the models are performing well but for this particular case logistic regression model performed the best.

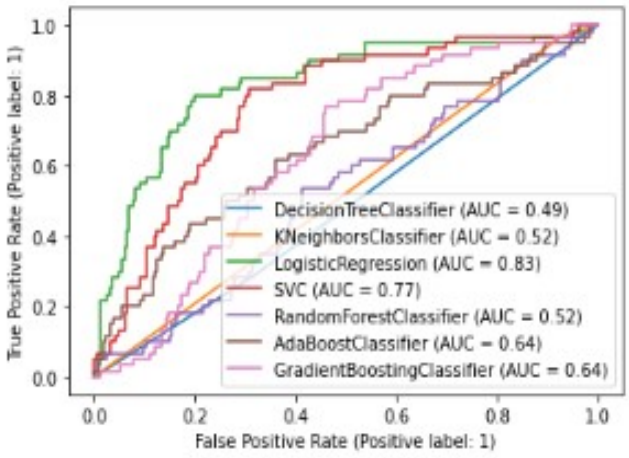
We will conclude our search for the best model by plotting ROC auc curve.

```
# Lets plot ROC AUC curve to choose best model:

from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.metrics import plot_roc_curve
```

```
# Lets check ROC AUC Curve for fitted models on training data: (True +ive Rate/False +ive Rate)

disp = plot_roc_curve(dtc,X_train_res,y_train_res)

plot_roc_curve(knn,X_train_res,y_train_res,ax=disp.ax_)

plot_roc_curve(log_reg_best,X_train_res,y_train_res,ax=disp.ax_)

plot_roc_curve(svc,X_train_res,y_train_res,ax=disp.ax_)

plot_roc_curve(rfc_f,X_train_res,y_train_res,ax=disp.ax_)

plot_roc_curve(ada_boosted,X_train_res,y_train_res,ax=disp.ax_)

plot_roc_curve(gbdt_clf_f,X_train_res,y_train_res,ax=disp.ax_)

plt.legend(prop={'size':10},loc='lower right')

plt.show()

# This result is on training data.
```



```
# Lets check ROC AUC Curve for fitted models on testing data: (True +ive Rate/False +ive Rate)

disp = plot_roc_curve(dtc,X_test,y_test)

plot_roc_curve(knn,X_test,y_test,ax=disp.ax_)

plot_roc_curve(log_reg_best,X_test,y_test,ax=disp.ax_)

plot_roc_curve(svc,X_test,y_test,ax=disp.ax_)

plot_roc_curve(rfc_f,X_test,y_test,ax=disp.ax_)

plot_roc_curve(ada_boosted,X_test,y_test,ax=disp.ax_)

plot_roc_curve(gbdt_clf_f,X_test,y_test,ax=disp.ax_)

plt.legend(prop={'size':10},loc='lower right')

plt.show()

# This result is on training data.
```



FOR THE TRAINING DATA SET VALUES WE CAN SEE THAT ALL THE MODELS GAVE AUC VALUE THAT IS ONE.

BUT WHEN WE ARE PLOTTING THE AUCROC CURVE ON THE TEST DATA SET VALUES OF AUC ARE THE BEST FOR THE LOGISTIC REGRESSION MODEL. THE AUC FOR LOGISTIC REGRESSION IS 0.83 RESPECTIVELY.

Finding Best random state for our model:

```
# Finding best random state for logistic regression model:

maxAccu = 0
maxRS = 0

for i in range(1,200):
    log_reg = LogisticRegression()
    log_reg.fit(X_train_res,y_train_res)
    y_pred = log_reg.predict(X_test)
    acc = accuracy_score(y_test,y_pred)
    print('Testing accuracy: ', acc, 'at random state', i)

    if acc > maxAccu:
        maxAccu = acc
        maxRS = i
        print('Max accuracy',maxAccu,'max random state',maxRS)
```

```
Testing accuracy:  0.8156996587030717 at random state 1
Max accuracy 0.8156996587030717 max random state 1
Testing accuracy:  0.8156996587030717 at random state 2
Testing accuracy:  0.8156996587030717 at random state 3
Testing accuracy:  0.8156996587030717 at random state 4
Testing accuracy:  0.8156996587030717 at random state 5
Testing accuracy:  0.8156996587030717 at random state 6
Testing accuracy:  0.8156996587030717 at random state 7
Testing accuracy:  0.8156996587030717 at random state 8
Testing accuracy:  0.8156996587030717 at random state 9
Testing accuracy:  0.8156996587030717 at random state 10
Testing accuracy:  0.8156996587030717 at random state 11
Testing accuracy:  0.8156996587030717 at random state 12
Testing accuracy:  0.8156996587030717 at random state 13
```

```
# Logistic Regression:

log_reg_best = LogisticRegression(C=1438.44988828766, random_state=1)
log_reg_best.fit(X_train_res,y_train_res)
pred_train = log_reg_best.predict(X_train_res)
y_pred = log_reg_best.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  82.25255972696246
Confusion matrix:
 [[230   3]
 [ 49  11]]
classification report:               precision    recall  f1-score   support

           0       0.82      0.99      0.90       233
           1       0.79      0.18      0.30        60

    accuracy                           0.82       293
   macro avg       0.81      0.59      0.60       293
weighted avg       0.82      0.82      0.78       293
```

Using 2nd approach we have utilized the poer of Random Forest model and the result is shared below:

```
: # Random forrest classifier model:

rfc_f = RandomForestClassifier()
rfc_f.fit(X_train_res,y_train_res)
pred_train_rfc_f = rfc_f.predict(X_train_res)
y_pred = rfc_f.predict(X_test)
acc = accuracy_score(y_test,y_pred)
print('Training accuracy: ', accuracy_score(y_train_res,pred_train_rfc_f)*100)
print('Testing accuracy: ', acc*100)
confusion_mat = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',confusion_mat)
clr = classification_report(y_test,y_pred)
print('classification report: ' ,clr)
```

```
Training accuracy:  100.0
Testing accuracy:  80.0
Confusion matrix:
 [[138  18]
 [ 21  18]]
classification report:               precision    recall  f1-score   support

           0       0.87      0.88      0.88       156
           1       0.50      0.46      0.48        39

    accuracy                           0.80       195
   macro avg       0.68      0.67      0.68       195
weighted avg       0.79      0.80      0.80       195
```

Understanding Final Result parameters:

Confusion matrix

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa – both variants are found in the literature. The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e. commonly mislabeling one as another).

Classification Report

What does a classification report show?

A Classification report is used to **measure the quality of predictions from a classification algorithm**. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

Various elements of classification report:

Precision — *What percent of your predictions were correct?*
Recall — *What percent of the positive cases did you catch?*
F1-score — *The weighted average of Precision and Recall*
support —*The number of occurrences of each given class*

Let's understand them in depth:

The report shows the main classification metrics precision, recall and f1-score on a per-class basis. The metrics are calculated by using true and false positives, true and false negatives. Positive and negative in this case are generic names for the predicted classes. There are four ways to check if the predictions are right or wrong:

1. **TN / True Negative:** when a case was negative and predicted negative
2. **TP / True Positive:** when a case was positive and predicted positive
3. **FN / False Negative:** when a case was positive but predicted negative
4. **FP / False Positive:** when a case was negative but predicted positive

Precision – What percent of your predictions were correct?

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

TP – True Positives
FP – False Positives

Precision – Accuracy of positive predictions.
Precision = TP/(TP + FP)

from sklearn.metrics import precision_score print("Precision score: {}".format(precision_score(y_true,y_pred)))

Recall – What percent of the positive cases did you catch?

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

FN – False Negatives

Recall: Fraction of positives that were correctly identified.
Recall = TP/(TP+FN)

from sklearn.metrics import recall_score print("Recall score: {}".format(recall_score(y_true,y_pred)))

F1 score – What percent of positive predictions were correct?

The F_1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F_1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F_1 should be used to compare classifier models, not global accuracy. **F1 Score = 2*(Recall * Precision) / (Recall + Precision)**

Accuracy score of our logististic regression model on the training dataset is coming out to be 100 percent and accuracy score of our model on test dataset is 82.25 %.

With a precision of 90 % our model was able to predict whether an insurance claim was legit

Pipelining

What is it and how we've used it?

A Machine Learning pipeline is a process of automating the workflow of a complete machine learning task**.** It can be done by enabling a sequence of data to be transformed and correlated together in a model that can be analyzed to get the output. A typical pipeline includes raw data input, features, outputs, model parameters, ML models, and Predictions. Moreover, an ML Pipeline contains multiple sequential steps that perform everything ranging from data extraction and pre-processing to model training and deployment in Machine learning in a modular approach. It means that in the pipeline, each step is designed as an independent module, and all these modules are tied together to get the final result.
Our model pipeline is mentioned below:

```
# Creating pipeline:

from sklearn.pipeline import Pipeline

pipe1 = Pipeline([('pt',PowerTransformer()),
                  ('pca',PCA(n_components=975)),('base_model1',LogisticRegression(C=1438.44988828766, random_state=1))])

pipe1.fit(X_train_res,y_train_res)

y_pred = pipe1.predict(X_test)

print('Training accuracy: ', accuracy_score(y_train_res,pred_train)*100)
print('Testing accuracy: ', acc*100)

confusion_mat = confusion_matrix(y_test,y_pred)

print('Confusion matrix: \n',confusion_mat)

clr = classification_report(y_test,y_pred)

print('classification report: ' ,clr)
```

Save the prediction file.

```
# Saving regression model to pickle string

import pickle
saved_model1 = pickle.dumps(pipe1)
pipe_pickle1 = pickle.loads(saved_model1)
pipe_pickle1.predict(X_test) # predicting testing data
```

```
Training accuracy:  100.0
Testing accuracy:  82.25255972696246
Confusion matrix:
 [[233   0]
 [ 60   0]]
classification report:                precision    recall  f1-score   support

           0       0.80      1.00      0.89       233
           1       0.00      0.00      0.00        60

    accuracy                           0.80       293
   macro avg       0.40      0.50      0.44       293
weighted avg       0.63      0.80      0.70       293

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0])
```

Remarks

In this project we build have built a classification model that detects and insurance fraud. This model can be used to detect the frauds and does it can prevent and reduce losses in insurance companies.

The major challenge in this data set and fraud detection is that because of imbalanced data set the values of class level category representing fraud detected were very less as compared to no fraud detected values.

We developed 8 class fire models mainly logistic regression ke nearest neighbour random forest decision tree nav base classifier gradient boosting class fire support vector classifier adda boost classifier respectively. All these models were tested after performing the hyper parameter tuning and cross validation using ke fold cross validation method.

Other notable steps include over sampling with SMOTE, hyper parameter tuning and plotting rocauc curve of the models.

The best and final fitted model was a logistic regression model with hyperparameter tuning done that give ROC auc values of 83. The model perform well as compared to other models. In conclusion the model was able to correctly distinguish between fraud claims and legit claims with average accuracy.

The analysis has limitations as well. Due to lack of sample size and imbalance nature of data set can we eliminated that could lead to even better classification and prediction.

Furthur more intense hyperparameter tuning and other approaches could have been utilized.

References

Following references have been fruitful in understanding the problem statement and framing of our research and development of model to predict Fraud cases.

- **www.google.com**
- **https://www.wikipedia.org/**
- **https://www.geeksforgeeks.org/**
- **https://www.kdnuggets.com**