Name : Naman Thaker

Roll no : 20BCE529

Subject : Data Mining

Division D

```
import numpy as np
import math
import pandas as pd
dataSet=pd.read_csv('./Automobile_insurance_fraud.csv')
dataSet
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_
0	328	48	521585	17-10-2014	OH	250/
1	228	42	342868	27-06-2006	IN	250/
2	134	29	687698	06-09-2000	OH	100/
3	256	41	227811	25-05-1990	IL	250/
4	228	44	367455	06-06-2014	IL	500/1
...	...	...	...	...	...	
995	3	38	941851	16-07-1991	OH	500/1
996	285	41	186934	05-01-2014	IL	100/
997	130	34	918516	17-02-2003	OH	250/
998	458	62	533940	18-11-2011	IL	500/1
999	456	60	556080	11-11-1996	OH	250/

1000 rows × 40 columns

```
dataSet['incident_severity'].replace({'Minor Damage': 0, 'Major Damage': 1})
```

```
0      1
1      0
2      0
3      1
4      0
```

```
      ..
995    0
996    1
997    0
998    1
999    0
Name: incident_severity, Length: 1000, dtype: object
```

```
def Display(mat, typ):
    for i in range(len(mat)):
        for j in range(len(mat)):
            if i>=j:
                if typ=='f':
                    print('{:.2f}'.format(mat[i][j]), end=' ')
                else:
                    print(int(mat[i][j]), end=' ')
        print()
    print()
    print()
    print()


def Dissimilarity(data):
    data=np.array(data)
    matrix=np.zeros((data.shape[0], data.shape[0]))
    # print('OK')
    p=data.shape[1]
    for i in range(len(data)):
        for j in range(len(data)):
            m=0
            for k in range(len(data[0])):
                if data[i][k]==data[j][k]:
                    m+=1
            matrix[i][j]=(p-m)/p
            matrix[j][i]=matrix[i][j]

    return matrix


mat = Dissimilarity(dataSet)


# ans=input('Do you want to get dissimilarity between any 2 particular data records?[Y/N]:
# if ans=='Y':
#     one=int(input('Enter 1st data entry: '))
#     two=int(input('Enter 2nd data entry: '))
#     print('{:.2f}'.format(mat[one-1][two-1]))


def BinaryDissimilarity(data, attributeName):
    data=data[attributeName]
    matrix=np.zeros((data.shape[0], data.shape[0]))
    for i in range(len(data)):
        for j in range(len(data)):
            if i>=j:
                if data[i]==data[j]:
```

```
                    matrix[i][j]=0
                else:
                    matrix[i][j]=1
                matrix[j][i]=matrix[i][j]
    return matrix
```

```
mat_Indian=BinaryDissimilarity(dataSet, 'incident_severity')
# Display(mat_Indian, 'i')
```

```
def NominalDissimilarity(data, attributeName):
    data=data[attributeName]
    matrix=np.zeros((data.shape[0], data.shape[0]))
    for i in range(len(data)):
        for j in range(i, len(data)):
            if data[i]==data[j]:
                matrix[i][j]=0
            else:
                matrix[i][j]=1
            matrix[j][i]=matrix[i][j]
    return matrix
```

```
mat_Team=NominalDissimilarity(dataSet, 'policy_state')
# Display(mat, 'i')
```

```
def NumericDissimilarity(data, attributeName):
    data=data[attributeName]
    matrix=np.zeros((data.shape[0], data.shape[0]))
    for i in range(data.shape[0]):
        for j in range(data.shape[0]):
            if i>=j:
                matrix[i][j] = abs(data[i]-data[j])/(max(data)-min(data))
                matrix[j][i] = matrix[i][j]

    return matrix
```

```
mat_FinalPrice=NumericDissimilarity(dataSet, 'total_claim_amount')
# Display(mat, 'f')
```

```
dissimilarity_matrix = [mat_Indian,mat_Team, mat_FinalPrice]
# dissimilarity_matrix[1][6][4]
np.shape(dissimilarity_matrix)
```

```
(3, 1000, 1000)
```

```
def TotalDissimilarity(data, dissimilarity_matrix):
    data=np.array(data)
    matrix=np.zeros((data.shape[0], data.shape[0]))
    for i in range(3):
        for j in range(1000):
```

```
            n=0
            d=0
            for k in range(100):
                dell=1
                n+=dell*dissimilarity_matrix[i][j][k]
                d+=dell
            matrix[i][j]=n/d


    return matrix

mat=TotalDissimilarity(dataSet, dissimilarity_matrix)
Display(mat, 'f')




# Python program to compute distance matrix

# import important libraries
import numpy as np
from scipy.spatial import distance_matrix
a = np.array([dataSet['total_claim_amount'], dataSet['injury_claim'],dataSet['property_cla
# Create the matrices
# x = np.array([[1,2],[2,1],[2,2]])
# y = np.array([[5,0],[1,2],[2,0]])
a.shape
a
b = np.array([dataSet['policy_annual_premium'], dataSet['policy_deductable'],dataSet['capi
b.shape
b
#Display the matrices
print("matrix x:\n", a)
print("matrix y:\n", b)

 # compute the distance matrix
dist_mat = distance_matrix(a, b, p=2)

# # display distance matrix
print("Distance Matrix:\n", dist_mat)
```

```
matrix x:
 [[71610  5070 34650 ... 67500 46980  5060]
 [ 6510   780  7700 ...  7500  5220   460]
 [13020   780  3850 ...  7500  5220   920]
 [52080  3510 23100 ... 52500 36540  3680]]
matrix y:
 [[ 1406.91  1197.22  1413.14 ...  1383.49  1356.92   766.19]
 [ 1000.    2000.    2000.   ...   500.    2000.    1000.  ]
 [53300.       0.   35100.   ... 35100.       0.       0.  ]
 [    0.       0.       0.   ...     0.       0.       0.  ]]
Distance Matrix:
 [[1830056.3824205  1833362.60363301 1487504.63138775 2804987.25957891]
 [ 249112.24712005  252186.479019   1051600.51930379 1413141.85282299]
```

```
[ 247143.99635962  249957.7182245  1055355.80488288 1409850.68177449]
```

Performing regression

Logistic Regression Kind of technique need to resolve this problem

EDA Process

```
data.describe()
```

	months_as_customer	age	policy_number	policy_deductable	policy_annu
count	1000.000000	1000.000000	1000.000000	1000.000000	
mean	203.954000	38.948000	546238.648000	1136.000000	
std	115.113174	9.140287	257063.005276	611.864673	
min	0.000000	19.000000	100804.000000	500.000000	
25%	115.750000	32.000000	335980.250000	500.000000	
50%	199.500000	38.000000	533135.000000	1000.000000	
75%	276.250000	44.000000	759099.750000	2000.000000	
max	479.000000	64.000000	999435.000000	2000.000000	

Checking null value in dataset

```
data.isnull().sum()
```

```
months_as_customer             0
age                            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_sex                    0
insured_education_level        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
incident_severity               0
authorities_contacted           0
incident_state                  0
incident_city                   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
_c39                         1000
dtype: int64
```

Oservation: 1) we have 1000 number of data and _c39 contains all if the data as null so need to drop that column

Dropping _c39

```
data.drop('_c39',
  axis='columns', inplace=True)
```

_c39 column has been dropped

```
import seaborn as sns
alpha = sns.countplot(x="police_report_available",data=data)
print(data["police_report_available"].value_counts())
```

```
?      343
NO     343
YES    314
Name: police_report_available, dtype: int64
```

Observation: we can change ? with nan and then with mean or median. But if will encode it it will be considered as 0

```
data['police_report_available'] = data['police_report_available'].replace(['?'],np.nan)
```

```
data['police_report_available']
```

```
0      YES
1      NaN
2       NO
3       NO
4       NO
      ...
995    NaN
996    NaN
997    YES
998    YES
999    NaN
Name: police_report_available, Length: 1000, dtype: object
```

```
data.isnull().sum()
```

```
months_as_customer             0
age                            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_sex                    0
insured_education_level        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
incident_severity              0
authorities_contacted          0
incident_state                 0
incident_city                  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        343
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
dtype: int64
```

Handling null values of police_report_available

```
#1. Function to replace NAN values with mode value this both rows are categorical,
#not numeric based with datatype of float or int
def impute_nan_most_frequent_category(data,ColName):
    # .mode()[0] - gives first category name
     most_frequent_category=data[ColName].mode()[0]

    # replace nan values with most occured category
     #data[ColName + "_Imputed"] = data[ColName]
     #data[ColName + "_Imputed"].fillna(most_frequent_category,inplace=True)
     data[ColName] = data[ColName]
     data[ColName].fillna(most_frequent_category,inplace=True)


#2. Call function to impute most occured category
for Columns in ['police_report_available']:
    impute_nan_most_frequent_category(data,Columns)

# Display imputed result
data[['police_report_available']].head(10)
```

police_report_available

```
#Rechecking null values in dataset
data.isnull().sum()
```

```
months_as_customer             0
age                            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_sex                    0
insured_education_level        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
incident_severity              0
authorities_contacted          0
incident_state                 0
incident_city                  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
dtype: int64
```

Observation _c39 is dropped and ? is replaced with NaN and handling of NAN

Data Cleaning

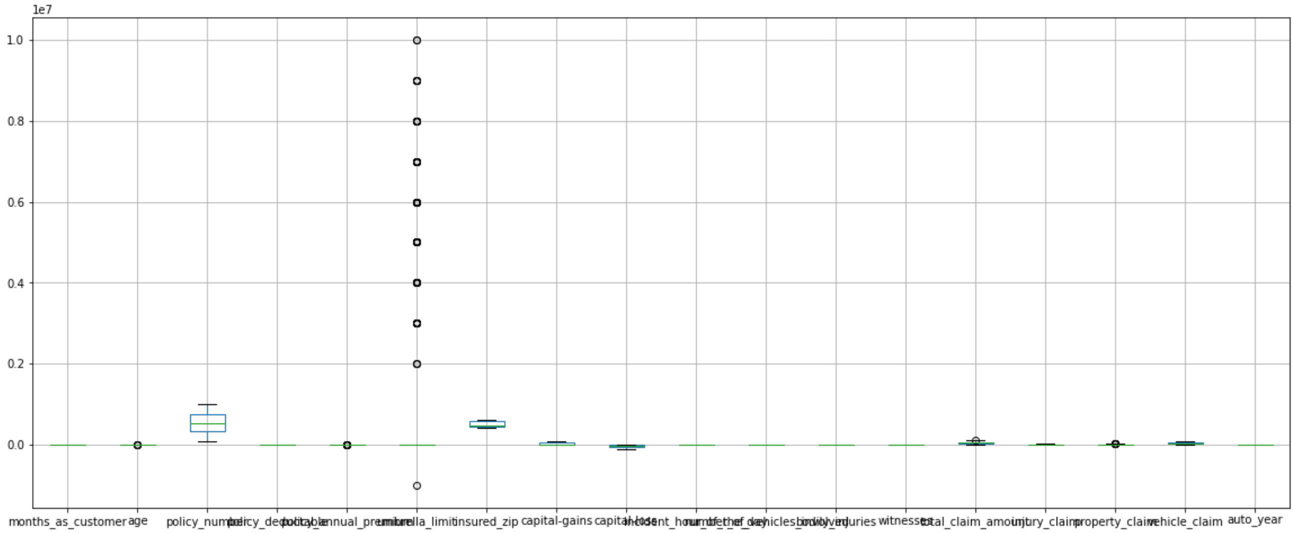
```
data.skew()
```

```
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
```

There contains a skewness

```
#checking for outliers
data.iloc[:,:].boxplot(figsize=[20,8])
plt.show()
```



There also contains an outlier in ubrella_limit

Encoding

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
```

```
for i in data.columns:
    if data[i].dtypes == "object":
        data[i]=enc.fit_transform(data[i].values.reshape(-1,1))
```

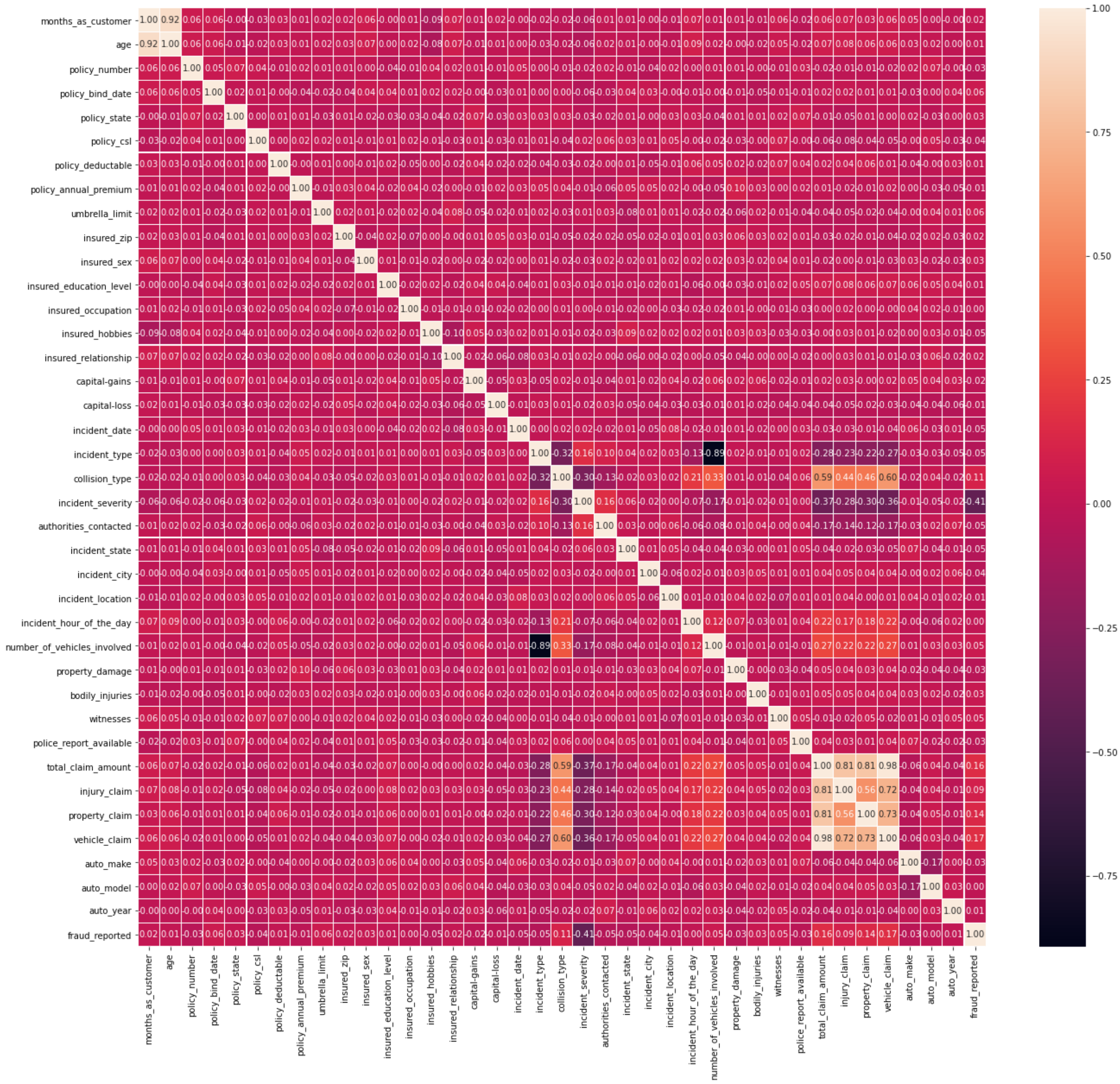
```
data
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_
0	328	48	521585	532.0	2.0	
1	228	42	342868	821.0	1.0	
2	134	29	687698	186.0	2.0	
3	256	41	227811	766.0	0.0	
4	228	44	367455	181.0	0.0	
...	...	...	...	...	...	
995	3	38	941851	487.0	2.0	
996	285	41	186934	129.0	0.0	
997	130	34	918516	509.0	2.0	
998	458	62	533940	573.0	0.0	
999	456	60	556080	359.0	2.0	

1000 rows × 39 columns

Corelation of feature variable with the target variable

```
corr_matrix_hmap=data.corr()
plt.figure(figsize=(22,20))
sns.heatmap(corr_matrix_hmap,annot=True,linewidths=0.1,fmt="0.2f")
plt.show()
```


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
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

```
corr_matrix_hmap["fraud_reported"].sort_values(ascending=False)
```

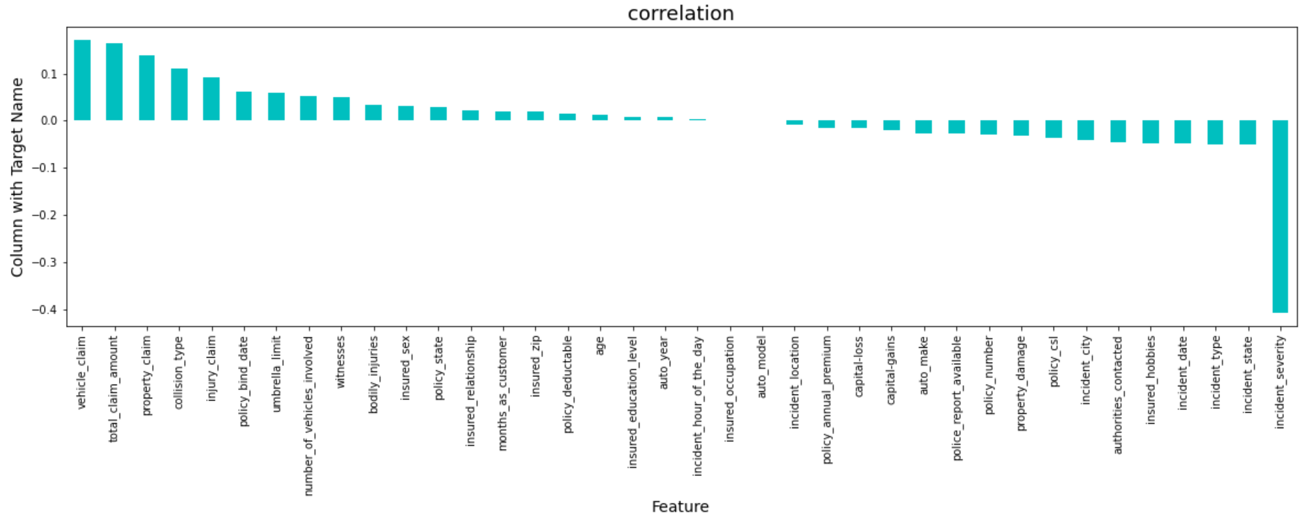
```
fraud_reported                 1.000000
vehicle_claim                  0.170049
total_claim_amount             0.163651
property_claim                 0.137835
collision_type                 0.110130
injury_claim                   0.090975
policy_bind_date               0.060642
umbrella_limit                 0.058622
number_of_vehicles_involved    0.051839
witnesses                      0.049497
bodily_injuries                0.033877
insured_sex                    0.030873
policy_state                   0.029432
insured_relationship           0.021043
months_as_customer             0.020544
insured_zip                    0.019368
policy_deductable              0.014817
age                            0.012143
insured_education_level        0.008808
auto_year                      0.007928
incident_hour_of_the_day       0.004316
insured_occupation             0.001564
auto_model                     0.000720
incident_location             -0.008832
policy_annual_premium         -0.014480
capital-loss                  -0.014863
capital-gains                 -0.019173
auto_make                     -0.027519
police_report_available       -0.027768
policy_number                 -0.029443
property_damage               -0.030497
policy_csl                    -0.037190
incident_city                 -0.040403
authorities_contacted         -0.045802
insured_hobbies               -0.046838
incident_date                 -0.047726
incident_type                 -0.050376
incident_state                -0.051407
incident_severity             -0.405988
Name: fraud_reported, dtype: float64
```

Most highly corelated variavle is : vehicle_claim

Least is: incident_severity

```
plt.figure(figsize=(20,5))
```

```
data.corr()['fraud_reported'].sort_values(ascending=False).drop(['fraud_reported']).plot(k
plt.xlabel('Feature',fontsize=14)
plt.ylabel('Column with Target Name',fontsize=14)
plt.title('correlation',fontsize=18)
plt.show()
```



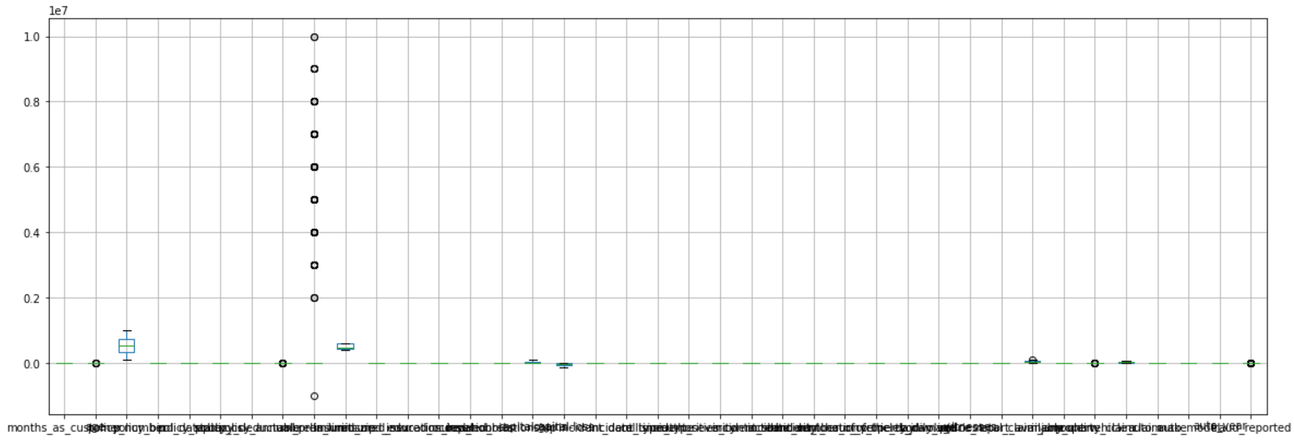
Maximun corelated: vehcile_claim

Minimum corelated:insured_occupation and auto_model and auto_year also somewhat

Negativel corelated: incident_severity

Outliers

```
#checking for outliers
data.iloc[:,:].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



```
# Removing Outliers
from scipy.stats import zscore
z= np.abs(zscore(data))
z
```

```
array([[1.07813958, 0.99083647, 0.09595307, ..., 1.64574255, 0.1834404 ,
        1.74601919],
       [0.2089946 , 0.33407345, 0.79152739, ..., 0.65747047, 0.31549088,
        1.74601919],
       [0.60800168, 1.08891308, 0.55056594, ..., 0.95970204, 0.31549088,
        0.57273139],
       ...,
       [0.64276748, 0.54161057, 1.44891961, ..., 0.02857005, 1.5139238 ,
        0.57273139],
       [2.20802805, 2.52328351, 0.04786687, ..., 1.28637088, 1.18130295,
        0.57273139],
       [2.19064515, 2.3043625 , 0.03830297, ..., 0.65747047, 0.31549088,
        0.57273139]])
```

```
threshold = 3
print(np.where(z<3))
```

```
(array([  0,   0,   0, ..., 999, 999, 999], dtype=int64), array([ 0,  1,  2, ..., 36,
```

```
#removing outliers
data_new = data[(z<3).all(axis=1)]
```

```
data.shape
```

```
(1000, 39)
```

```
#After removing outliers
data_new.shape
```

```
(980, 39)
```

```
data=data_new
```

```
data
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_
0	328	48	521585	532.0	2.0	
1	228	42	342868	821.0	1.0	
2	134	29	687698	186.0	2.0	
3	256	41	227811	766.0	0.0	
4	228	44	367455	181.0	0.0	
...	...	...	...	...	...	
995	3	38	941851	487.0	2.0	
996	285	41	186934	129.0	0.0	
997	130	34	918516	509.0	2.0	
998	458	62	533940	573.0	0.0	
999	456	60	556080	359.0	2.0	

980 rows × 39 columns

Outliers are been handled

Seperating Independent Variables and Target Variables

```
# x= independent variable
x = data.iloc[:,0:-1]
x.head()
```

months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_cs

```
#y = target variable = fraud_reported
y = data.iloc[:,-1]
y.head()
```

```
0    1.0
1    1.0
2    0.0
3    1.0
4    0.0
Name: fraud_reported, dtype: float64
```

```
x.shape
```

```
(980, 38)
```

```
y.shape
```

```
(980,)
```

Rechecking skewness after removing outliers

```
x.skew()
```

```
months_as_customer             0.362608
age                            0.475385
policy_number                  0.036283
policy_bind_date               0.006386
policy_state                  -0.038157
policy_csl                     0.098248
policy_deductable              0.476090
policy_annual_premium          0.035964
umbrella_limit                 1.801424
insured_zip                    0.837283
insured_sex                    0.139324
insured_education_level        0.006286
insured_occupation            -0.055360
insured_hobbies               -0.061488
insured_relationship           0.078339
capital-gains                  0.466619
capital-loss                  -0.376884
incident_date                  0.002604
incident_type                  0.090563
collision_type                -0.194015
incident_severity              0.277726
authorities_contacted         -0.114044
incident_state                -0.149255
incident_city                  0.043882
incident_location             -0.003369
incident_hour_of_the_day      -0.039280
number_of_vehicles_involved    0.509725
property_damage                0.101196
bodily_injuries                0.003757
```

```
witnesses                    0.026211
police_report_available      0.796221
total_claim_amount          -0.593593
injury_claim                 0.271759
property_claim               0.361356
vehicle_claim               -0.620936
auto_make                   -0.028739
auto_model                  -0.073462
auto_year                   -0.054522
dtype: float64
```

```
x.dtypes
```

```
months_as_customer             int64
age                            int64
policy_number                  int64
policy_bind_date             float64
policy_state                 float64
policy_csl                   float64
policy_deductable              int64
policy_annual_premium        float64
umbrella_limit                 int64
insured_zip                    int64
insured_sex                  float64
insured_education_level      float64
insured_occupation           float64
insured_hobbies              float64
insured_relationship         float64
capital-gains                  int64
capital-loss                   int64
incident_date                float64
incident_type                float64
collision_type               float64
incident_severity            float64
authorities_contacted        float64
incident_state               float64
incident_city                float64
incident_location            float64
incident_hour_of_the_day       int64
number_of_vehicles_involved    int64
property_damage              float64
bodily_injuries                int64
witnesses                      int64
police_report_available      float64
total_claim_amount             int64
injury_claim                   int64
property_claim                 int64
vehicle_claim                  int64
auto_make                    float64
auto_model                   float64
auto_year                      int64
dtype: object
```

Handling Skewness

```
#Method for removing skew
```

```
from sklearn.preprocessing import power_transform
z = power_transform(x[0:])
data_new= pd.DataFrame(z,columns=x.columns)
```

```
x = data_new
```

```
#after removing skewness
x.skew()
```

```
months_as_customer            -0.133972
age                           -0.002183
policy_number                 -0.161791
policy_bind_date              -0.293677
policy_state                  -0.150765
policy_csl                    -0.096814
policy_deductable              0.022179
policy_annual_premium         -0.007258
umbrella_limit                -7.932397
insured_zip                    0.000000
insured_sex                    0.139324
insured_education_level       -0.187642
insured_occupation            -0.238129
insured_hobbies               -0.248575
insured_relationship          -0.160168
capital-gains                  0.031294
capital-loss                   0.088750
incident_date                 -0.264010
incident_type                 -0.095572
collision_type                -0.204055
incident_severity             -0.079569
authorities_contacted         -0.223816
incident_state                -0.256064
incident_city                 -0.181833
incident_location             -0.288690
incident_hour_of_the_day      -0.258027
number_of_vehicles_involved    0.372833
property_damage               -0.093063
bodily_injuries               -0.133824
witnesses                     -0.151669
police_report_available        0.796221
total_claim_amount            -0.508540
injury_claim                  -0.416732
property_claim                -0.357397
vehicle_claim                 -0.521805
auto_make                     -0.229846
auto_model                    -0.276099
auto_year                     -0.013973
dtype: float64
```

```
# #one can drop umbrella_limit
# x.drop('umbrella_limit',
#   axis='columns')
```

```
#one can drop umbrella_limit
```

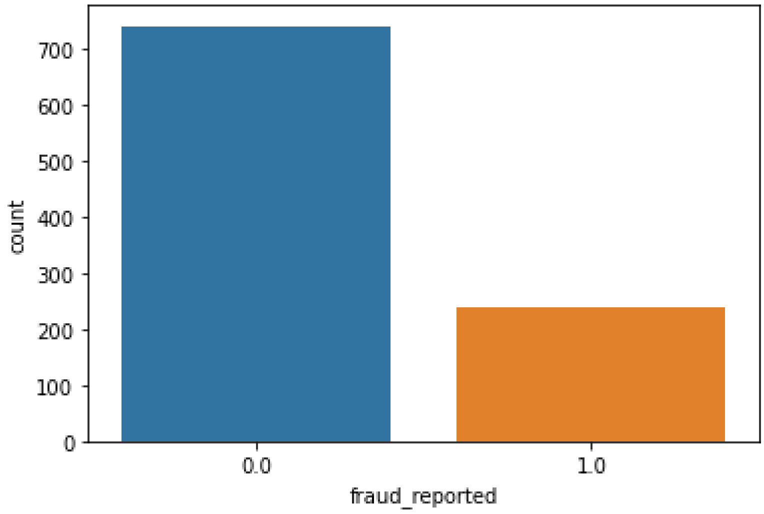
Visualizations

```
data.columns
```

```
Index(['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'],
      dtype='object')
```

```
import seaborn as sns
alpha = sns.countplot(x="fraud_reported",data=data)
print(data["fraud_reported"].value_counts())
```

```
0.0    740
1.0    240
Name: fraud_reported, dtype: int64
```



fraud_reported which is out target variable contains 2 values, 0 and 1

```
x.columns
```

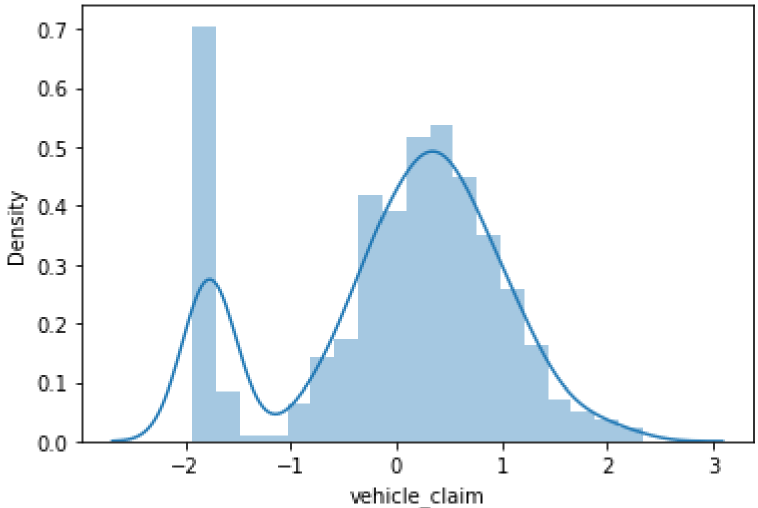
```
Index(['months_as_customer', 'age', 'policy_number', 'policy_bind_date',
       'policy_state', 'policy_csl', 'policy_deductable',
       'policy_annual_premium', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', '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'],
          dtype='object')
```

```
df_visual= x[['months_as_customer', 'age', 'policy_number', 'policy_bind_date',
       'policy_state', 'policy_csl', 'policy_deductable',
       'policy_annual_premium', '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']].copy()
```

```
import seaborn as sns
sns.distplot(df_visual['vehicle_claim'],kde=True)
```

```
<AxesSubplot:xlabel='vehicle_claim', ylabel='Density'>
```



it contains overfitting

```
# import seaborn as sns
# sns.distplot(df_visual['umbrella_limit'],kde=True)
```

It don't contains overfitting

Scaling

```
x.describe()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_st
count	9.800000e+02	9.800000e+02	9.800000e+02	9.800000e+02	9.800000e-
mean	-8.791154e-17	-2.356392e-16	-3.743746e-16	1.340198e-16	-1.518060e
std	1.000511e+00	1.000511e+00	1.000511e+00	1.000511e+00	1.000511e-
min	-2.411559e+00	-2.858572e+00	-1.934451e+00	-2.169553e+00	-1.265764e-
25%	-6.921974e-01	-7.252830e-01	-7.735994e-01	-7.993862e-01	-1.265764e-
50%	7.107941e-02	3.204583e-05	2.358119e-02	9.725113e-02	5.900219e
75%	6.834392e-01	6.510520e-01	8.405397e-01	8.631367e-01	1.141477e-
max	2.034478e+00	2.261323e+00	1.633618e+00	1.545052e+00	1.141477e-

8 rows × 37 columns

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

```
x=mms.fit_transform(x)
x
```

```
array([[0.77734391, 0.75409801, 0.53557702, ..., 0.80977259, 0.04489444,
        0.43657897],
       [0.60966749, 0.64175994, 0.33363663, ..., 0.67536561, 0.39386981,
        0.58692128],
       [0.42638858, 0.33644422, 0.70661621, ..., 0.37906556, 0.82810234,
        0.58692128],
       ...,
       [0.41773978, 0.46641077, 0.92660097, ..., 0.87453491, 0.57327901,
        0.04745782],
       [0.97067268, 0.97260724, 0.5487646 , ..., 0.11101935, 0.18895892,
        0.14315387],
       [0.9678556 , 0.9443819 , 0.57219064, ..., 0.67536561, 0.39386981,
        0.58692128]])
```

Model Training

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.33,random_state = 42)
```

```
x_train.shape
```

```
(656, 37)
```

```
y_train.shape
```

```
(656,)
```

```
x_test.shape
```

```
(324, 37)
```

```
y_test.shape
```

```
(324,)
```

```
y_train
```

```
177    0.0
130    0.0
646    0.0
407    0.0
902    0.0
      ...
109    1.0
278    1.0
878    1.0
445    1.0
105    0.0
Name: fraud_reported, Length: 656, dtype: float64
```

```
from sklearn.linear_model import LogisticRegression
lm = LogisticRegression()
```

```
lm.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
lm.score(x_train,y_train)
```

```
0.8109756097560976
```

Prediction

```
#predict the values
pred=lm.predict(x_test)
```

```
print("Predicted Allitation",pred)
print("Actual Allitation",y_test)
```

```
Predicted Allitation [0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0.
 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
Actual Allitation 804    0.0
450    0.0
144    0.0
710    1.0
68     0.0
      ...
589    0.0
950    0.0
535    1.0
925    0.0
361    1.0
Name: fraud_reported, Length: 324, dtype: float64
```

```
print('Accuracy Score:',accuracy_score(y_test,pred))
```

```
Accuracy Score: 0.75
```

Finding Best Random State

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.33,random_state = i)
    LR = LogisticRegression()
    LR.fit(x_train,y_train)
    predrf = LR.predict(x_test)
    acc =accuracy_score(y_test,predrf)
    if acc > maxAccu:
        maxAccu = acc
        maxRS = i

print("Best score is: ",maxAccu,"on Random_state",maxRS)
```

```
Best score is:  0.8209876543209876 on Random_state 58
```

Tain-Test Model as per Best Ransom state

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.33,random_state = 58)
LR = LogisticRegression()
LR.fit(x_train,y_train)
predrf = LR.predict(x_test)
```

```
from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report
print('Accuracy Score:', accuracy_score(y_test,predrf))
print('Confusion Matrix:', confusion_matrix(y_test,predrf))
print('Classification Report:', classification_report(y_test,predrf))
```

```
Accuracy Score: 0.8209876543209876
Confusion Matrix: [[232  17]
 [ 41  34]]
Classification Report:               precision    recall  f1-score   support

         0.0       0.85      0.93      0.89       249
         1.0       0.67      0.45      0.54        75

    accuracy                           0.82       324
   macro avg       0.76      0.69      0.71       324
weighted avg       0.81      0.82      0.81       324
```

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
preddt = dt.predict(x_test)
print('Accuracy Score:', accuracy_score(y_test,preddt))
print('Confusion Matrix:', confusion_matrix(y_test,preddt))
print('Classification Report:', classification_report(y_test,preddt))
```

```
Accuracy Score: 0.7716049382716049
Confusion Matrix: [[207  42]
 [ 32  43]]
Classification Report:               precision    recall  f1-score   support

         0.0       0.87      0.83      0.85       249
         1.0       0.51      0.57      0.54        75

    accuracy                           0.77       324
   macro avg       0.69      0.70      0.69       324
weighted avg       0.78      0.77      0.78       324
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

```
rfc.fit(x_train,y_train)
predrfc = rfc.predict(x_test)
print('Accuracy Score:', accuracy_score(y_test,predrfc))
print('Confusion Matrix:', confusion_matrix(y_test,predrfc))
print('Classification Report:', classification_report(y_test,predrfc))
```

```
Accuracy Score: 0.7932098765432098
Confusion Matrix: [[236  13]
 [ 54  21]]
Classification Report:               precision    recall  f1-score   support

         0.0       0.81      0.95      0.88       249
         1.0       0.62      0.28      0.39        75

    accuracy                           0.79       324
   macro avg       0.72      0.61      0.63       324
weighted avg       0.77      0.79      0.76       324
```

```
from sklearn import svm
svm = svm.SVC()
svm.fit(x_train,y_train)
predsvm = svm.predict(x_test)
print('Accuracy Score:', accuracy_score(y_test,predsvm))
print('Confusion Matrix:', confusion_matrix(y_test,predsvm))
print('Classification Report:', classification_report(y_test,predsvm))
```

```
Accuracy Score: 0.7746913580246914
Confusion Matrix: [[246   3]
 [ 70   5]]
Classification Report:               precision    recall  f1-score   support

         0.0       0.78      0.99      0.87       249
         1.0       0.62      0.07      0.12        75

    accuracy                           0.77       324
   macro avg       0.70      0.53      0.50       324
weighted avg       0.74      0.77      0.70       324
```

```
pred_train = LR.predict(x_train)
pred_test =LR.predict(x_test)
Train_accuracy = accuracy_score(y_train,pred_train)
Test_accuracy = accuracy_score(y_test,pred_test)
maxAccu=0
maxRS=0
```

```
from sklearn.model_selection import cross_val_score
for j in range(2,16):
    cv_score=cross_val_score(LR,x,y,cv=j)
    cv_mean = cv_score.mean()
    if cv_mean > maxAccu:
```

```
        maxAccu = cv_mean
        maxRS = j
    print(f"At cross fold {j} cv score is {cv_mean} and accuracy score training is {Train_


    print("\n")
```

```
 At cross fold 2 cv score is 0.7775510204081633 and accuracy score training is 0.7911!

 At cross fold 3 cv score is 0.7693914435626598 and accuracy score training is 0.7911!

 At cross fold 4 cv score is 0.7744897959183674 and accuracy score training is 0.7911!

 At cross fold 5 cv score is 0.7795918367346939 and accuracy score training is 0.7911!

 At cross fold 6 cv score is 0.7734550351638486 and accuracy score training is 0.7911!

 At cross fold 7 cv score is 0.7734693877551021 and accuracy score training is 0.7911!

 At cross fold 8 cv score is 0.7724160335865654 and accuracy score training is 0.7911!

 At cross fold 9 cv score is 0.7744553932117643 and accuracy score training is 0.7911!

 At cross fold 10 cv score is 0.7775510204081633 and accuracy score training is 0.791:

 At cross fold 11 cv score is 0.7806491885143572 and accuracy score training is 0.791:

 At cross fold 12 cv score is 0.773461808692161 and accuracy score training is 0.7911!

 At cross fold 13 cv score is 0.7795411605937923 and accuracy score training is 0.791:

 At cross fold 14 cv score is 0.7775510204081633 and accuracy score training is 0.791:

 At cross fold 15 cv score is 0.7765034965034966 and accuracy score training is 0.791:
```

```
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(LR,x,y,cv=j)
cv_mean = cv_score.mean()
print("Cross validation score for Logistic Regression",cv_mean)
```

```
Cross validation score for Logistic Regression 0.7765034965034966
```

```
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(dt,x,y,cv=j)
cv_mean = cv_score.mean()
print("Cross validation score for Decision Tree",cv_mean)
```

```
Cross validation score for Decision Tree 0.8032478632478633
```

```
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(rfc,x,y,cv=j)
cv_mean = cv_score.mean()
print("Cross validation score for Random Forest Classifier",cv_mean)
```

```
Cross validation score for Random Forest Classifier 0.7633255633255634
```

```
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(svm,x,y,cv=j)
cv_mean = cv_score.mean()
print("Cross validation score for Support Vector Machhine",cv_mean)
```

```
Cross validation score for Support Vector Machhine 0.7479564879564883
```

Lesser the diffrence between Accuracy and cross validation, Best the model

Decision Tree shows max accuracy

%Accuracyscore = accuracy - crossvalidation

Regularization

To mitigate the problem of overfitting and underfitting Regularization Methods are used: Lasso, Ridge or ElasticNet .

```
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
parameters = {'alpha':[.0001,.001,.01,.1,1,10],'random_state':list(range(0,10))}
EN=ElasticNet()
clf=GridSearchCV(EN,parameters)
clf.fit(x_train,y_train)

print(clf.best_params_)
```

```
{'alpha': 0.01, 'random_state': 0}
```

```
EN = ElasticNet(alpha=0.01,random_state=0)
EN.fit(x_train,y_train)
EN.score(x_train,y_train)
pred_EN=EN.predict(x_test)

lss= accuracy_score(y_test,pred_test)
lss
```

```
0.8209876543209876
```

```
#cross_validation_mean = cv_mean
#cross_validation_score= cv_score

cross_validation_score = cross_val_score(EN,x,y,cv=5)
cross_validation_mean = cross_validation_score.mean()
cross_validation_mean
```

```
0.17065039633155782
```

Ensemble Technique

```
from sklearn.model_selection import GridSearchCV

parameters = {'max_depth':np.arange(2,15),'criterion':["gini","entrophy"]}

rf = DecisionTreeClassifier()
clf=GridSearchCV(rf,parameters,cv=5)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 5}
```

```
rf=DecisionTreeClassifier(criterion="gini",max_depth=5)
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_decision=rf.predict(x_test)

rfs = accuracy_score(y_test,pred_decision)
```

```
print('Accuracy Score:',rfs*100)

rfscore=cross_val_score(rf,x,y,cv=5)
rfc=rfscore.mean()

print("Cross Validation Score:",rfc*100)

#print(clf.best_params_)
```

```
Accuracy Score: 84.25925925925925
Cross Validation Score: 82.14285714285714
```

Saving Model

```
import pickle
filename = "insurance.pkl"
pickle.dump(rf,open(filename,"wb"))
```

Conclusion

```
loaded_model=pickle.load(open('insurance.pkl','rb'))
result=loaded_model.score(x_test,y_test)
print(result)
```

```
0.8425925925925926
```

```
conclusion = pd.DataFrame([loaded_model.predict(x_test)[:],pred_decision[:]],index=["Predi
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	314	315	316	317	318
Predicted	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0
Original	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0

2 rows × 324 columns