Insurance Prediction

Problem

> Just like medical insurance, there is vehicle insurance where every year customer needs to pay a premium of certain amount to insurance provider company so that in case of unfortunate accident by the vehicle, the insurance provider company will provide a compensation (called 'sum assured') to the customer. Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company because it can then accordingly plan its communication strategy to reach out to those customers and optimise its business model and revenue.

Goal

> Build machine learning to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company.

Importing the libraries for the modeling.

In [1]:
```
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import confusion_matrix
```

In [2]:
```
#loading the dataset
df_edapickle.load(open('C:\\Users\\acer\\Desktop\\Vechicle_Insurance_Dataset\\models\\ExploratoryDataAnalysis.pkl', 'rb'))
df_eda.head()
```

Out[2]:

	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage	Response	Gender_Male	Vehicle_Age_< 1 Year	Vehicle_Age_> 2 Years	Vehicle_Damage_Yes
0	44	1	28.0	0	40454.0	26.0	217	1	1	0	1	1
1	76	1	3.0	0	33536.0	26.0	183	0	1	0	0	0
2	47	1	28.0	0	38294.0	26.0	27	1	1	0	1	1
3	21	1	11.0	1	28619.0	152.0	203	0	1	1	0	0
4	29	1	41.0	1	27496.0	152.0	39	0	0	1	0	0

In [3]:
```
df_eda.isnull().sum().sum()
```

Out[3]:
```
0
```

In [4]:
```
#independent and dependent variables
X=df_eda.drop(columns=['Response'],axis=1)
y=df_eda['Response']
```

In [5]:
```
#spliting the data into train and test data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
```

In [6]:
```
#shape of the train and test data
X_train.shape,X_test.shape,y_train.shape
```

Out[6]:
```
((259552, 11), (111237, 11), (259552,))
```

In [7]:
```
#standardizing data
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
X_train=pd.DataFrame(X_train)

X_test=scaler.fit_transform(X_test)
X_test=pd.DataFrame(X_test)
```

Building a Logistic Regression mode

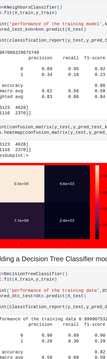
In [8]:
```
log_reg=LogisticRegression()
log_reg.fit(X_train,y_train)  #building a algorithm

print('performance of the traing data',log_reg.score(X_train,y_train))
y_pred_test_log=log_reg.predict(X_test)

print(classification_report(y_test,y_pred_test_log))
```

```
performance of the traing data 0.8775752006955974
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     97751
           1       0.00      0.00      0.00     13486

    accuracy                           0.88    111237
   macro avg       0.44      0.50      0.47    111237
weighted avg       0.77      0.88      0.82    111237
```

In [9]:
```
print(confusion_matrix(y_test,y_pred_test_log))
import seaborn as sns
sns.set(rc={'figure.facecolor':'red'})
sns.heatmap(confusion_matrix(y_test,y_pred_test_log),annot=True)
```

```
[[97748     3]
 [13486     0]]
<AxesSubplot:>
```

Out[9]:



Building a KNN Classifier model

In [10]:
```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)

print('performance of the training model',knn.score(X_train,y_train))
y_pred_test_knn=knn.predict(X_test)

print(classification_report(y_test,y_pred_test_knn))
```

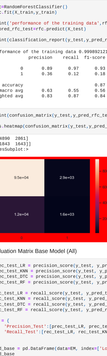
```
0.8967066329675749
              precision    recall  f1-score   support

           0       0.89      0.95      0.92     97751
           1       0.34      0.18      0.23     13486

    accuracy                           0.86    111237
   macro avg       0.62      0.56      0.58    111237
weighted avg       0.83      0.86      0.84    111237
```

In [11]:
```
[[93123  4628]
 [11110  2376]]
```

In [12]:
```
print(confusion_matrix(y_test,y_pred_test_knn))
sns.heatmap(confusion_matrix(y_test,y_pred_test_knn),annot=True)
```

```
[[93123  4628]
 [11110  2376]]
<AxesSubplot:>
```

Out[12]:



Building a Decision Tree Classifier model

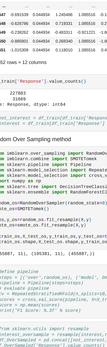
In [22]:
```
dtc=DecisionTreeClassifier()
dtc.fit(X_train,y_train)

print('performance of the training data',dtc.score(X_train,y_train))
y_pred_dtc_test=dtc.predict(X_test)

print(classification_report(y_test,y_pred_dtc_test))
```

```
performance of the training data 0.9999075329799839
              precision    recall  f1-score   support

           0       0.90      0.89      0.90     97751
           1       0.28      0.30      0.29     13486

    accuracy                           0.82    111237
   macro avg       0.59      0.60      0.59    111237
weighted avg       0.83      0.82      0.82    111237
```

In [23]:
```
print(confusion_matrix(y_test,y_pred_dtc_test))

sns.heatmap(confusion_matrix(y_test,y_pred_dtc_test),annot=True)
```

```
[[87426 10325]
 [ 9437  4049]]
<AxesSubplot:>
```

Out[23]:



Building a Random Forest Classifier model

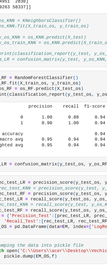
In [28]:
```
rfc=RandomForestClassifier()
rfc.fit(X_train,y_train)

print('performance of the training data',rfc.score(X_train,y_train))
y_pred_rfc_test=rfc.predict(X_test)

print(classification_report(y_test,y_pred_rfc_test))
```

```
performance of the training data 0.9998521214009878
              precision    recall  f1-score   support

           0       0.89      0.97      0.93     97751
           1       0.36      0.12      0.18     13486

    accuracy                           0.87    111237
   macro avg       0.63      0.55      0.56    111237
weighted avg       0.83      0.87      0.84    111237
```

In [29]:
```
print(confusion_matrix(y_test,y_pred_rfc_test))

sns.heatmap(confusion_matrix(y_test,y_pred_rfc_test),annot=True)
```

```
[[94890  2861]
 [11843  1643]]
<AxesSubplot:>
```

Out[29]:



Evaluation Matrix Base Model (All)

In [31]:
```
prec_test_LR = precision_score(y_test, y_pred_test_log)
prec_test_KNN = precision_score(y_test, y_pred_test_knn)
prec_test_DTC = precision_score(y_test, y_pred_dtc_test)
prec_test_RF = precision_score(y_test, y_pred_rfc_test)

rec_test_LR = recall_score(y_test, y_pred_test_log)
rec_test_KNN = recall_score(y_test, y_pred_test_knn)
rec_test_DTC = recall_score(y_test, y_pred_dtc_test)
rec_test_RF = recall_score(y_test, y_pred_rfc_test)

EM = {
    'Precision_Test':[prec_test_LR, prec_test_KNN, prec_test_DTC, prec_test_RF],
    'Recall_Test':[rec_test_LR, rec_test_KNN, rec_test_DTC, rec_test_RF]
}

mod_base = pd.DataFrame(data=EM, index=['LogReg','KNN_Clas','DTC','RF_Clas']).round(2)
mod_base
```

Out[31]:

	Precision_Test	Recall_Test
LogReg	0.00	0.00
KNN_Clas	0.34	0.18
DTC	0.28	0.30
RF_Clas	0.36	0.12

- The evaluation matrix's score is so bad. So far away from best score

Data Improvement

- As we know the target variable data are imbalanced, so we will try to do data improvement with Random Over Sampling and Random Under Sampling

In [45]:
```
df_train = pd.concat([X_train, y_train], axis=1)
df_train.dropna()
```

Out[45]:

	0	1	2	3	4	5	6	7	8	9	10	Response
0	-0.562032	0.044934	0.118010	1.085516	-0.057017	0.730568	-0.315536	-1.092674	1.138075	-0.205431	-1.006944	1.0
1	0.474032	0.044934	-1.761117	-0.921221	0.238916	-1.601813	0.711763	0.915639	-0.878677	-0.205431	0.993156	1.0
2	-0.844088	0.044934	-1.761117	-0.921221	-0.260164	0.212291	-1.061983	0.915639	-0.878677	-0.205431	0.993156	1.0
4	0.538786	0.044934	0.118010	1.085516	1.197796	-1.601813	0.210059	-1.092674	-0.878677	-0.205431	-1.006944	0.0
5	-0.756293	0.044934	-0.406146	1.085516	-1.807175	0.730568	-1.199491	0.915639	1.138075	-0.205431	-1.006944	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
259547	-0.891539	0.044934	1.243486	1.085516	-0.197488	0.730568	-0.614169	-1.092674	1.138075	-0.205431	-1.006944	0.0
259548	-0.938799	0.044934	0.718302	1.085516	0.279017	0.730568	-1.414937	-1.092674	1.138075	-0.205431	-1.006944	0.0
259549	-0.236260	0.044934	-0.483211	-0.921221	-1.807175	0.804612	0.874561	-1.092674	-0.878677	-0.205431	0.993156	0.0
259550	-0.895901	0.044934	0.268345	1.085516	-1.807175	0.730568	0.652037	-1.092674	1.138075	-0.205431	-1.006944	1.0
259551	-1.031506	0.044934	0.118010	1.085516	0.461383	0.730568	1.177632	0.915639	1.138075	-0.205431	-1.006944	0.0

176652 rows × 12 columns

In [46]:
```
df_train['Response'].value_counts()
```

```
0.0    157863
1.0     31699
Name: Response, dtype: int64
```

In [47]:
```
# not_interest = df_train[df_train['Response'] == 0] ## Majority
# interest = df_train[df_train['Response'] == 1] ## Minority
```

Random Over Sampling method

In [55]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

random_os=RandomOverSampler(random_state=0)
smote_os=SMOTETomek()

X_os,y_os=random_os.fit_resample(X,y)
smote_os=smote_os.fit_resample(X,y)

X_train_os,X_test_os,y_train_os,y_test_os=train_test_split(X_os,y_os,test_size=0.3,random_state=1)
X_train_os.shape,X_test_os.shape,y_train_os.shape
```

Out[55]:
```
((455887, 11), (195381, 11), (455887,))
```

In [73]:
```
# define pipeline
# steps = [('over',random_os), ('model', DecisionTreeClassifier())]
# pipeline = Pipeline(steps=steps)
# # evaluate pipeline
# cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# scores = cross_val_score(pipeline, X=X_train_os, y=y_train_os, scoring='f1_micro', cv=10, n_jobs=-1)
# score = np.mean(scores)
# print('f1 Score: %.3f' % score)
```

In [74]:
```
# from sklearn.utils import resample
# interest_oversample = resample(interest,replace=True,n_samples = len(not_interest),random_state = 42)
# df_OverSampled = pd.concat([not_interest, interest_oversample])
# df_OverSampled['Response'].value_counts()
```

In [56]:
```
#### LogReg Model Base + Random Over Sampling
os_LR = LogisticRegression()
os_LR.fit(X_train_os, y_train_os)
os_LR_pred = os_LR.predict(X_test_os, y_os_LR))  ## test data
print(classification_report(y_test_os, y_os_LR))
```

```
c:\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
              precision    recall  f1-score   support

           0       0.95      0.60      0.73     97600
           1       0.71      0.97      0.82     97781

    accuracy                           0.79    195381
   macro avg       0.83      0.78      0.78    195381
weighted avg       0.83      0.78      0.78    195381
```

In [57]:
```
cm_LR = confusion_matrix(y_test_os, y_os_LR, labels=[1, 0])
print(cm_LR)
```

```
[[94951  2830]
 [39263 58337]]
```

In [58]:
```
# os_KNN = KNeighborsClassifier()
# os_KNN.fit(X_train_os, y_train_os)

# y_os_KNN = os_KNN.predict(X_test)
# y_os_train_KNN = os_KNN.predict(X_train_os)

# print(classification_report(y_test, y_os_KNN)) ## test data
# cm_LR = confusion_matrix(y_test, y_os_KNN, labels=[1, 0])
```

In [59]:
```
os_RF = RandomForestClassifier()
os_RF.fit(X_train_os, y_train_os)
y_os_RF = os_RF.predict(X_test_os)
print(classification_report(y_test_os, y_os_RF))  ## test data
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     97600
           1       0.98      1.00      0.99     97781

    accuracy                           0.99    195381
   macro avg       0.99      0.99      0.99    195381
weighted avg       0.99      0.99      0.99    195381
```

In [60]:
```
cm_LR = confusion_matrix(y_test_os, y_os_RF, labels=[1, 0])
```

In [64]:
```
prec_test_LR = precision_score(y_test_os, y_os_LR)
#prec_test_KNN = precision_score(y_test, y_os_KNN)
prec_test_RF = precision_score(y_test_os, y_os_RF)
rec_test_LR = recall_score(y_test_os, y_os_LR)
#rec_test_KNN = recall_score(y_test, y_os_KNN)
rec_test_RF = recall_score(y_test_os, y_os_RF)
EM = {'Precision_Test':[prec_test_LR, prec_test_RF],
      'Recall_Test':[rec_test_LR, rec_test_RF]}
EM_OS = pd.DataFrame(data=EM, index=['LogReg_OS', 'RF_Clas_OS']).round(2)
```

In [70]:
```
#dumping the data into pickle file
with open('C:\\Users\\acer\\Desktop\\Vechicle_Insurance_Dataset\\models\\train_model.pkl', 'wb') as f:
    pickle.dump(EM_OS,f)
```