Group no: 327: Health Insurance Cross Sell Prediction

First name	Last Name	IIT Email
Abhishek	Singh	asingh135@hawk.iit.edu

Table of Contents

1. Introduction

The data set belong to the health insurance company which sells health insurances to their customers. So, they have provided there previous years clients data who have taken health insurance. After examining the data set, I found out, the health insurance company now want to sell vehicle insurance too. Now based on the data set I will build the model and analyze whether the customers are interested to buy vehicle insurance or not. The data set consist of 12 attributes and 381109 rows. The data set consist of train.csv file which consist of various dependent and independent variable. First all the pre-processing task of data cleaning and scaling will be done. Then, I will build various models and compare each model based on their evaluation metrics and pick a best model suited of the data set.

2. Data Sets

The data set is taken from Kaggle.com. The data set can be found from the link:
https://www.kaggle.com/anmolkumar/health-insurance-cross-sell-prediction .
It has 381109 rows and 12 attributes. The data consist of two files train.csv and test.csv. The following are the attributes in the data set.

- Id
  Unique id for each customer.

- Gender:
  Gender of the customer

- Age:
  Age of the customer

- Driving_License:
  0: Customer does not have DL. 1: Customer already has DL

- Region_Code:
  Unique code for the region of the customer

- Previously_Insured:
  1: Customer already has Vehicle Insurance. 0 : Customer doesn't have Vehicle Insurance

- Vehicle_Age:
  Age of the Vehicle

- Vehicle_Damage:
  1: Customer got his/her vehicle damaged in the past. 0: Customer didn't get his/her vehicle damaged in the past.

- Annual_Premium:
  The amount customer needs to pay as premium in the year

- Policy_Sales_Channel:
  Anonymized Code for the channel of outreaching to the customer ie. Different Agents, Over Mail, Over Phone, In Person, etc.

- Vintage:
  Number of Days, Customer has been associated with the company

- Response:
  1: Customer is interested. 0: Customer is not interested

The data set consist of one independent variable. i.e Response. All the other 11 attributes are dependent variables.

Qualitative	Quantitative
• Id • Gender • Driving_license • Region_Code • Previously_insured • Vehicle_Damage • Policysaleschannel • response	• Age • Vehicle Age • Annual premium • Vintage •

3. **Research Problems**

The main task of this project is to find those customers who have already taken health insurance loan and predict among them who are interested in taking vehicle insurance too. This helps us to find our target customers and after finding those customers how can we transform those potential customers into actual customers. The independent variable in this data set is **response**. This independent variable will help us find whether the customer is interested in vehicle insurance or not.
Some challenges which I have found out by examining the data set is:

1. Vehicle_Age is given in some form of range. I need to convert that in numeric structure. So that it can be used in modeling.
2. I need to convert the categorical variable into numeric values this can be done by converting them into dummy variables.
3. I will also do the necessary scaling of the attributes.
4. Feature selection and feature reduction will be done to find out which features affect

4. **Potential Solutions**

- After analyzing the data set, I found are independent variable i.e. response and our dependent variable i.e. id, gender, age, Driving_License, Region_Code, Previously_Insured, Vehicle_Age, Vehicle_Damage. Annual_Premium, Policy_Sales_Channel, Vintage.
- After doing all the pre-processing. I will analyze the correlation between the independent variable (response) and each independent variable.
- I will visualize our dataset and find out how our data set is distributed with respect to the independent variable i.e. response.

- I will use various algorithm to create models and use feature selection method to find out the best model.
- I will use Precision, recall, F1 score, accuracy and AUC to find out the best models and best features.

5. Data Preprocessing:

This is Data set which consist of 11 features. The independent variable is Response which means weather the person is interested to buy the insurance or not. We are going to create models to predict this feature.

```
df = pd.read_csv('C:/Data mining/Proj/train.csv')
```

```
df.head()
```

Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0

a. Finding out all null values.

We use the below code to find al the null values in our data set. We can see in the output there is no null values in our Data set.

```
##counting the number of na values in data frame
df.isnull().sum(axis = 0)
```

```
id                      0
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
Vintage                 0
Response                0
dtype: int64
```

b. Converting the categorical variable in the numeric values.

We will use one hot encoding to convert the categorical variables in to numeric values. We can see the below code convert the categorical variables i.e Gender_Age_Map, Vehicle_Damage, and Vehicale Age into numeric values. The final Output is displayed below.

```python
###hot enocoding

Gender_Age_Map  = {'Male':1,'Female':0}

df['Gender'] = df['Gender'].map(Gender_Age_Map)

Vehicle_Damage_Age_Map  = {'Yes':1,'No':0}

df['Vehicle_Damage'] = df['Vehicle_Damage'].map(Vehicle_Damage_Age_Map)


df=pd.get_dummies(df,drop_first=True)
df.head()
```

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
id										
1	1	44	1	28.0	0	1	40454.0	26.0	217	1
2	1	76	1	3.0	0	0	33536.0	26.0	183	0
3	1	47	1	28.0	0	1	38294.0	26.0	27	1
4	1	21	1	11.0	1	0	28619.0	152.0	203	0
5	0	29	1	41.0	1	0	27496.0	152.0	39	0

```python
df=df.rename(columns={"Vehicle_Age_< 1 Year": "Vehicle_Age_less_than_1_Year", "Vehicle_Age_> 2 Years":
                      "Vehicle_Age_greater_than_2_Years"})
df['Vehicle_Age_less_than_1_Year']=df['Vehicle_Age_less_than_1_Year'].astype('int')
df['Vehicle_Age_greater_than_2_Years']=df['Vehicle_Age_greater_than_2_Years'].astype('int')

df.head()
```

16]:

usly_Insured	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response	Vehicle_Age_less_than_1_Year	Vehicle_Age_greater_than_2_Years
0	1	40454.0	26.0	217	1	0	1
0	0	33536.0	26.0	183	0	0	0
0	1	38294.0	26.0	27	1	0	1
1	0	28619.0	152.0	203	0	1	0
1	0	27496.0	152.0	39	0	1	0

c. Min-Max Scaling

The below code converts the values into uniform scale. The output shows all the code is converted into scale between 0 and 1.

```python
from mlxtend.preprocessing import minmax_scaling

df_scale = minmax_scaling(df, columns=df.columns)
```

```python
df_scale.head()
```

Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
1.0	0.369231	1.0	0.538462	0.0	1.0	0.070366	0.154321	0.716263	1.0
1.0	0.861538	1.0	0.057692	0.0	0.0	0.057496	0.154321	0.598616	0.0
1.0	0.415385	1.0	0.538462	0.0	1.0	0.066347	0.154321	0.058824	1.0
1.0	0.015385	1.0	0.211538	1.0	0.0	0.048348	0.932099	0.667820	0.0
0.0	0.138462	1.0	0.788462	1.0	0.0	0.046259	0.932099	0.100346	0.0

6. Data Visualization

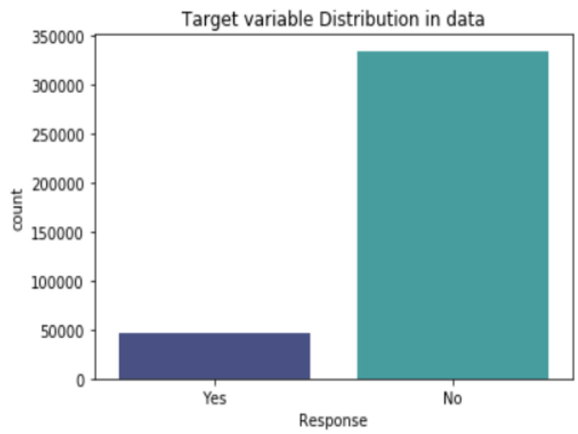
For Visualization purpose we have used the original dataset and not the data set after the preprocessing. This is done just for better Visualization purpose.

a. Total Count of independent variable.

The below graph shows how our independent variable which is response is distributed. More than 300,000 customers were not interested to buy the Vehicle insurance. Close to 50,000 people were interested to buy the insurance.

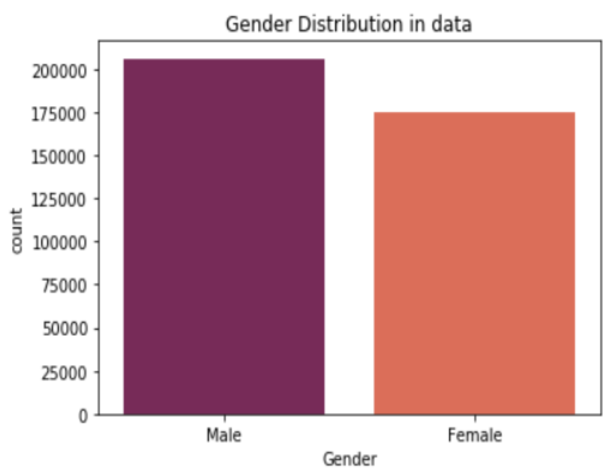
```
df_vis = pd.read_csv('C:/Data mining/Proj/train.csv')
cols=['Driving_License','Previously_Insured','Response']
for col in cols:
    df_vis[col] = df_vis[col].map({0:'No',1:'Yes'})
```

```
###response visualization

sns.countplot(df_vis['Response'],palette='mako')
plt.title("Target variable Distribution in data");
```



b. Gender Distribution

The below graph shows that our customers are almost evenly distributed between the gender. The code is displayed below.



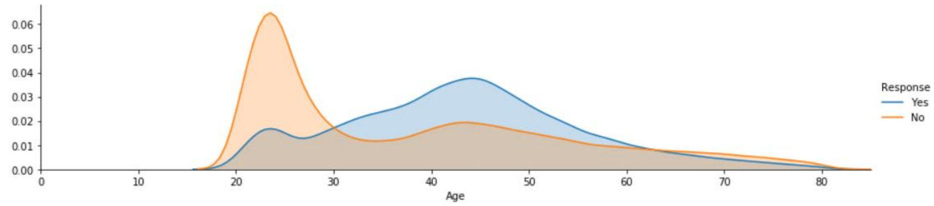

```
##gender distribution

sns.countplot(df_vis['Gender'],palette='rocket')
plt.title("Gender Distribution in data");
```

c. Age distribution

In the below graph we can see that the customers who were not interested to buy the insurance were between 20-30. Also, the customers who were interested to buy the insurance were distributed in the range of 30 to 50.

```python
####Age distribution


print("Age distribution according to Response")
facetgrid = sns.FacetGrid(df_vis,hue="Response",aspect = 4)
facetgrid.map(sns.kdeplot,"Age",shade = True)
facetgrid.set(xlim = (0,df_vis["Age"].max()))
facetgrid.add_legend();
```

```
Age distribution according to Response
```

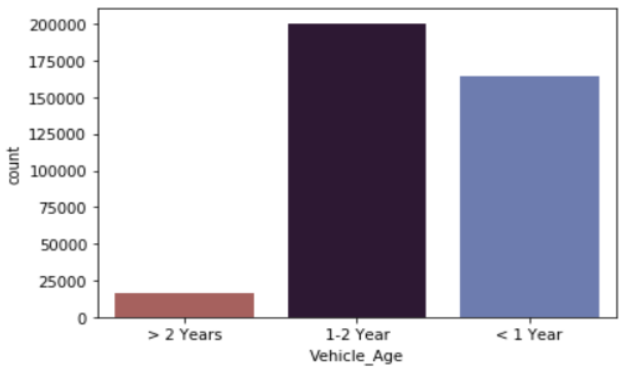


d. Vehicle Age Distribution.

The below graph shows us that the majority of the car which the customer owned were new car. As most of the car belonged in the group of 0 – 2 years.

```python
sns.countplot(df_vis.Vehicle_Age, palette='twilight_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25440d63488>
```

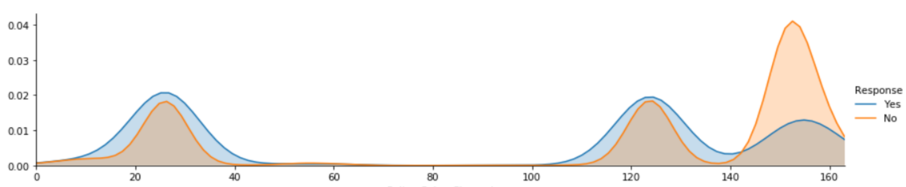


e. Policy Sales Channel

The below graph shows from which channels what were the response of the customers.

```python
print("Policy_Sales_Channel distribution according to Response")
facetgrid = sns.FacetGrid(df_vis,hue="Response",aspect = 4)
facetgrid.map(sns.kdeplot,"Policy_Sales_Channel",shade = True)
facetgrid.set(xlim = (0,df_vis["Policy_Sales_Channel"].max()))
facetgrid.add_legend();
```

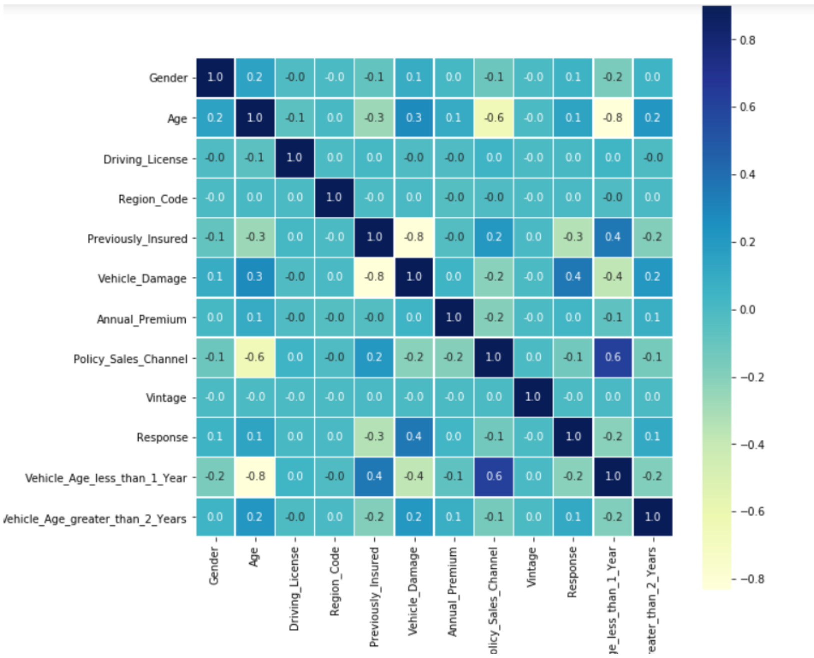
```
Policy_Sales_Channel distribution according to Response
```



7. Data Modeling

a. Correlation Metrics

The below graph shows us the correlation between all the features. There is a strong negative correlation between Vehicle damage and previously insured. Similarly there are various other correlation which we can see in the correlation metrics.



b. Test and Train split.

We have created test and train split to train and test the data set. I have used 20% test set and 80% training set.

```
X_train, X_test, y_train, y_test = train_test_split(df_scale.drop(['Response'], axis = 1),
                                                     df_scale['Response'], test_size = 0.2,random_state = 789)
```

```
print(f"Target variable disribution in train set: \n{y_train.value_counts()}\n\nand in test set: \n{y_test.value_counts()}")
```

```
Target variable disribution in train set:
0.0    267521
1.0     37366
Name: Response, dtype: int64

and in test set:
0.0    66878
1.0     9344
Name: Response, dtype: int64
```

c. Up Sampling

Since we can see our two variables in the response which is our independent variable are not uniformly distributed. This can be seen in below code which shows us that our train consist of 267,521 zeros and only 37,366 one’s. This means if we train our model on this data set. Then our classification will be highly biased towards zero and it will classify majority of our data set to zero. To Overcome this problem, we will use Up sampling and dummy rows to one’s. So that class is not imbalanced, and we can get correct classification. After performing Up Sampling we can see that both the class now consist equal number of rows which is 267,521.

```
print(f"Target variable disribution in train set
```

```
Target variable disribution in train set:
0.0    267521
1.0     37366
Name: Response, dtype: int64
```

```
#combining train features and target
####Sampling
df_fs = pd.concat([X_train_fs,y_train_fs],axis=1)

from sklearn.utils import resample,shuffle
df_majority = df_fs[df_fs['Response']==0]
df_minority = df_fs[df_fs['Response']==1]
df_minority_upsampled = resample(df_minority,replace=True,
                                 n_samples=y_train_fs.value_counts()[0],random_state = 123)
balanced_df = pd.concat([df_minority_upsampled,df_majority])
balanced_df = shuffle(balanced_df)
balanced_df.Response.value_counts()
```

```
1.0    267521
0.0    267521
Name: Response, dtype: int64
```

d. Logistic Regression

After applying Logistic regression model to our sampled data set. I found the accuracy score 64 %, precision value for ‘0’ is 99% and recall value for ‘1’ was very high which was 97%. On the contrary, recall value for ‘0’ was comparatively less which was 59% and the precision value for ‘1’ was also very less which was 25%. The area under the curve is 83.63%.

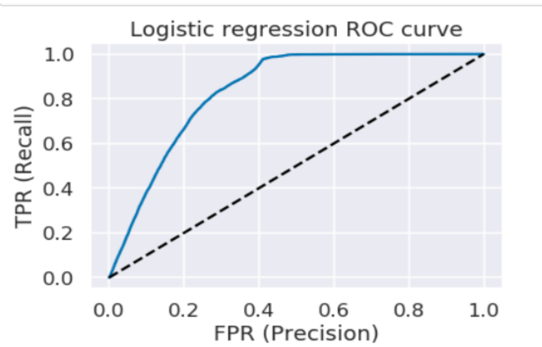
```
###logistic after sampling
logisticRegression = LogisticRegression(max_iter = 10000)
GLM_fit_sampling=logisticRegression.fit(X_train_sampling, Y_train_sampling)
GLM_clas = pd.DataFrame(GLM_fit_sampling.predict(X_test_fs))
GLM_probability = pd.DataFrame(GLM_fit_sampling.predict_proba(X_test_fs))

predictions = logisticRegression.predict(X_test_fs)
```

```
print(f"Accuracy score is {100*accuracy_score(y_test_fs,predictions).round(2)}
\nROC-AUC score is {100*roc_auc_score(y_test_fs,predictions).round(2)}")
```

```
Accuracy score is 64.0
ROC-AUC score is 78.0
```

```
print(classification_report(y_test_fs,predictions ))
```

	precision	recall	f1-score	support
0.0	0.99	0.59	0.74	66878
1.0	0.25	0.97	0.40	9344
accuracy			0.64	76222
macro avg	0.62	0.78	0.57	76222
weighted avg	0.90	0.64	0.70	76222




```
Area under curve (AUC):  0.83693
```

e. Random Forest

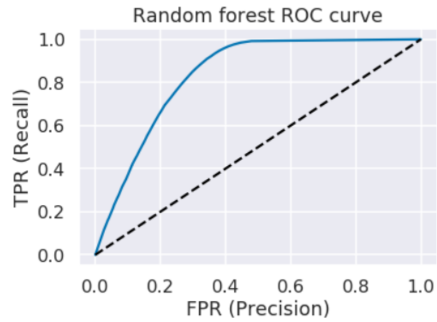
After applying Random Forest model to our sampled data set. I found the accuracy score 85%, precision value for ‘0’ is 90% and recall value for ‘0’ was very high which was 93%. On the contrary, recall value for ‘1’ was comparatively less which was 27% and the precision value for ‘1’ was also very less which was 34%. The area under the curve is 83.59%.

```
###random Forest after sampling
rfc = RandomForestClassifier(n_estimators=100)
rfc_fit=rfc.fit(X_train_sampling, Y_train_sampling)
rfc_pred = rfc.predict(X_test_fs)
print(f"Accuracy score is {100*accuracy_score(y_test_fs,rfc_pred).round(2)}
\nROC-AUC score is {100*roc_auc_score(y_test_fs,rfc_pred).round(2)}")
```

```
Accuracy score is 85.0
ROC-AUC score is 60.0
```

```
print(classification_report(y_test_fs,rfc_pred ))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.91     66878
         1.0       0.34      0.27      0.30      9344

    accuracy                           0.85     76222
   macro avg       0.62      0.60      0.61     76222
weighted avg       0.83      0.85      0.84     76222
```




```
Area under curve (AUC):  0.83595
```

f. XGB Classifier

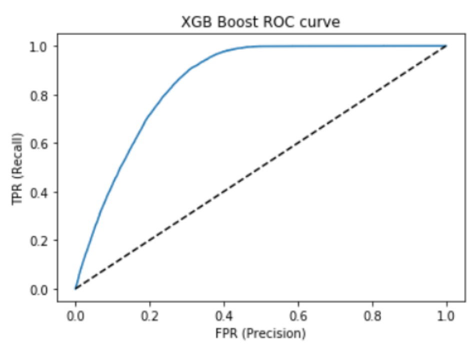
After applying XGB Classifier model to our sampled data set. I found the accuracy score 71%, precision value for ‘0’ is 98% and recall value for ‘0’ was moderate which was 69%. On the contrary, recall value for ‘1’ was comparatively high which was 91% and the precision value for ‘1’ was very less which was 29%. The area under the curve is 85.52%.

```
##XGB classisfier
xgb = XGBClassifier()
xgb_fit=xgb.fit(X_train_sampling, Y_train_sampling)
xgb_pred = xgb.predict(X_test_fs)
print(f"Accuracy score is {100*accuracy_score(y_test_fs,xgb_pred).round(2)}
\nROC-AUC score is {100*roc_auc_score(y_test_fs,xgb_pred).round(2)}")
```

```
Accuracy score is 71.0
ROC-AUC score is 80.0
```

```
print(classification_report(y_test_fs,xgb_pred ))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.69      0.81     66878
         1.0       0.29      0.91      0.44      9344

    accuracy                           0.71     76222
   macro avg       0.64      0.80      0.62     76222
weighted avg       0.90      0.71      0.76     76222
```




```
Area under curve (AUC):  0.8552
```

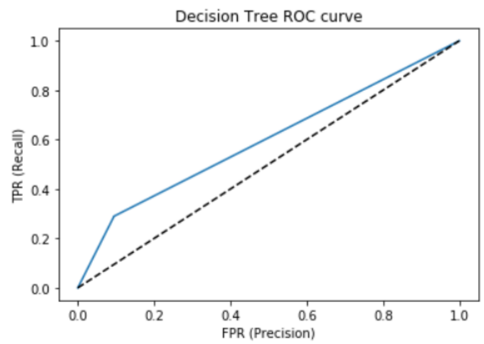
g. Decision Tree

After applying Decision tree model to our sampled data set. I found the accuracy score 83%, precision value for ‘0’ is 90% and recall value for ‘0’ was high which was 91%. On the contrary, recall value for ‘1’ was comparatively less which was 29% and the precision value for ‘1’ was very less which was 30%. The area under the curve is 59.74%.

```
####decision treee after sampling
model= tree.DecisionTreeClassifier(random_state=456)

model.fit(X_train_sampling, Y_train_sampling)

Prediction_for_DT = model.predict(X_test_fs)
```

```
print(classification_report(y_test_fs,Prediction_for_DT ))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.91      0.90     66878
         1.0       0.30      0.29      0.29      9344

    accuracy                           0.83     76222
   macro avg       0.60      0.60      0.60     76222
weighted avg       0.83      0.83      0.83     76222
```




```
Area under curve (AUC):  0.59744
```

h. KNN

After applying KNN model to our sampled data set. I calculated the error rate for all the value between K= 1 to 40. After finding the error rate all the k value. The least error was for K= 39. So, I calculated the Precision and recall for the same. For K=39, I found the accuracy score 71%, precision value for ‘0’ is 98% and recall value for ‘0’ was moderate which was 68%. On the contrary, recall value for ‘1’ was comparatively high which was 88% and the precision value for ‘1’ was very less which was 28%.

```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train_sampling, Y_train_sampling)
    pred_i = knn.predict(X_test_fs)
    error.append(np.mean(pred_i != y_test_fs))
```

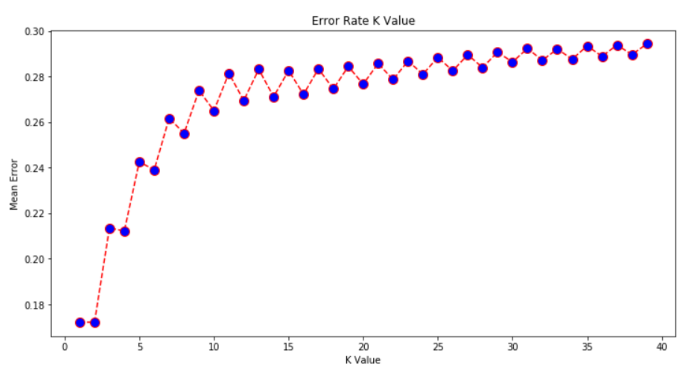
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Text(0, 0.5, 'Mean Error')
```

```
classifier=KNeighborsClassifier(n_neighbors=39)

classifier.fit(X_train_sampling, Y_train_sampling)

y_pred = classifier.predict(X_test_fs)
```

```
print(classification_report(y_test_fs, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.68      0.80     66878
         1.0       0.28      0.88      0.42      9344

    accuracy                           0.71     76222
   macro avg       0.63      0.78      0.61     76222
weighted avg       0.89      0.71      0.76     76222
```



i. Feature Selection

I performed the forward and backward feature selection. By analyzing both the feature selection technique I found that there were three features which were decreasing the overall accuracy of the data set. The three feature which I removed are ‘Vintage’, ‘Annual Premium’ and ‘Region_Code’. The code below shows the output of forward and backward feature selection.

1. Forward Feature Selection

```python
sfs1 = SFS(classifier_pipeline,
           k_features=11,
           forward=True,
           scoring='accuracy',
           cv=cv)

# Perform SFFS
sfs1.fit(fs_inputs, fs_target)
sfs1.subsets_
```

```
{1: {'feature_idx': (0,),
  'cv_scores': array([0.87830285, 0.87420955, 0.8780667 , 0.87956233, 0.8768597 ,
         0.87701713, 0.8744457 , 0.87790927, 0.87712209, 0.88087116]),
  'avg_score': 0.8774366478737436,
  'feature_names': ('Gender',)},
 2: {'feature_idx': (0, 1),
  'cv_scores': array([0.87830285, 0.87420955, 0.8780667 , 0.87956233, 0.8768597 ,
         0.87701713, 0.8744457 , 0.87790927, 0.87712209, 0.88087116]),
  'avg_score': 0.8774366478737436,
  'feature_names': ('Gender', 'Age')},
 3: {'feature_idx': (0, 1, 10),
  'cv_scores': array([0.87830285, 0.87420955, 0.87809294, 0.87958857, 0.8768597 ,
         0.87701713, 0.8744457 , 0.87790927, 0.87712209, 0.88084492]),
  'avg_score': 0.8774392717192478,
  'feature_names': ('Gender', 'Age', 'Vehicle_Age_greater_than_2_Years')},
 4: {'feature_idx': (0, 1, 4, 10),
  'cv_scores': array([0.87830285, 0.87420955, 0.87809294, 0.87958857, 0.8768597 ,
         0.87701713, 0.8744457 , 0.87790927, 0.87709585, 0.88084492]),
  'avg_score': 0.8774366478048925,
```

```
 5: {'feature_idx': (0, 1, 4, 5, 10),
  'cv_scores': array([0.87830285, 0.87420955, 0.87809294, 0.87945737, 0.8768597 ,
         0.87701713, 0.8744457 , 0.87790927, 0.87709585, 0.88084492]),
  'avg_score': 0.8774235282331153,
  'feature_names': ('Gender',
   'Age',
   'Previously_Insured',
   'Vehicle_Damage',
   'Vehicle_Age_greater_than_2_Years')},
 6: {'feature_idx': (0, 1, 4, 5, 9, 10),
  'cv_scores': array([0.87825037, 0.87420955, 0.87801422, 0.87961481, 0.87696466,
         0.87704337, 0.87457689, 0.87780431, 0.87714833, 0.8808974 ]),
  'avg_score': 0.8774523914287273,
  'feature_names': ('Gender',
   'Age',
   'Previously_Insured',
   'Vehicle_Damage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
```

```
 7: {'feature_idx': (0, 1, 2, 4, 5, 9, 10),
  'cv_scores': array([0.87817166, 0.87415707, 0.87809294, 0.87953609, 0.87691218,
         0.8769909 , 0.87457689, 0.87777807, 0.87701713, 0.88092364]),
  'avg_score': 0.8774156566966023,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Previously_Insured',
   'Vehicle_Damage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
 8: {'feature_idx': (0, 1, 2, 4, 5, 7, 9, 10),
  'cv_scores': array([0.87465561, 0.87137572, 0.87410459, 0.87633492, 0.87297631,
         0.87308126, 0.87155939, 0.87575766, 0.873711  , 0.8779323 ]),
  'avg_score': 0.8741488754750628,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Previously_Insured',
   'Vehicle_Damage',
```

```
 9: {'feature_idx': (0, 1, 2, 3, 4, 5, 7, 9, 10),
  'cv_scores': array([0.86505208, 0.86187715, 0.86321534, 0.86683635, 0.86334654,
         0.86290047, 0.86245441, 0.86636404, 0.86408124, 0.86788245]),
  'avg_score': 0.8644010072746596,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Region_Code',
   'Previously_Insured',
   'Vehicle_Damage',
   'Policy_Sales_Channel',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
```

```
 10: {'feature_idx': (0, 1, 2, 3, 4, 5, 7, 8, 9, 10),
  'cv_scores': array([0.85416284, 0.84807536, 0.85119782, 0.85397917, 0.84983338,
         0.85161764, 0.84988586, 0.85240482, 0.85062056, 0.85499869]),
  'avg_score': 0.8516776127592769,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Region_Code',
   'Previously_Insured',
   'Vehicle_Damage',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
```

```
 11: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
  'cv_scores': array([0.86610165, 0.86240193, 0.86620661, 0.8690142 , 0.86623285,
         0.86481593, 0.86395004, 0.86599669, 0.86555063, 0.87063763]),
  'avg_score': 0.8660908153489227,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Region_Code',
   'Previously_Insured',
   'Vehicle_Damage',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')}}
```

2. Backward Feature Selection.

```python
cv=KFold(n_splits=10,shuffle=False)
classifier_pipeline= make_pipeline(RandomForestClassifier(n_estimators=100, n_jobs=-1))
```

```python
sfs1 = SFS(classifier_pipeline,
           k_features=1,
           forward=False,
           scoring='accuracy',
           cv=cv)

# Perform SFFS
sfs1.fit(fs_inputs, fs_target)
sfs1.subsets_
```

```
{11: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
  'cv_scores': array([0.86702002, 0.86177219, 0.86641652, 0.86985385, 0.8669413 ,
         0.86494713, 0.86436987, 0.86636404, 0.86607541, 0.87071635]),
  'avg_score': 0.866447667907815,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Region_Code',
   'Previously_Insured',
   'Vehicle_Damage',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
 10: {'feature_idx': (0, 1, 2, 3, 4, 6, 7, 8, 9, 10),
  'cv_scores': array([0.86597045, 0.86297919, 0.86610165, 0.86969641, 0.86486841,
         0.86523576, 0.86473721, 0.86652148, 0.86591798, 0.87066387]),
  'avg_score': 0.8662692415939433,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Region_Code',
   'Previously_Insured',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
```

```
 9: {'feature_idx': (0, 1, 2, 3, 6, 7, 8, 9, 10),
  'cv_scores': array([0.86833198, 0.86431739, 0.86914539, 0.87040487, 0.86738737,
         0.86670515, 0.866469  , 0.86967017, 0.86738737, 0.87268433]),
  'avg_score': 0.8682503022338297,
  'feature_names': ('Gender',
   'Age',
   'Driving_License',
   'Region_Code',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
 8: {'feature_idx': (0, 1, 3, 6, 7, 8, 9, 10),
  'cv_scores': array([0.86835822, 0.8633203 , 0.86812206, 0.8706935 , 0.86757104,
         0.86801711, 0.86678387, 0.87011624, 0.86757104, 0.87257938]),
  'avg_score': 0.8683132759029558,
  'feature_names': ('Gender',
   'Age',
   'Region_Code',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year',
   'Vehicle_Age_greater_than_2_Years')},
```

```
7: {'feature_idx': (0, 1, 3, 6, 7, 8, 9),
  'cv_scores': array([0.8687518 , 0.8639238 , 0.8687518 , 0.87145444, 0.86762352,
         0.86796463, 0.86599669, 0.86812206, 0.86757104, 0.87184466]),
  'avg_score': 0.868200445657842,
  'feature_names': ('Gender',
   'Age',
   'Region_Code',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage',
   'Vehicle_Age_less_than_1_Year')},
 6: {'feature_idx': (0, 1, 3, 6, 7, 8),
  'cv_scores': array([0.86880428, 0.86436987, 0.8681483 , 0.8706935 , 0.86780719,
         0.86683635, 0.86547191, 0.86809583, 0.86709874, 0.87189714]),
  'avg_score': 0.8679223108738681,
  'feature_names': ('Gender',
   'Age',
   'Region_Code',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage')},
```

```
5: {'feature_idx': (1, 3, 6, 7, 8),
  'cv_scores': array([0.86631156, 0.86279552, 0.86660019, 0.86817454, 0.86686259,
         0.86507832, 0.86457978, 0.86570806, 0.864055  , 0.87029651]),
  'avg_score': 0.8660462079098163,
  'feature_names': ('Age',
   'Region_Code',
   'Annual_Premium',
   'Policy_Sales_Channel',
   'Vintage')},
 4: {'feature_idx': (1, 3, 6, 8),
  'cv_scores': array([0.86263808, 0.85893836, 0.86339902, 0.86602293, 0.86182467,
         0.86061767, 0.85864973, 0.86255937, 0.86145732, 0.86659669]),
  'avg_score': 0.8622703854443434,
  'feature_names': ('Age', 'Region_Code', 'Annual_Premium', 'Vintage')},
 3: {'feature_idx': (1, 6, 8),
  'cv_scores': array([0.85888589, 0.85397917, 0.86011913, 0.86009289, 0.85710162,
         0.85809871, 0.85397917, 0.8578888 , 0.85770512, 0.86329047]),
  'avg_score': 0.8581140964300988,
  'feature_names': ('Age', 'Annual_Premium', 'Vintage')},
```

```
 2: {'feature_idx': (1, 8),
  'cv_scores': array([0.87591509, 0.87287135, 0.87659731, 0.8775944 , 0.87523287,
         0.8755215 , 0.87226785, 0.87628244, 0.87570518, 0.87874574]),
  'avg_score': 0.8756733718499541,
  'feature_names': ('Age', 'Vintage')},
 1: {'feature_idx': (8,),
  'cv_scores': array([0.87830285, 0.87420955, 0.8780667 , 0.87956233, 0.8768597 ,
         0.87701713, 0.8744457 , 0.87790927, 0.87712209, 0.88087116]),
  'avg_score': 0.8774366478737436,
  'feature_names': ('Vintage',)}}
```

j. Logistic Regression after FS

After applying Logistic regression model to our sampled and feature selected data set. I found the accuracy score 64 %, precision value for ‘0’ is 99% and recall value for ‘1’ was very high which was 97%. On the contrary, recall value for ‘0’ was comparatively less which was 59% and the precision value for ‘1’ was also very less which was 25%. The area under the curve is 83.61%.

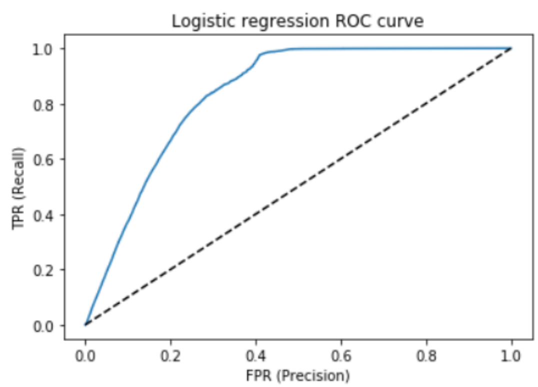
```
###logistic after sampling and fs
logisticRegression = LogisticRegression(max_iter = 10000)
GLM_fit_sampling=logisticRegression.fit(X_train_fs1, Y_train_sampling)
GLM_clas = pd.DataFrame(GLM_fit_sampling.predict(X_test_fs1))
GLM_probability = pd.DataFrame(GLM_fit_sampling.predict_proba(X_test_fs1))

predictions = logisticRegression.predict(X_test_fs1)
```

```
print(f"Accuracy score is {100*accuracy_score(y_test_fs,predictions).round(2)}
\nROC-AUC score is {100*roc_auc_score(y_test_fs,predictions).round(2)}")
```

```
Accuracy score is 64.0
ROC-AUC score is 78.0
```

```
print(classification_report(y_test_fs,predictions ))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.59      0.74     66878
         1.0       0.25      0.97      0.40      9344

    accuracy                           0.64     76222
   macro avg       0.62      0.78      0.57     76222
weighted avg       0.90      0.64      0.70     76222
```




```
Area under curve (AUC):  0.83612
```

k. XGB Classifier after Fs

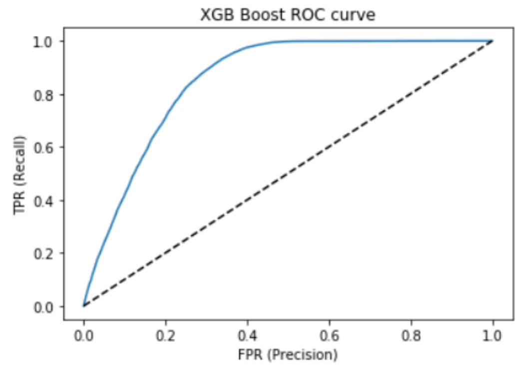
After applying XGB model to our sampled and feature selected data set. I found the accuracy score 71%, precision value for ‘0’ is 98% and recall value for ‘1’ was very high which was 99%. On the contrary, recall value for ‘0’ was comparatively less which was 68% and the precision value for ‘1’ was also very less which was 28%. The area under the curve is 85.43%.

```
xgb = XGBClassifier()
xgb_fit=xgb.fit(X_train_fs1, Y_train_sampling)
xgb_pred = xgb.predict(X_test_fs1)
print(f"Accuracy score is {100*accuracy_score(y_test_fs,xgb_pred).round(2)}\nROC-AUC score is {100*roc_auc_score(y_test_fs
```

```
Accuracy score is 71.0
ROC-AUC score is 80.0
```

```
print(classification_report(y_test_fs,xgb_pred ))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.68      0.80     66878
         1.0       0.28      0.91      0.43      9344

    accuracy                           0.71     76222
   macro avg       0.63      0.80      0.62     76222
weighted avg       0.90      0.71      0.76     76222
```




```
Area under curve (AUC):  0.85434
```

l. Random Forest after FS

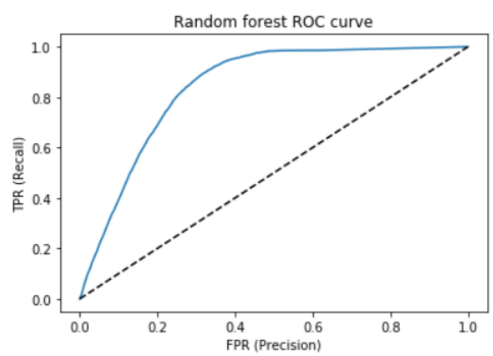
After applying Random Forest model to our sampled and feature selected data set. I found the accuracy score 72%, precision value for ‘0’ is 98% and recall value for ‘1’ was very high which was 88%. On the contrary, recall value for ‘0’ was comparatively less which was 69% and the precision value for ‘1’ was also very less which was 29%. The area under the curve is 84.02%.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100)
rfc_fit=rfc.fit(X_train_fs1, Y_train_sampling)
rfc_pred = rfc.predict(X_test_fs1)
print(f"Accuracy score is {100*accuracy_score(y_test_fs,rfc_pred).round(2)}\nROC-AUC score is {100*roc_auc_score(y_test_fs
```

```
Accuracy score is 72.0
ROC-AUC score is 79.0
```

```
print(classification_report(y_test_fs,rfc_pred ))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.69      0.81     66878
         1.0       0.29      0.88      0.43      9344

    accuracy                           0.72     76222
   macro avg       0.63      0.79      0.62     76222
weighted avg       0.89      0.72      0.76     76222
```




```
Area under curve (AUC):  0.84028
```

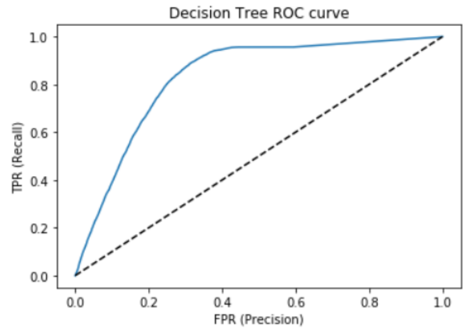
m. Decision Tree.

After applying Decision Tree model to our sampled and feature selected data set. I found the accuracy score 72%, precision value for ‘0’ is 98% and recall value for ‘1’ was very which 88%. On the contrary, recall value for ‘0’ was comparatively less which was 69% and the precision value for ‘1’ was also very less which was 29%. The area under the curve is 84.93%.

```
##decision tree after fs

model= tree.DecisionTreeClassifier(random_state=456)

model.fit(X_train_fs1, Y_train_sampling)

Prediction_for_DT = model.predict(X_test_fs1)
print(f"Accuracy score is {100*accuracy_score(y_test_fs,Prediction_for_DT).round(2)}\nROC-AUC score is {100*roc_auc_score(y_
```

```
Accuracy score is 72.0
ROC-AUC score is 79.0
```

```
print(classification_report(y_test_fs,Prediction_for_DT ))
              precision    recall  f1-score   support

         0.0       0.98      0.69      0.81     66878
         1.0       0.29      0.88      0.43      9344

    accuracy                           0.72     76222
   macro avg       0.63      0.79      0.62     76222
weighted avg       0.89      0.72      0.76     76222
```




```
Area under curve (AUC):  0.82934
```

8. Comparison of the Models

In this I have compared all the model after sampling and Feature Selection. To show exactly what was the effect of Sampling on our data set. I have created models for before sampling too and added in the comparesion.

a. Logistic Regression

Before Sampling and Fs

```
print(classification_report(y_test, predictions))
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93     66878
         1.0       0.00      0.00      0.00      9344

    accuracy                           0.88     76222
   macro avg       0.44      0.50      0.47     76222
weighted avg       0.77      0.88      0.82     76222
```

After Sampling

```
print(classification_report(y_test_fs,predictions ))
              precision    recall  f1-score   support

         0.0       0.99      0.59      0.74     66878
         1.0       0.25      0.97      0.40      9344

    accuracy                           0.64     76222
   macro avg       0.62      0.78      0.57     76222
weighted avg       0.90      0.64      0.70     76222
```

After Sampling and Fs

```
print(classification_report(y_test_fs,predictions ))
              precision    recall  f1-score   support

         0.0       0.99      0.59      0.74     66878
         1.0       0.25      0.97      0.40      9344

    accuracy                           0.64     76222
   macro avg       0.62      0.78      0.57     76222
weighted avg       0.90      0.64      0.70     76222
```

b. Random Forest

Before Sampling and Fs

```
print(classification_report(y_test, rfc_pred))
              precision    recall  f1-score   support

         0.0       0.89      0.97      0.93     66878
         1.0       0.37      0.12      0.19      9344

    accuracy                           0.87     76222
   macro avg       0.63      0.55      0.56     76222
weighted avg       0.82      0.87      0.84     76222
```

After Sampling

```
print(classification_report(y_test_fs,rfc_pred ))
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.91     66878
         1.0       0.34      0.27      0.30      9344

    accuracy                           0.85     76222
   macro avg       0.62      0.60      0.61     76222
weighted avg       0.83      0.85      0.84     76222
```

After Sampling and Fs

```
print(classification_report(y_test_fs,rfc_pred ))
              precision    recall  f1-score   support

         0.0       0.98      0.69      0.81     66878
         1.0       0.29      0.88      0.43      9344

    accuracy                           0.72     76222
   macro avg       0.63      0.79      0.62     76222
weighted avg       0.89      0.72      0.76     76222
```

c. KNN

Before Sampling and Fs

	precision	recall	f1-score	support
0.0	0.88	0.99	0.93	66878
1.0	0.42	0.03	0.06	9344
accuracy			0.88	76222
macro avg	0.65	0.51	0.50	76222
weighted avg	0.82	0.88	0.83	76222

After Sampling

	precision	recall	f1-score	support
0.0	0.98	0.68	0.80	66878
1.0	0.28	0.88	0.42	9344
accuracy			0.71	76222
macro avg	0.63	0.78	0.61	76222
weighted avg	0.89	0.71	0.76	76222

After Sampling and Fs

	precision	recall	f1-score	support
0.0	0.98	0.68	0.80	66878
1.0	0.28	0.88	0.42	9344
accuracy			0.71	76222
macro avg	0.63	0.78	0.61	76222
weighted avg	0.89	0.71	0.76	76222

d. Decision tree

Before Sampling and Fs

```
print(classification_report(y_test, Prediction_for_DT))
```

	precision	recall	f1-score	support
0.0	0.90	0.90	0.90	66878
1.0	0.29	0.31	0.30	9344
accuracy			0.82	76222
macro avg	0.60	0.60	0.60	76222
weighted avg	0.83	0.82	0.83	76222

After Sampling

```
print(classification_report(y_test_fs,Prediction_for_DT ))
```

	precision	recall	f1-score	support
0.0	0.90	0.91	0.90	66878
1.0	0.30	0.29	0.29	9344
accuracy			0.83	76222
macro avg	0.60	0.60	0.60	76222
weighted avg	0.83	0.83	0.83	76222

After Sampling and Fs

```
print(classification_report(y_test_fs,Prediction_for_DT ))
```

	precision	recall	f1-score	support
0.0	0.98	0.69	0.81	66878
1.0	0.29	0.88	0.43	9344
accuracy			0.72	76222
macro avg	0.63	0.79	0.62	76222
weighted avg	0.89	0.72	0.76	76222

e. XGB Classifier

Before Sampling and Fs

```
print(classification_report(y_test, xgb_pred))
```

	precision	recall	f1-score	support
0.0	0.88	1.00	0.93	66878
1.0	0.46	0.02	0.05	9344
accuracy			0.88	76222
macro avg	0.67	0.51	0.49	76222
weighted avg	0.83	0.88	0.83	76222

After Sampling

```
print(classification_report(y_test_fs,xgb_pred ))
```

	precision	recall	f1-score	support
0.0	0.98	0.69	0.81	66878
1.0	0.29	0.91	0.44	9344
accuracy			0.71	76222
macro avg	0.64	0.80	0.62	76222
weighted avg	0.90	0.71	0.76	76222

After Sampling and Fs

```
print(classification_report(y_test_fs,xgb_pred ))
```

	precision	recall	f1-score	support
0.0	0.98	0.68	0.80	66878
1.0	0.28	0.91	0.43	9344
accuracy			0.71	76222
macro avg	0.63	0.80	0.62	76222
weighted avg	0.90	0.71	0.76	76222

9. Conclusion

- After analyzing the dataset and all the models I have found many problems such as class imbalance issue and redundant columns.
- To Overcome the problem of imbalance issue I have used oversampling and to remove redundant columns I have used Feature Selection.
- Feature Selection increased the accuracy of some model, but it was not much of use. This maybe because the number of features is very less.
- The overall best models are Decision Tree and random Forest. They have overall same Accuracy and similar Precision and recall values.