Capstone Project

Health Insurance Cross Sell Prediction

Team Members
Ajinkya Dakhale
Harshjot Singh
Suvir Kapse

Content

- Problem Statement
- Data Summary
- Data Visualization(EDA)
- Feature Transformation
- Feature Encoding
- Feature Selection
- Balancing Imbalanced Dataset
- Model Fitting
- Model Selection(Various Classification Algorithms)
- Hyperparameter tuning
- Conclusion

Problem Statement

- Our client is an Insurance company that has provided Health Insurance to its customers now they need your help in building a model to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company.

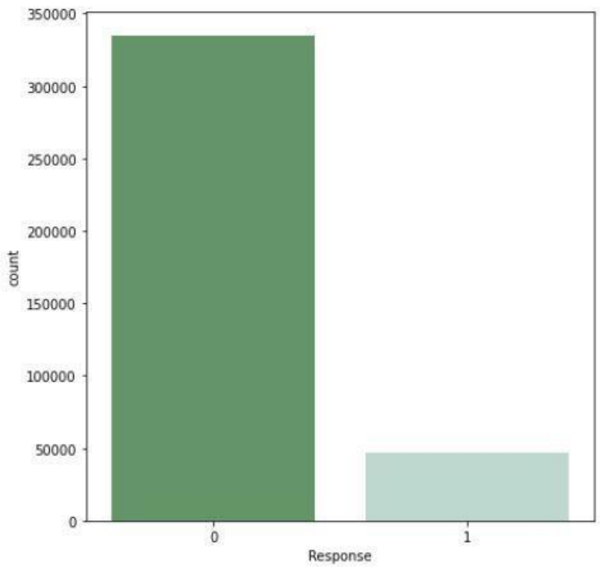
- Data shape(381109, 12)

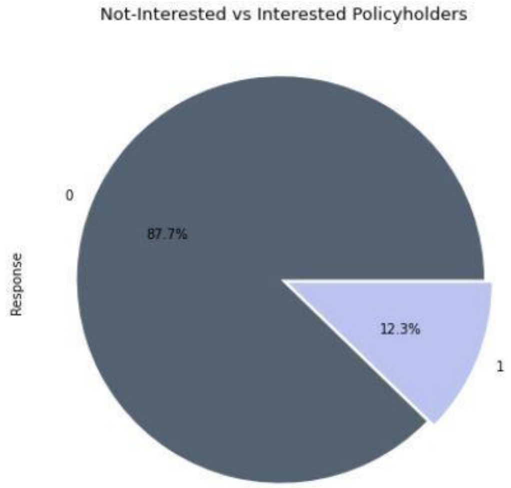
Data Summary

- id :Unique ID for the 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
- PolicySalesChannel :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

EDA

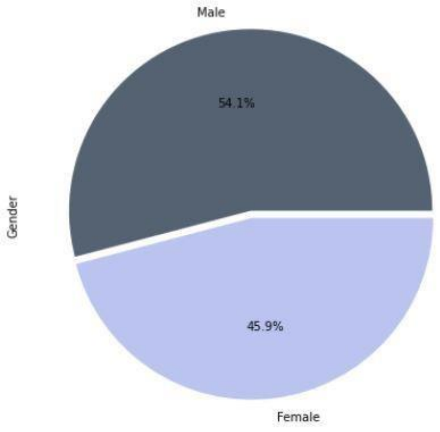
- Response (Dependent Variable)
- Response :1:Customer is interested, 0 :Customer is not interested

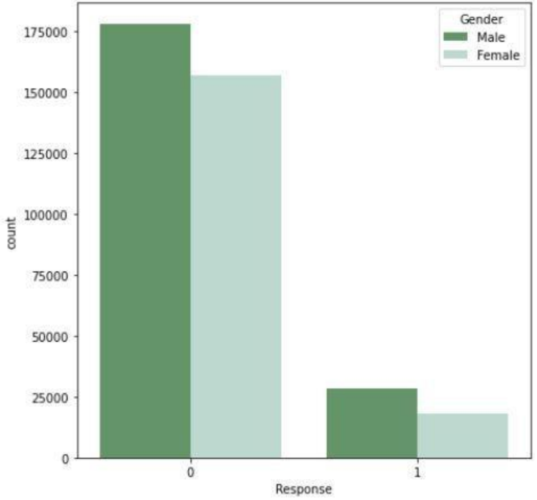




EDA

- Gender of the customer Vs. Response
- From below plots we can see number of men is bit more than women, so we have a little gender-gap here.
- Males seems to have more interest in vehicle insurance than women so we have to target woman more to increase conversion rate of women for vehicle insurance.

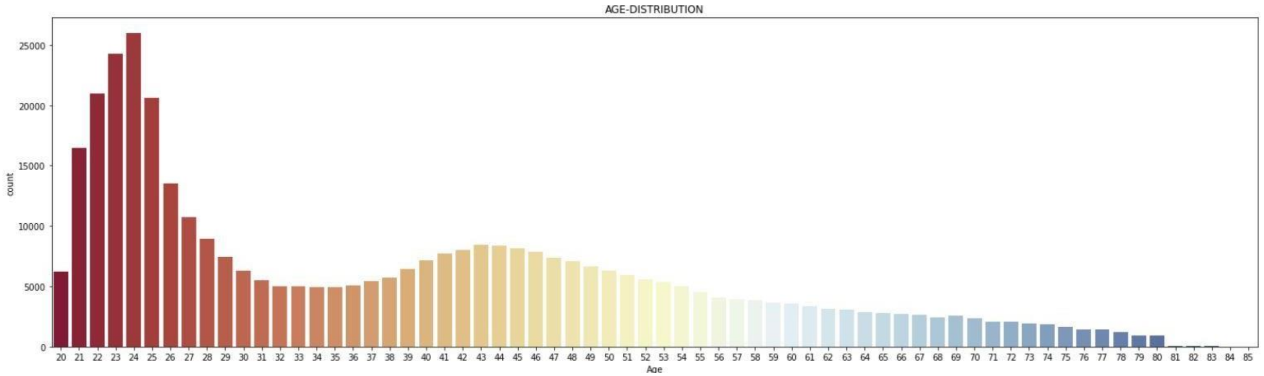




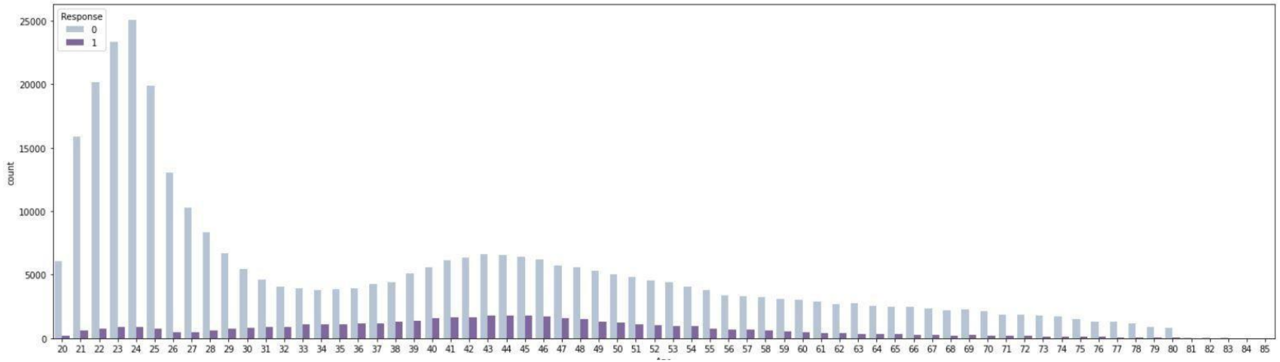
EDA

Here we can see customers in 30s-60s are most interested in vehicle insurance, which is quite natural as matured generation are aware of insurance and its benefit.

(Age)



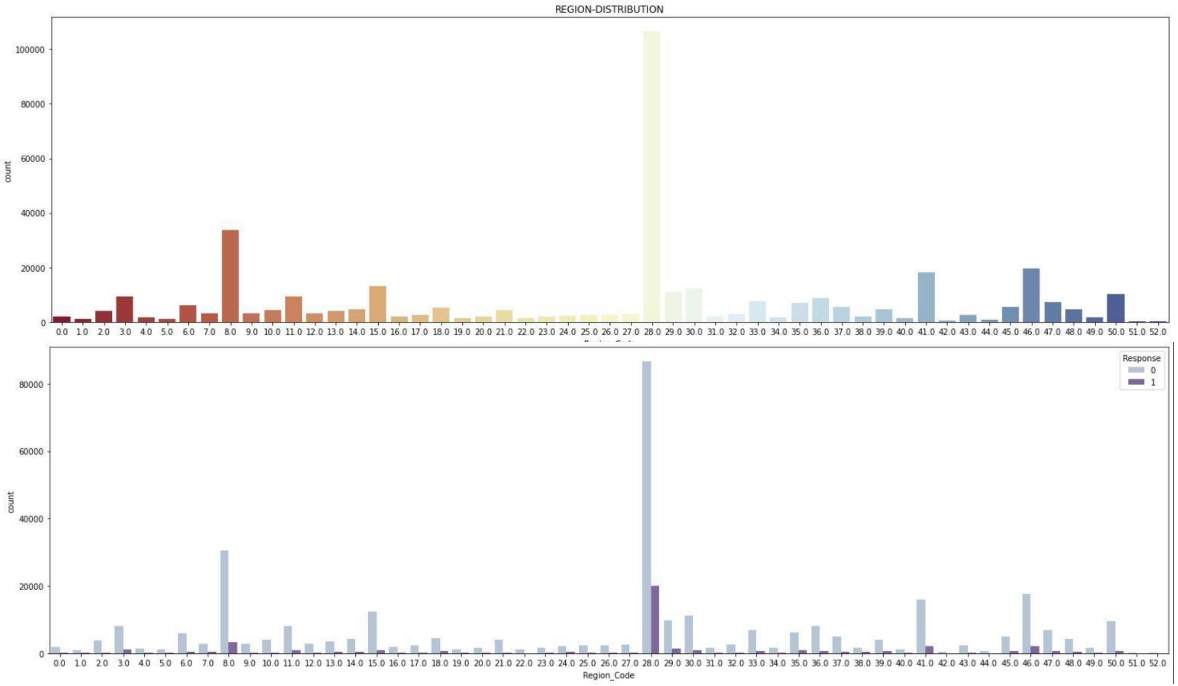
(Age Vs. Response)



EDA

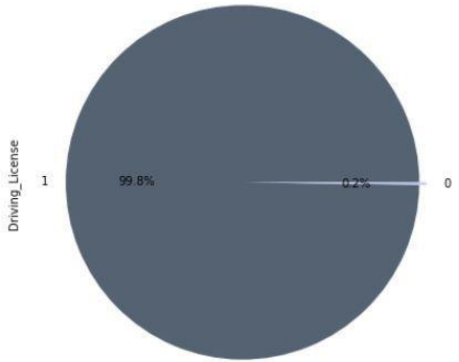
(Region_Code)

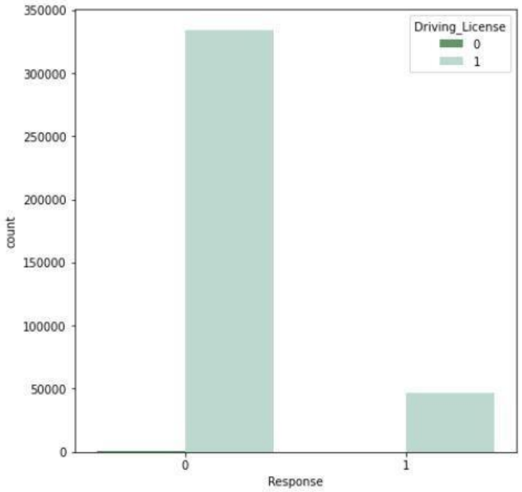
Region 28 has highest number of customer interest in vehicle insurance.



EDA

(Driving_License)



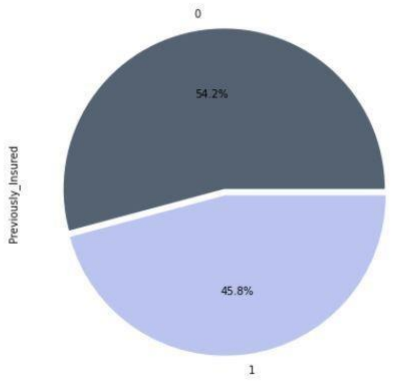


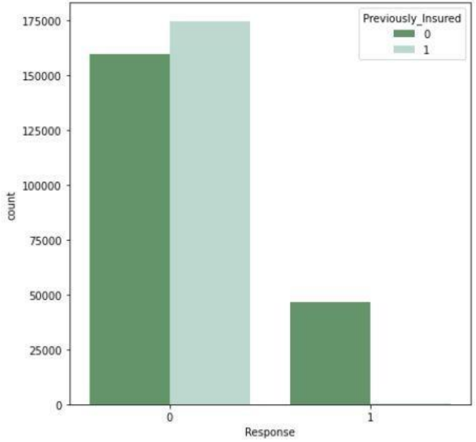
Customer without driving license is just 0.2% of all the customer, we can conclude that almost every customer has driving license.

EDA

(Previously_insured)

It seems that customer those who already had vehicle insurance tends to have less interest in having another vehicle insurance.





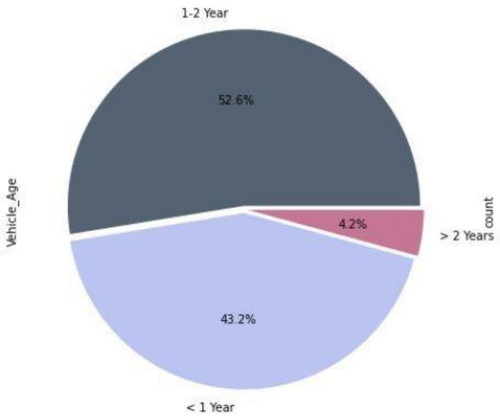
1:Customer already has Vehicle Insurance,

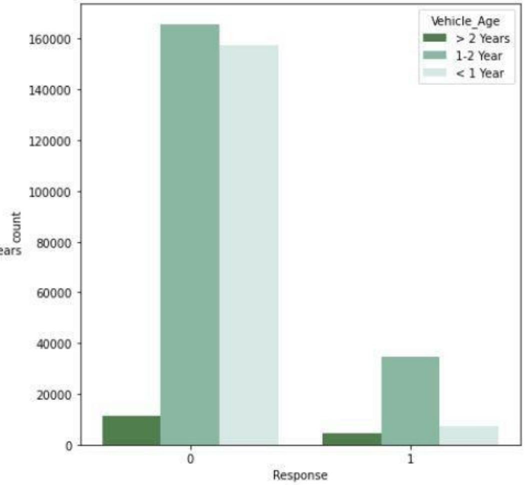
0 :Customer doesn't have Vehicle Insurance

EDA

(Vehicle_Age)

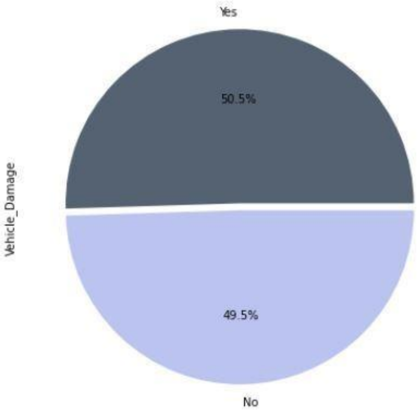
- From Vehicle_Age Vs. Response graph, we can say that if the vehicle age is in between 1to 2 year they tend to have more interest in vehicle insurance than others.

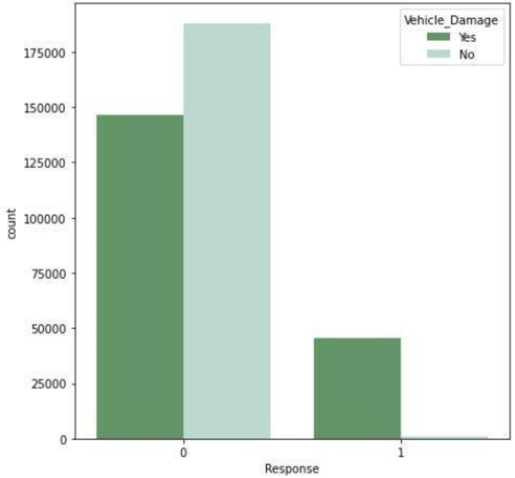




EDA

(Vehicle_Demage)

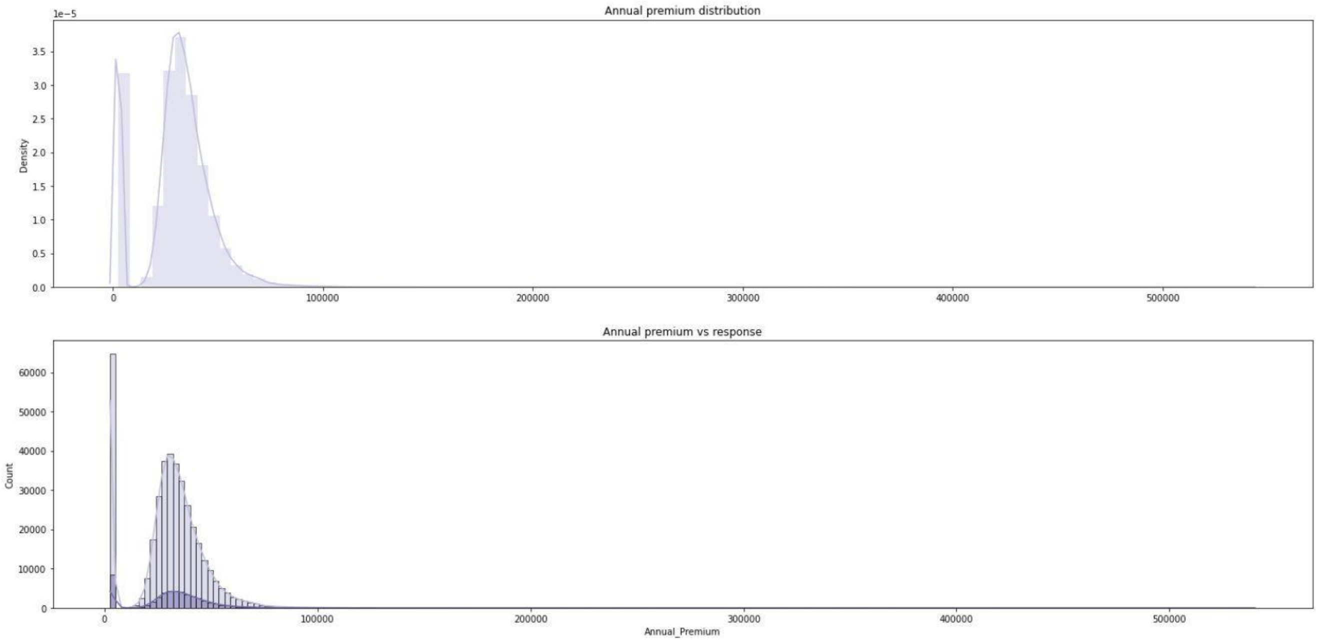




We can see from below graph that customer interested in vehicle insurance are mostly those who had their vehicle damaged in past.

EDA

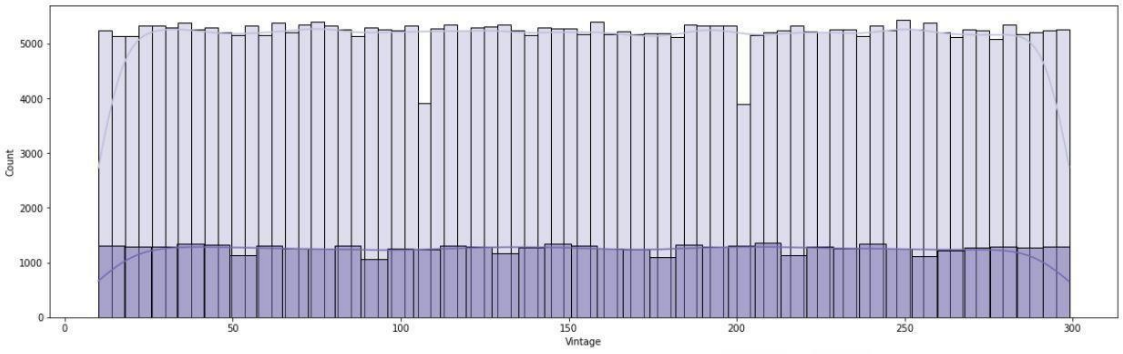
(Annual_premium)

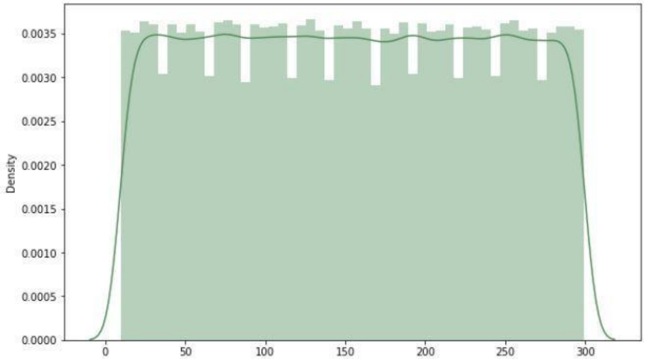


- The amount customer needs to pay as premium in the year
- From below graph we can see that it is right skewed.

EDA

(Vintage)



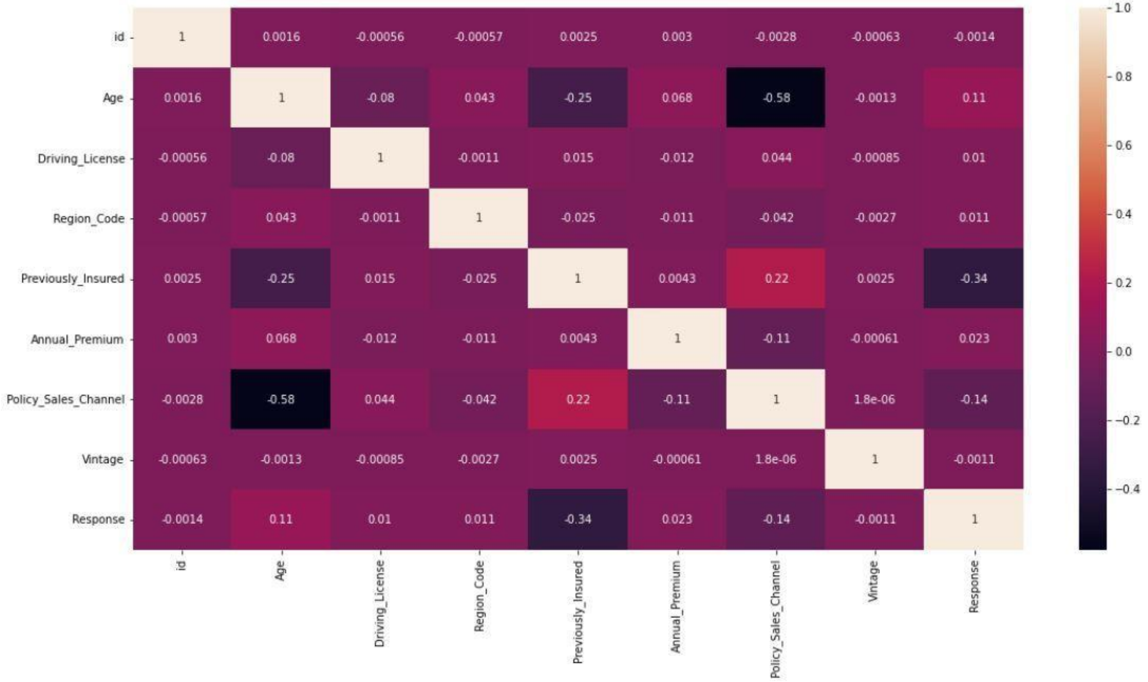


- Number of Days, Customer has been associated with the company
- There seems to be no such relation between vintage and response.

EDA

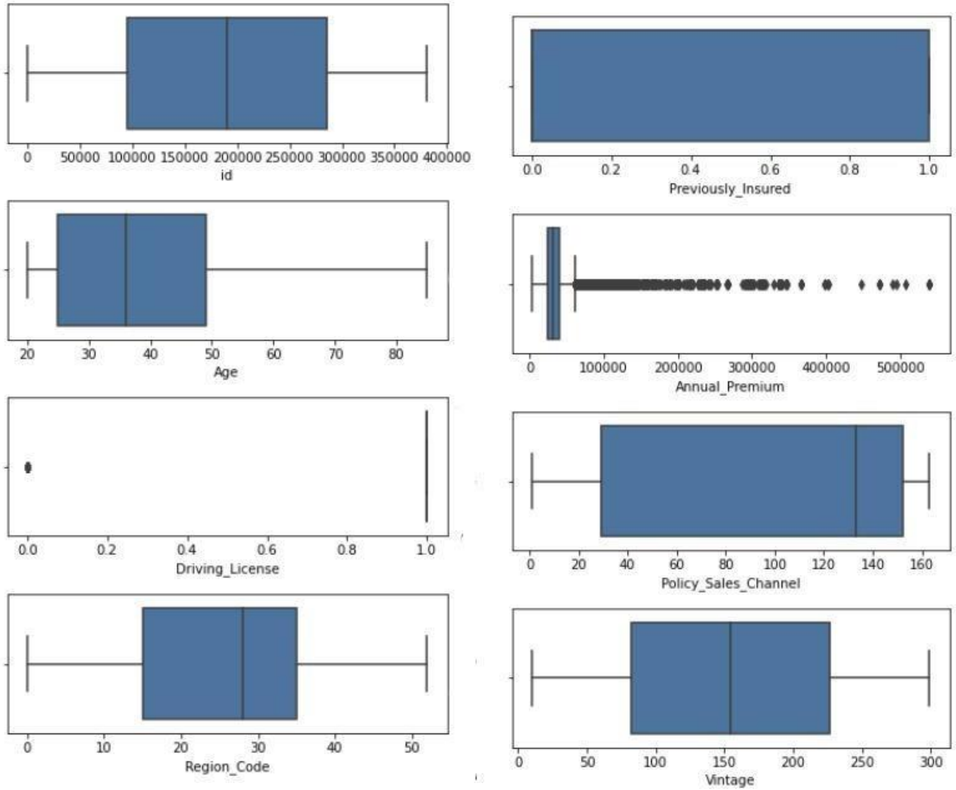
Multivariate analysis

To find correlation between features



Checking Outliers

Here we check for outliers in each Feature

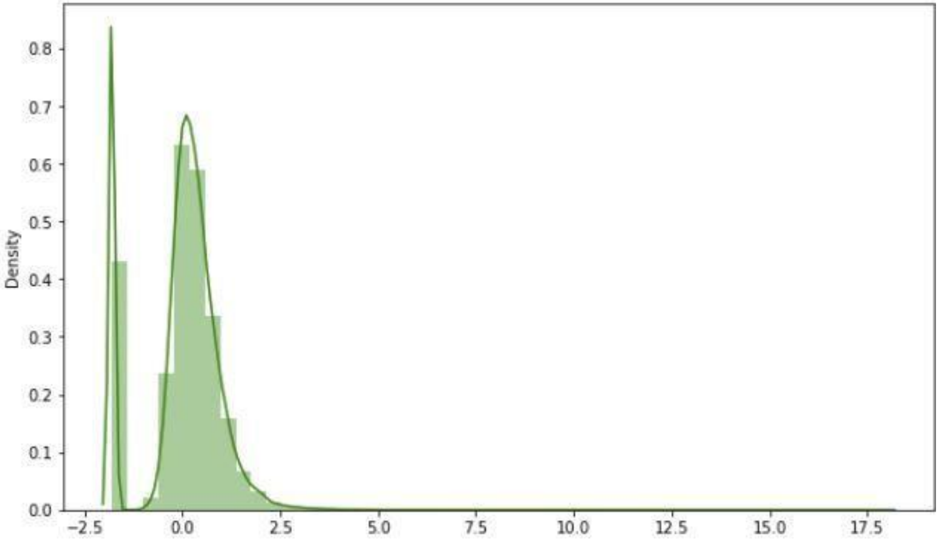


Feature Transformation

Since we have outliers in annual_premium and the data is also right skewed, we apply power Transform to Annual_premium.

```
features = Health_df[['Annual_Premium']]
```

```
#instatiate
from sklearn.preprocessing import PowerTransformer

pt = PowerTransformer(method='yeo-johnson', standardize=True,)

#Fitting data to the powertransformer
skl_yeojohnson = pt.fit(features)

#Lets get the Lambdas that were found
print (skl_yeojohnson.lambdas_)

calc_lambdas = skl_yeojohnson.lambdas_

#Transform the data
skl_yeojohnson = pt.transform(features)

#Pass the transformed data into a new dataframe
Health_df['Annual_Premium'] = pd.DataFrame(data=skl_yeojohnson, columns=['Annual_Premium'])

Health_df['Annual_Premium']
```



Encoding

```
#encoding numerical columns to categorcial
#we will use both label as well as one hot encoding for transformation of data type
le = LabelEncoder()
ohe = OneHotEncoder()

#mapping
Health_df["Vehicle_Age"]=Health_df["Vehicle_Age"].map({"> 2 Years":2,"1-2 Year":1,"< 1 Year":0})


#categorical to numerical
Health_df['Gender'] = ohe.fit_transform(Health_df[["Gender"]]).toarray()
Health_df['Vehicle_Age'] = le.fit_transform(Health_df[['Vehicle_Age']])
Health_df['Vehicle_Damage'] = le.fit_transform(Health_df[['Vehicle_Damage']])
```

```
Health_df.head()
```

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	0.0	44	1	28.0	0	2	1	40454.0	26.0	217	1
1	0.0	76	1	3.0	0	1	0	33536.0	26.0	183	0
2	0.0	47	1	28.0	0	2	1	38294.0	26.0	27	1
3	0.0	21	1	11.0	1	0	0	28619.0	152.0	203	0
4	1.0	29	1	41.0	1	0	0	27496.0	152.0	39	0

Feature Selection

- For feature selection we use VIF variance inflation factor. Vif for Driving_license is highest i.e. 43, which is beyond tolerance level.
- And around 99.8% of total customer have driving license, which makes almost everyone has it.
- Thus we drop Driving_License feature from our dataset .

Initial VIF

	variables	VIF
0	Gender	1.900505
1	Age	18.411094
2	Driving_License	43.300484
3	Region_Code	4.956856
4	Previously_Insured	5.726796
5	Vehicle_Age	6.033224
6	Vehicle_Damage	6.354748
7	Annual_Premium	4.182503
8	Policy_Sales_Channel	8.082449
9	Vintage	4.373894

VIF after dropping Driving_license

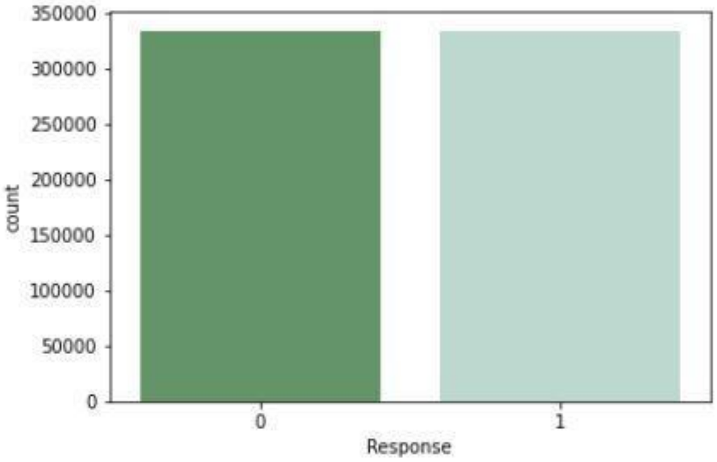
	variables	VIF
0	Gender	1.847842
1	Age	13.471196
2	Region_Code	4.558850
3	Previously_Insured	4.620188
4	Vehicle_Age	6.023186
5	Vehicle_Damage	5.186692
6	Annual_Premium	1.013756
7	Policy_Sales_Channel	5.217964
8	Vintage	4.066432

Balancing Imbalanced Dataset

- We have highly imbalanced data in our dataset i.e. in our response feature we have 2 variable, 1: Customer is interested, 0 :Customer is not interested.
- If we train a binary classification model without fixing this problem, the model will be completely biased towards majority .
- To deal with this problem we will balance our dataset using SMOTE .

```
X=Health_df.drop(['Response'],axis=1) #contain all  independent variable
y=Health_df['Response']           #dependent variable
```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE()

# fit predictor and target variable
X, y = smote.fit_resample(X, y)
```



Train –Test Split

- After applying Smote to balance the data, we now fit dataset to train- test split for training and testing purpose(model evaluation).

```
X_train,X_test,y_train,y_test = train_test_split(X, y,test_size=0.20,random_state = 10)
```

Model Selection

Models we will be using here are:

- Logistic Regression
- Decision Tree
- Random Forest
- XGBClassifier

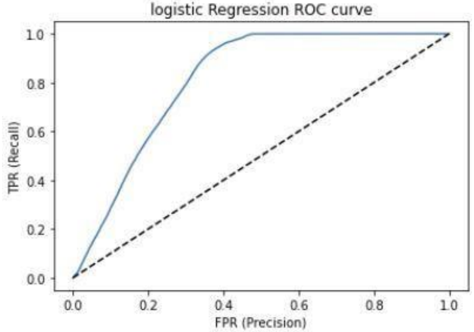
Logistic Regression

Logistic Regression

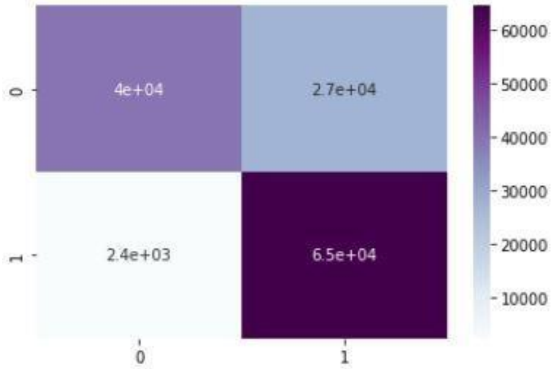
```
[ ] LOG_RE=LogisticRegression()
    LOG_RE=LOG_RE.fit(X_train,y_train)
    LOG_RE_pred=LOG_RE.predict(X_test)

    accu_logreg = accuracy_score(y_test,LOG_RE_pred)
    recall_logreg = recall_score(y_test,LOG_RE_pred)
    prec_logreg = precision_score(y_test,LOG_RE_pred)
    f1score_logreg=f1_score(y_test,LOG_RE_pred)

    #print accuracy and Auc values of model
    print("Accuracy : ", accuracy_score(y_test,LOG_RE_pred)*100)
    print(classification_report(y_test,LOG_RE_pred))

    matrix = confusion_matrix(y_test,LOG_RE_pred)
    print('Confusion matrix : \n',matrix)
```

```
Accuracy :  77.72278708133972
              precision    recall  f1-score   support

           0       0.94      0.59      0.73     66882
           1       0.70      0.96      0.81     66878

    accuracy                           0.78    133760
   macro avg       0.82      0.78      0.77    133760
weighted avg       0.82      0.78      0.77    133760

Confusion matrix :
 [[39538 27344]
 [ 2454 64424]]
```

Roc-Curve



Confusion Matrix

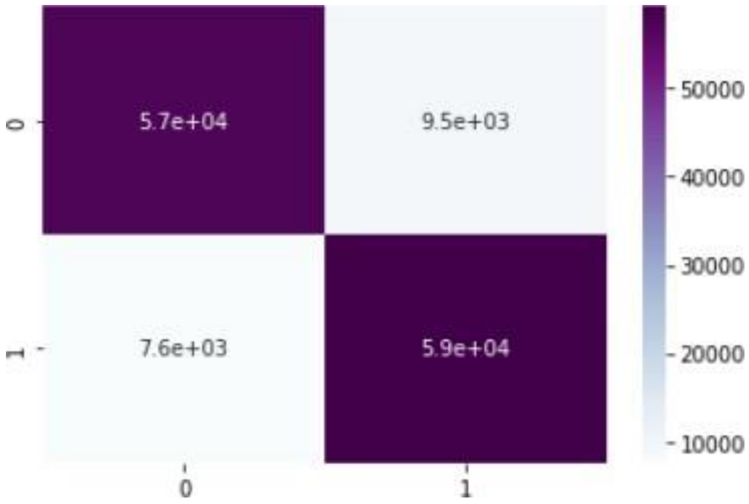


Decision Tree

```python
# Creating instance for our model, fiting and predicitng
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X_train, y_train)
dtree_pred = dtree.predict(X_test)
dtree_probability =dtree.predict_proba(X_test)[:,1]

# evaluating the model on the following metrics.
accu_dtree = accuracy_score(y_test,dtree_pred)
recall_dtree = recall_score(y_test,dtree_pred)
prec_dtree = precision_score(y_test,dtree_pred)
f1score_dtree=f1_score(y_test,dtree_pred)


#print accuracy ,classification report and confusion matrix  values of model.

print(accuracy_score(y_test, dtree_pred)*100)
print(classification_report(y_test, dtree_pred))

print('Confusion matrix : \n',confusion_matrix(y_test,dtree_pred, labels=[1,0]))
```

```
87.19348086124403
              precision    recall  f1-score   support

           0       0.88      0.86      0.87     66882
           1       0.86      0.89      0.87     66878

    accuracy                           0.87    133760
   macro avg       0.87      0.87      0.87    133760
weighted avg       0.87      0.87      0.87    133760
```

Confusion Matrix

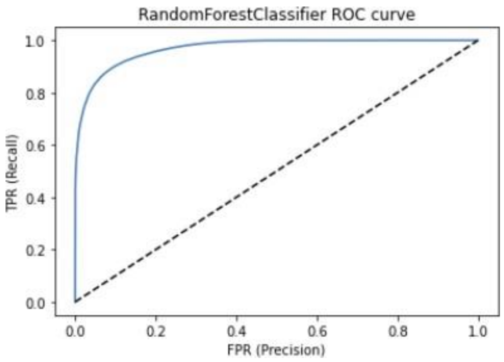


Random Forest classifier

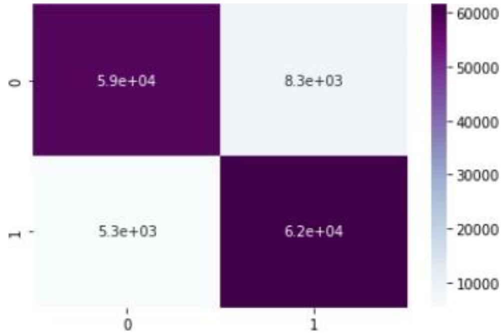
```
# Creating instance for our model, fiting and predicitng
rf_tree = RandomForestClassifier()
rf_tree.fit(X_train, y_train)
rf_tree_pred = rf_tree.predict(X_test)
rf_tree_probability = rf_tree.predict_proba(X_test)[:,1]

# evaluating the model on the following metrics.
accu_rf= accuracy_score(y_test,rf_tree_pred)
recall_rf = recall_score(y_test,rf_tree_pred)
prec_rf= precision_score(y_test,rf_tree_pred)
f1score_rf=f1_score(y_test,rf_tree_pred)

#print accuracy ,classification report and confusion matrix  values of model.

print(accuracy_score(y_test, rf_tree_pred)*100)
print(classification_report(y_test, rf_tree_pred))
print("Confusion matrix\n", confusion_matrix(y_test, rf_tree_pred))
```

```
89.84823564593302
              precision    recall  f1-score   support

           0       0.92      0.88      0.90     66882
           1       0.88      0.92      0.90     66878

    accuracy                           0.90    133760
   macro avg       0.90      0.90      0.90    133760
weighted avg       0.90      0.90      0.90    133760
```

Roc-Curve



Confusion Matrix

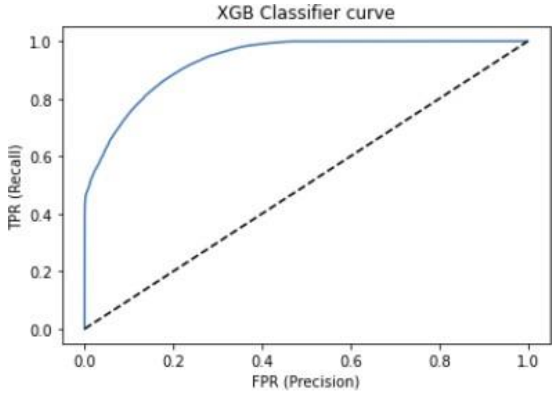


XGBClassifier

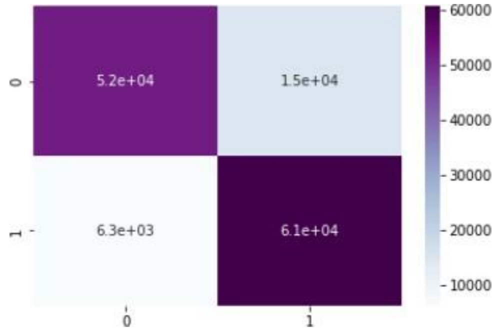
```
# Importing of XGBClassifier
import xgboost as xgb
xgb_model=xgb.XGBClassifier()
xgb_model.fit(X_train,y_train)
xgb_pred = xgb_model.predict(X_test)
xgb_model_probability = xgb_model.predict_proba(X_test)[:,1]

# evaluating the model on the following metrics.
accu_xgb = accuracy_score(y_test,xgb_pred)
recall_xgb = recall_score(y_test,xgb_pred)
prec_xgb = precision_score(y_test,xgb_pred)
f1score_xgb=f1_score(y_test,xgb_pred)

#print accuracy ,classification report and confusion matrix  values of model.

print(accuracy_score(y_test, xgb_pred)*100)
print(classification_report(y_test, xgb_pred))
print("Confusion matrix\n", confusion_matrix(y_test, xgb_pred))
```

```
84.15146531100478
              precision    recall  f1-score   support

           0       0.89      0.78      0.83     66882
           1       0.80      0.91      0.85     66878

    accuracy                           0.84    133760
   macro avg       0.85      0.84      0.84    133760
weighted avg       0.85      0.84      0.84    133760
```

Roc -Curve



Confusion Matrix



Hyperparameter Tuning (RandomForestClassifier)

```
# Importing Bayes search CV for hyperparameter tuning
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
```

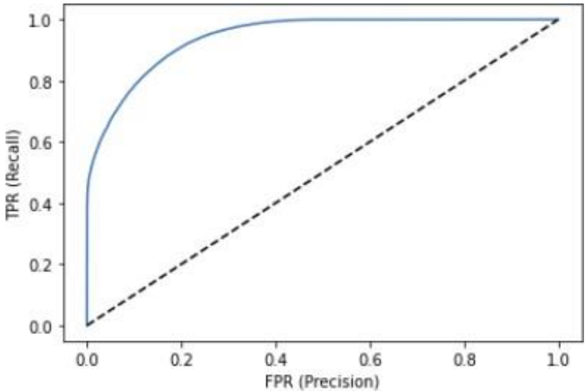
```
# #Hyperparameter Tuning on RandomForestClassifier

rf = RandomForestClassifier(random_state=40)
#Cross validation and hyperparameter tuning
rf_bayes = BayesSearchCV(estimator= rf,
                         search_spaces = {
                          'max_depth': Integer(2,100),
                          'min_samples_leaf': Integer(1,100),
                          'min_samples_split': Integer(2,100),
                          'n_estimators': Integer(1,140),
                          'max_features': ["auto", "sqrt", "log2"]
                         },
                       cv = 5, verbose=2, scoring='accuracy',n_iter=10)

rf_bayes.fit(X_train,y_train)
```

```
rf_bayes.best_estimator_
```

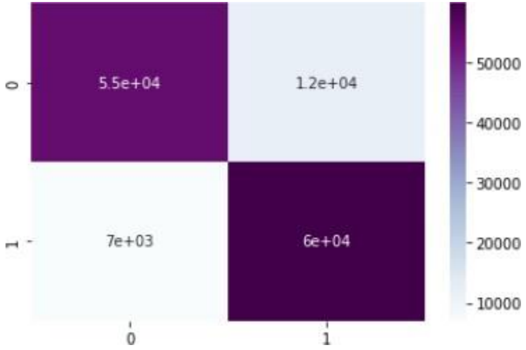
```
RandomForestClassifier(max_depth=26, max_features='sqrt', min_samples_leaf=27,
                       min_samples_split=94, n_estimators=15, random_state=40)
```

```
              precision    recall  f1-score   support

           0       0.89      0.82      0.85     66882
           1       0.83      0.90      0.86     66878

    accuracy                           0.86    133760
   macro avg       0.86      0.86      0.86    133760
weighted avg       0.86      0.86      0.86    133760
```

Roc-Curve



Confusion Matrix



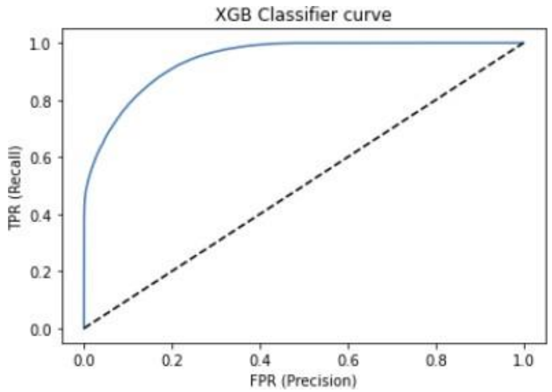
Hyperparameter Tuning (XGBoostClassifier)

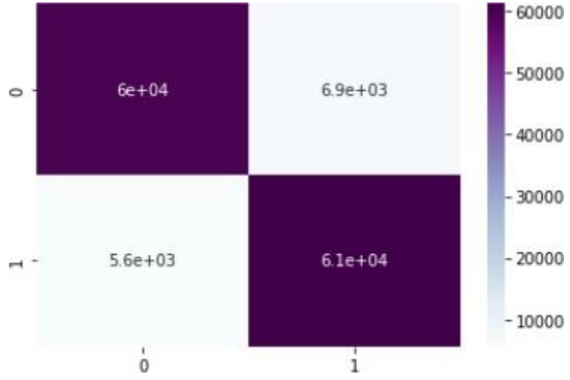
```
#hyper parameter tuning
xgb_model=xgb.XGBClassifier()
#xgb = XGBoostClassifier()
#Cross validation and hyperparameter tuning
xgb_bayes = BayesSearchCV(estimator= xgb_model,
                          search_spaces = {
                           'max_depth': Integer(2,100),
                           'min_samples_leaf': Integer(1,100),
                           'min_samples_split': Integer(2,100),
                           'n_estimators': Integer(1,140),
                           'max_features': ["auto", "sqrt", "log2"]
                          },
                        cv = 5, verbose=2, scoring='accuracy',n_iter=4)

xgb_bayes.fit(X_train,y_train)
```

```
xgb_bayes.best_estimator_
```

```
XGBClassifier(max_depth=99, max_features='auto', min_samples_leaf=55,
              min_samples_split=68, n_estimators=110)
```

	precision	recall	f1-score	support
0	0.91	0.90	0.91	66882
1	0.90	0.92	0.91	66878
accuracy			0.91	133760
macro avg	0.91	0.91	0.91	133760
weighted avg	0.91	0.91	0.91	133760





Comparing The Model After Hyperparameter Tuning

```
## Comparing the performance of the models
ind=['Randomforest','XGBClassifier']
df={"Accuracy":[accu_rf_hp,accu_xgb_hp],"Recall":[recall_rf_hp,recall_xgb_hp],"Precision":[prec_rf_hp,prec_xgb_hp],'f1_score':[f1score_rf_hp,f1score_xgb_hp]}
result=pd.DataFrame(data=df,index=ind)
result
```

	Accuracy	Recall	Precision	f1_score
Randomforest	0.855315	0.895048	0.829152	0.860841
XGBClassifier	0.906026	0.915862	0.898188	0.906939

Conclusion

- The given dataset is an imbalance problem as the Response variable with the value 1is significantly lower than the value zero
- The male customers own slightly more vehicles and they are more tend to buy insurance in comparison to their female counterparts.
- The male customers own slightly more vehicles and they are more tend to buy insurance in comparison to their female counterparts.
- Customers of aged between 30 to 60 are more likely to buy insurance. Whereas Youngsters under 30 are not intrigued by vehicle insurance. Reasons could be the absence of involvement, less awareness about insurance and they may not have costly vehicles yet.
- the customers who have driving licences will option for insurance instead of those who don’t have it
- Consumers with 1-2-year-old vehicles are more interested in buying insurance. as compared to Consumers with less than 1-year-old Vehicles
- Customers with Vehicle_Damage are likely to buy insurance as they have experienced the expenditure in repairing vehicles The variable such as Age, Previously_insured, Annual_premium is more affecting the target variable.
- We used different type of algorithms to train our model like, Logistic Regression, Random Forest model, Decision tree and XGB Classifier. And Also we tuned the parameters of XGB Classifier and Random Forest model Comparing the model on the basis of precision, recall, accuracy ,F1 score we can see that the XGBClassifier model performs better. Even comparing ROC curve XGB Classifier performed better because curves closer to the top-left corner indicate better performance.