Travel insurance Project

https://drive.google.com/drive/folders/1MNre59Ma59HLxKIhUgSa7adhNDjJ9T1V?usp=sharing (https://drive.google.com/drive/folders/1MNre59Ma59HLxKIhUgSa7adhNDjJ9T1V?usp=sharing)

In this project we create a model that can predict for whether a customer can claim for Travel Insurance or not.

case study :

Insurance companies take risks over customers. Risk management is a very important aspect of the insurance industry. Insurers consider every quantifiable factor to develop profiles of high and low insurance risks. Insurers collect vast amounts of information about policyholders and analyse the data. As a Data scientist in an insurance company, you need to analyse the available data and predict whether to approve the insurance or not.

About Dataset

Feature Description

ID Unique identifier

Agency Agency name

Agency Type Type of travel insurance agency

Distribution Channel Online/Offline distribution channel

Product Name Travel insurance product name

Duration Duration of travel

Destination Destination of travel

Net sales Net sales of travel insurance policies

Commision The commission received by travel insurance agency

Gender Traveller's gender

Age Traveller's Age

In [ ]:

```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Cross-Validation
from sklearn.model_selection import train_test_split

# LabelEncoding
from sklearn.preprocessing import LabelEncoder

# Evaluation
from sklearn.metrics import classification_report

# Scaling
from sklearn.preprocessing import MinMaxScaler

# Ridge, Lasso
from sklearn.linear_model import Ridge, Lasso

# Logistic Regression
from sklearn.linear_model import LogisticRegression

# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# GridSearchCV
from sklearn.model_selection import GridSearchCV

# Boosting, RandomForest
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomForestCl
from xgboost import XGBClassifier

# Ensemble
from sklearn.ensemble import VotingClassifier, BaggingClassifier

# Feature Selection
from sklearn.feature_selection import chi2, SelectKBest

# SVM
from sklearn.svm import LinearSVC, SVC

# Skewness
from scipy.stats import skew

# Over and Under Sampling
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

# Pickle
import pickle

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
# Reading the data and viewing a small part of it to get some understanding of the data.

df = pd.read_csv("E:\MLcsv\data.csv")
print(df.shape)
df.head(8)
```

In [ ]:

```
# We will get a list of the number of unique values for each column

df.nunique()
```

In [ ]:

```
# We will check for null values and the Dtype of each feature.

df.info()
```

In [ ]:

```
((df.isnull().sum())*100)/len(df)
```

hence 71% of the Gender column have null values.We will drop the column as there does not seem to be any other feature that could help us with filling in the missing data.

In [ ]:

```
df.drop("Gender", axis=1, inplace=True)
```

In [ ]:

```
# Having a look at all the unique values of each feature.

for cols in df:
    print("\n{:20} - {}" .format(cols.title(), df[cols].unique()))
```

In [ ]:

```
# Checking for correlation

df.corr()
```

Dropping the ID column. Each value is unique and does not seem to affect the data.

In [ ]:

```
df.drop("ID", axis=1, inplace=True)
```

In [ ]:

```
# Having a look at how many claims and non-claims are present in the dataset.
print(df["Claim"].value_counts(), "\n")
(df["Claim"].value_counts()*100)/len(df)
```

We can see that there is a huge imbalance between the claims and non-claims.

We will build a baseline model before we perform Over Sampline and Under Sampling.

In [ ]:

```
# Finding out how many customers have their age input as over 100yrs old

len(df[df["Age"] > 100])
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```
#creating a variable to calculate the mean of all Senior customers.

mean_senior = df["Age"][df["Age"] > 70].mean()
```

separation of the categorical and numerical data.

In [ ]:

```
df.nunique()
```

Apart from the target, "Claim", there are two more features that are bivariate - "Agency Type" and "Distribution Channel".
We could look to perform Hot Encoding on them.

We will separate the Categorical and Numerical features, and explore them further.

In [ ]:

```
cat = ["Agency", "Agency Type", "Distribution Channel", "Product Name", "Destination"]
num = ["Duration", "Net Sales", "Commision (in value)", "Age"]
```

In [ ]:

```
for cols in cat:
    if (cols == "Product Name") or (cols == "Destination"):
        plt.figure(figsize=(20,30))
        sns.countplot(data=df, hue=df["Claim"], y=cols)
    else:
        plt.figure(figsize=(12,12))
        sns.countplot(data=df, hue=df["Claim"], x=cols)
    plt.xticks(rotation=90)
    plt.show()
```

In [ ]:

```
for cols in num:
    plt.figure(figsize=(8,8))
    sns.boxplot(data=df, x="Claim", y=cols)
    plt.show()
```

We would need to manage only some of the outliers, and not all as it could lead to a lot of data loss. Apart from Age, another would be Duration. From the information below, we could replace all values in duration that are greater than 360, wtih 360.

Policy Duration: Cover trips from as short as 1 day to max of 360 days. Most of the Insurance Companies provides coverage for 180 days which can be extended for a further period of 180 days, provided there is no claim.

http://www.insurancepandit.com/travel/individual_travel_health_insurance.php (http://www.insurancepandit.com/travel/individual_travel_health_insurance.php)

In [ ]:

```
df.describe()
```

In [ ]:

```
for cols in num:
    skew_cols = skew(df[cols])
    print("{:<25} : {}" .format(cols, skew_cols))
    plt.figure(figsize=(8,8))
    sns.distplot(df[cols])
    plt.show()
```

In [ ]:

```
for cols in num:
    print("\n", cols)
    print(df[cols].value_counts().sort_index())
```

There is some skewness within the data. This will be handled later on.

In [ ]:

```
# The entries where the duration is -ve, we will drop those rows.

duration = df[df["Duration"] < 0].index
df.drop(duration, inplace=True)
```

In [ ]:

```
df[(df["Net Sales"] < 0) & (df["Claim"] == 0)]
```

In [ ]:

```
df[(df["Net Sales"] < 0) & (df["Claim"] == 1)]
```

LabelEncoding, One Hot Encoding, Frequency Encoding

In [ ]:

```
# Label Encoding

for cols in cat:
    le = LabelEncoder()
    df[cols] = le.fit_transform(df[cols])

df.head(8)
```

```
# Frequency Encoding

fe = df.groupby('Destination').size()/len(df)
df.loc[:,'Dest Freq'] = df['Destination'].map(fe)
df.drop(columns='Destination',axis=1,inplace=True)

fe_1 = df.groupby('Agency').size()/len(df)
df.loc[:,'Agency Freq'] = df['Agency'].map(fe_1)
df.drop(columns='Agency',axis=1,inplace=True)

fe_2 = df.groupby('Product Name').size()/len(df)
df.loc[:,'Product Name Freq'] = df['Product Name'].map(fe_2)
df.drop(columns='Product Name',axis=1,inplace=True)
```

```
# One-Hot Encoding

df = pd.get_dummies(df, columns=["Agency Type", "Distribution Channel"], drop_first=True)
df.head()
```

In [ ]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

Train, Test, Split

Function to train, test, and split

In [ ]:

```
def tts(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1

    return X_train, X_test, y_train, y_test
```

Fit and Predict

Function to fit and predict the model, and to display the report

In [ ]:

```
def model_sel(model, X_train, X_test, y_train, y_test):

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    return classification_report(y_test, y_pred)
```

All Models

Function where all the models will be defined and then passed to 'model_sel' for the model to be created.

In [ ]:

```
def models(X_train, y_train, X_test="None", y_test="None", sampled="No"):
    if sampled == "No":
        X_train, X_test, y_train, y_test = tts(X_train, y_train)
    else:
        pass

    lr = LogisticRegression()
    dtc = DecisionTreeClassifier()
    abc = AdaBoostClassifier(n_estimators=100)
    gbc = GradientBoostingClassifier(n_estimators=100)
    xbc = XGBClassifier(n_estimators=200, reg_alpha=1)
    rfc = RandomForestClassifier()
    lsvc = LinearSVC(random_state=1)
    svc = SVC(random_state=1)
    print("{} \n {}\n" .format("LOGISTIC REGRESSION", model_sel(lr, X_train, X_test, y_trai
    print("{} \n {}\n" .format("DECISION TREE", model_sel(dtc, X_train, X_test, y_train, y_
    print("{} \n {}\n" .format("ADABOOST", model_sel(abc, X_train, X_test, y_train, y_test)
    print("{} \n {}\n" .format("GRADIENT BOOST", model_sel(gbc, X_train, X_test, y_train, y
    print("{} \n {}\n" .format("XGBOOST", model_sel(xbc, X_train, X_test, y_train, y_test))
    print("{} \n {}\n" .format("RANDOM FOREST", model_sel(rfc, X_train, X_test, y_train, y_
    print("{} \n {}\n" .format("LINEAR SVM", model_sel(lsvc, X_train, X_test, y_train, y_te
    print("{} \n {}\n" .format("SVM", model_sel(svc, X_train, X_test, y_train, y_test)))

    return lr, abc, gbc, xbc, rfc, lsvc, svc
```

Manual Under Sampling

We will match the number of non-claims to claims. Below are the steps

**1. Get the count of undersampled and oversampled Claims.
**
**2. Create new variable that will randomly select the same number of oversampled Cl aims as there is undersampled.
**
**3. Concatenate the two into a numpy array.
**
**4. Create a new DataFrame taking the indexes from the concatenated array.
**
**5. Use this DataFrame to run the models.
**

In [ ]:

```
def sampling(df):
    min_claim = len(df[df["Claim"] == 1])
    min_claim_ind = df[df["Claim"] == 1].index

    maj_claim_ind = df[df["Claim"] == 0].index

    random_major = np.random.choice(maj_claim_ind, min_claim, replace=False)

    sample_ind = np.concatenate([min_claim_ind, random_major])

    under_sample = df.loc[sample_ind]

    # print(sns.countplot(data=under_sample, x="Claim"))

    X = under_sample.loc[:, df.columns!="Claim"]
    y = under_sample.loc[:, df.columns=="Claim"]

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X, y)
    return lr, abc, gbc, xbc, rfc, lsvc, svc, X, y
```

Over Sampling

The number of minority values will be made to equal the number of majority values.

In [ ]:

```
def over_sample():
    X = df.drop("Claim", axis=1)
    y = df["Claim"]
    print(Counter(y))
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
    oversample = RandomOverSampler(sampling_strategy='minority')
    X_over, y_over = oversample.fit_resample(X_train, y_train)
    print(Counter(y_over))

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X_over, y_over, X_test, y_test, "Yes")
    return lr, abc, gbc, xbc, rfc, lsvc, svc, X_over, y_over
```

Under Sampling

The number of majority values will be reduced down to equal the number of minority values.

In [ ]:

```
def under_sample():
    X = df.drop("Claim", axis=1)
    y = df["Claim"]
    print(Counter(y))
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
    undersample = RandomUnderSampler(sampling_strategy='majority')
    X_under, y_under = undersample.fit_resample(X_train, y_train)
    print(Counter(y_under))

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X_under, y_under, X_test, y_test, "Yes")
    return lr, abc, gbc, xbc, rfc, lsvc, svc, X_under, y_under
```

```
def samp_model_sel(model, X_train, y_train, X_test, y_test):

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    return classification_report(y_test, y_pred)
```

```
def samp_models(X, y, X_test, y_test):
    lr = LogisticRegression()
    dtc = DecisionTreeClassifier()
    abc = AdaBoostClassifier(n_estimators=100)
    gbc = GradientBoostingClassifier(n_estimators=100)
    xbc = XGBClassifier(n_estimators=200, reg_alpha=1)
    rfc = RandomForestClassifier()
    lsvc = LinearSVC(random_state=1)
    svc = SVC(random_state=1)
    print("{} \n {}\n" .format("LOGISTIC REGRESSION", samp_model_sel(lr, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("DECISION TREE", samp_model_sel(dtc, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("ADABOOST", samp_model_sel(abc, X, y, X_test, y_test)))
    print("{} \n {}\n" .format("GRADIENT BOOST", samp_model_sel(gbc, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("XGBOOST", samp_model_sel(xbc, X, y, X_test, y_test)))
    print("{} \n {}\n" .format("RANDOM FOREST", samp_model_sel(rfc, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("LINEAR SVM", samp_model_sel(lsvc, X, y, X_test, y_test)))
    print("{} \n {}\n" .format("SVM", samp_model_sel(svc, X, y, X_test, y_test)))

    return lr, abc, gbc, xbc, rfc, lsvc, svc
```

GridSearchCV

By passing the model along with parameters that it can carry, this function will iterate using the model parameters, and deliver the best model.

In [ ]:

```
def gridsearch(model, paramater, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1

    gscv = GridSearchCV(estimator=model, param_grid=parameter)
    gscv.fit(X_train, y_train)
    y_pred = gscv.predict(X_test)
    print(classification_report(y_test, y_pred))
    print(gscv.best_estimator_)
    return gscv
```

First Baseline Models

We will build four models - No Sampling, Manual Under Sampling, Over Sampled, Under Sampled.

In [ ]:

```
# Without Sampling

lr, abc, gbc, xbc, rfc, lscv, svc = models(X, y)
```

In [ ]:

```
# With manual Under Sampling

lr_sample, abc_sample, gbc_sample, xbc_sample, rfc_sample, lsvc_sample, svc_sample, X, y =
```

In [ ]:

```
# Over Sampled

lr_over, abc, gbc_over, xbc_over, rfc_over, lsvc_over, svc_over, X, y = over_sample()
```

In [ ]:

```
# Under Sampled

lr_under, abc_under, gbc_under, xbc_under, rfc_under, lsvc_under, svc_under, X, y = under_s
```

The scores are all zero for the base model without Sampling.
For all the sampling models, the scores increased drastically. Over Sampled models produced the best results.
Going forward, we will not run the models where no sampling is done.

Outliers

Those over 100yrs will be replaced by the mean of Senior aged customers, and where the Duration is more that 360 will be replaced by 360.

In [ ]:

```
df["Age"][df["Age"] > 60] = mean_senior
```

In [ ]:

```
df["Duration"][df["Duration"] > 360] = 360
```

In [ ]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [ ]:

```
# lr_out, abc_out, gbc_out, xbc_out, rfc_out, lsvc_out, svc_out = models(X, y)
```

In [ ]:

```
# Manual Under Sampling and Outliers

lr_out_sample, abc_out_sample, gbc_out_sample, xbc_out_sample, rfc_out_sample, lsvc_out_sam
```

In [ ]:

```
# Over Sampling and Outliers

lr_out_over, abc_out_over, gbc_out_over, xbc_out_over, rfc_out_over, lsvc_out_over, svc_out
```

In [ ]:

```
# Under Sampling and Outliers

lr_out_under, abc_out_under, gbc_out_under, xbc_out_under, rfc_out_under, lsvc_out_under, s
```

Skewness

In [ ]:

```
print("{:<15} : {}" .format("Duration", skew(df["Duration"])))
print("{:<15} : {}" .format("Commision (in value)", skew(df["Commision (in value)"])))
print("{:<15} : {}" .format("Age", skew(df["Age"])))
```

In [ ]:

```
df["Duration"] = np.sqrt(df["Duration"])
df["Commision (in value)"] = np.sqrt(df["Commision (in value)"])
df["Age"] = np.sqrt(df["Age"])
```

In [ ]:

```
print("{:<15} : {}" .format("Duration", skew(df["Duration"])))
print("{:<15} : {}" .format("Commision (in value)", skew(df["Commision (in value)"])))
print("{:<15} : {}" .format("Age", skew(df["Age"])))
```

In [ ]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [ ]:

```
# lr_skew, abc_skew, gbc_skew, xbc_skew, rfc_skew, lsvc_skew, svc_skew = models(X, y)
```

In [ ]:

```
# Manual Under Sampling and Skewing

lr_skew_sample, abc_skew_sample, gbc_skew_sample, xbc_skew_sample, rfc_skew_sample, lsvc_sk
```

In [ ]:

```
# Over Sampling and Skewing

lr_skew_over, abc_skew_over, gbc_skew_over, xbc_skew_over, rfc_skew_over, lsvc_skew_over, s
```

In [ ]:

```
# Under Sampling and Skewing

lr_skew_under, abc_skew_under, gbc_skew_under, xbc_skew_under, rfc_skew_under, lsvc_skew_un
```

Performing Chi-squared test

In [ ]:

```
len(df.columns)
```

In [ ]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [ ]:

```
X_cols = []

for col in X:
    X_cols.append(col)
```

In [ ]:

```
def model_chi(model, X):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1

    chi_test = SelectKBest(score_func=chi2, k=6)

    X_train_chi = chi_test.fit_transform(X_train, y_train)
    X_test_chi = chi_test.transform(X_test)

    model.fit(X_train_chi, y_train)
    y_pred = model.predict(X_test_chi)

    print(classification_report(y_test, y_pred))

    num = 0

    for each in chi_test.scores_:
        print("{:2} {:20} - {}" .format(num, X_cols[num], each))
        num += 1
```

In [ ]:

```
def model_new(X):
    lr = LogisticRegression()
    dtc = DecisionTreeClassifier(criterion="entropy")
    abc = AdaBoostClassifier(n_estimators=100)
    gbc = GradientBoostingClassifier(n_estimators=100)
    xbc = XGBClassifier(n_estimators=200, reg_alpha=1)
    rfc = RandomForestClassifier()
    print("{} \n {}\n" .format("LOGISTIC REGRESSION", model_chi(lr,X)))
    print("{} \n {}\n" .format("DECISION TREE", model_sel(dtc,X)))
    print("{} \n {}\n" .format("ADABOOST", model_chi(abc,X)))
    print("{} \n {}\n" .format("GRADIENT BOOST", model_chi(gbc,X)))
    print("{} \n {}\n" .format("XGBOOST", model_chi(xbc,X)))
    print("{} \n {}\n" .format("RANDOM FOREST", model_chi(rfc,X)))

    return lr, abc, gbc, xbc, rfc
```

lr_chi, abc_chi, gbc_chi, xbc_chi, rfc_chi = model_new(X)

It is clear that RandomForest has the best score.

In [ ]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [ ]:

```
X_cols = []

for col in X:
    X_cols.append(col)
```

In [ ]:

```
# lr_chi, abc_chi, gbc_chi, xbc_chi, rfc_chi = model_new(X)
```

Scaling

In [ ]:

```
df_old = df.copy(deep=True)
```

In [ ]:

```
mm = MinMaxScaler()

X = df.drop("Claim", axis=1)
cols = X.columns.to_list()
df[cols] = mm.fit_transform(df[cols])
```

In [ ]:

```
df.head()
```

In [ ]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [ ]:

```
# lr_scale, abc_scale, gbc_scale, xbc_scale, rfc_scale, lsvc_scale, svc_scale = models(X, y
```

In [ ]:

```
# Manual Under Sampling and Scalling

lr_scale_sample, abc_scale_sample, gbc_scale_sample, xbc_scale_sample, rfc_scale_sample, ls
```

In [ ]:

```
# Over Sampling and Scalling

lr_scale_over, abc_scale_over, gbc_scale_over, xbc_scale_over, rfc_scale_over, lsvc_scale_o
```

In [ ]:

```
# Under Sampling and Scalling

lr_scale_under, abc_scale_under, gbc_scale_under, xbc_scale_under, rfc_scale_under, lsvc_sc
```

Saving best model in a file through Pickle

In [ ]:

```
file = open("TravelInsurance.ser", "wb")
pickle.dump(rfc_under, file)

file.close()
```

Verdict

Version 1) Following are the process involved -

a) Read and analyzed dataset.
b) Removed 'Gender' as it had 71% null values.
c) Performed Label Encoding.
d) Created defintions for fitting and predicting models.
e) Skewness, Outliers, Scaling, Chi-Squared Test, Boosting.

Result -The scores achieved for each and every model in this version was zero (as you can see below). A different approach was required.

LOGISTIC REGRESSION

	precision	recall	f1-score	support
0	0.99	1.00	0.99	14952
1	0.00	0.00	0.00	213
accuracy			0.99	15165
macro avg	0.49	0.50	0.50	15165
weighted avg	0.97	0.99	0.98	15165

DECISION TREE

	precision	recall	f1-score	support
0	0.99	0.98	0.99	14952
1	0.05	0.06	0.06	213
accuracy			0.97	15165
macro avg	0.52	0.52	0.52	15165
weighted avg	0.97	0.97	0.97	15165

ADABOOST

	precision	recall	f1-score	support
0	0.99	1.00	0.99	14952
1	0.00	0.00	0.00	213
accuracy			0.99	15165
macro avg	0.49	0.50	0.50	15165
weighted avg	0.97	0.99	0.98	15165

GRADIENT BOOST

	precision	recall	f1-score	support
0	0.99	1.00	0.99	14952
1	0.00	0.00	0.00	213
accuracy			0.99	15165
macro avg	0.49	0.50	0.50	15165
weighted avg	0.97	0.99	0.98	15165

```
XGBOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.98     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.98      0.98     15165


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.16      0.02      0.04       213

    accuracy                           0.98     15165
   macro avg       0.57      0.51      0.52     15165
weighted avg       0.97      0.98      0.98     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


SVM
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165
```

Version 2

From this version onwards, Sampling techniques were added. This helped increase the score value greatly. The definition added was 'sampling(df)'. This technique manually applied undersampling. Some of the best scores achieved are shown below. Also, updates we done to Boosting. Along with other models, they were added to Bagging Classifier with parameters, and then passed to GridSearchCV. It is important to note that Sampling should only be done on the Training data, and not on the entire dataset.

Adaboost Baseline Sampling

```
ADABOOST
              precision    recall  f1-score   support

           0       0.75      0.76      0.76       227
           1       0.75      0.73      0.74       218

    accuracy                           0.75       445
   macro avg       0.75      0.75      0.75       445
weighted avg       0.75      0.75      0.75       445
```

RandomForest Skew Sampling

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.74      0.75      0.75       227
           1       0.74      0.73      0.73       218

    accuracy                           0.74       445
   macro avg       0.74      0.74      0.74       445
weighted avg       0.74      0.74      0.74       445
```

XGBoost and RandomForest Scaling Sampling

```
XGBOOST
              precision    recall  f1-score   support

           0       0.74      0.74      0.74       227
           1       0.73      0.73      0.73       218

    accuracy                           0.73       445
   macro avg       0.73      0.73      0.73       445
weighted avg       0.73      0.73      0.73       445


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       227
           1       0.77      0.72      0.74       218

    accuracy                           0.76       445
   macro avg       0.76      0.76      0.76       445
weighted avg       0.76      0.76      0.76       445
```

Gradient Boosting GridSearch Sampling

```
              precision    recall  f1-score   support

           0       0.75      0.74      0.74       227
           1       0.73      0.75      0.74       218

    accuracy                           0.74       445
   macro avg       0.74      0.74      0.74       445
weighted avg       0.74      0.74      0.74       445

GradientBoostingClassifier(max_depth=6, n_estimators=46)
```

LinearSVC Baseline Sampling

LINEAR SVM

	precision	recall	f1-score	support
0	0.96	0.23	0.37	227
1	0.55	0.99	0.71	218
accuracy			0.60	445
macro avg	0.76	0.61	0.54	445
weighted avg	0.76	0.60	0.54	445

LinearSVC Outliers Sampling

LINEAR SVM

	precision	recall	f1-score	support
0	0.83	0.43	0.56	227
1	0.60	0.91	0.73	218
accuracy			0.66	445
macro avg	0.72	0.67	0.64	445
weighted avg	0.72	0.66	0.64	445

Final Version

Here, we added the function 'under_sample()' and 'over_sample()'. All Boosting, Bagging, and GridSearch code blocks were changed to Raw in this version. Reason being that Over Sampling greatly increased the score values right from the Baseline models (screenshot below) onwards, especially for DecisionTree and RandomForest.

DECISION TREE

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14952
1	0.83	1.00	0.91	213
accuracy			1.00	15165
macro avg	0.92	1.00	0.95	15165
weighted avg	1.00	1.00	1.00	15165

RANDOM FOREST

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14952
1	0.83	1.00	0.91	213
accuracy			1.00	15165
macro avg	0.92	1.00	0.95	15165
weighted avg	1.00	1.00	1.00	15165

Overall, RandomForest produced the best results. Even after some EDA and Preprocessing, the scores achieved for DecisionTree and RandomForest after each EDA process were almost identical, although there was a bit of variance in scores between the models. For this, we saved the model 'rfc_under' into a serial file through Pickle.