2022F-T2 AML 2053 – Big Data Algorithms and Statistics

Final Project

Professor: Jey Kanesh

Submitted by: Vengie Candar Dinampo

Table of Contents

Introduction 3
Overview 3
Problem Statement 3
About this dataset 3
Attribute information 4
1. Data Preprocessing: Data cleaning 5
1.1 Understanding the data and its dimension 5
1.2 Performing Clean Up of Data 7
1.3 Data Visualization 9
1.4 Removing Outlier 13
1.5 Create Hold-out test K-Folds 14
2. Exploratory Data Analysis (EDA) 15
2.1 Basic Exploration 15
2.2 Exploring Feature Distribution 16
2.3 Checking Correlation between Variables 20
2.4 Checking for Number of Categories in Each Categorical Feature 21
2.5 Check for Missing Values 22
2.6 Selecting Model and Metrics 23
3. Cross Validation 23
4. Feature Engineering 24
4.1 Imputing Missing Values in Numerical Variables 24
4.2 Imputing Missing Values in Categorical Variables 25
4.3 Encoding Categorical Variables 26
4.4 Checking for multicollinearity 30
4.5 Train Test Split 34
5. Model Building 35
6. Model Selection (With Pipeline) 39
6.1 Building a reusable pipeline to compare with another model 39
7. Fitting the Model 41
9. Model Optimisation 45
10. Testing model by Predicting the Test Data 47

Introduction

What is your data about, and what is the context?

Overview

This a final project of Term2-BDM 2053-Big Data Algorithms and Statistics. It’s all about vehicle insurance claim. In this business problem, I will be working with vehicle insurance claim data to demonstrate how to create a predictive model that predicts if a vehicle insurance claim is fraudulent or not. This will be a classification problem. With prediction data, to classify if the claim occurred or not.

This is to illustrate the building of a Machine Learning Pipeline:

- Data preparation and cleaning
- Exploratory Data Analysis
- Cross Validation
- Feature Engineering
- Building Model
- Model Selection
- Fitting the Model
- Model Optimision
- Predicting the test Data

Problem Statement

A classification problem of whether a vehicle insurance claim occurred, from start to finish. Thus, by classifying whether a claim occurred. Performing a simple hold-out validation as a test set. In the spirit of never having seen this artificially created test set.

About this dataset

The Dataset contains information on policyholders having the attributes like policy tenure, age of the car, age of the car owner, the population density of the city, make and model of the car, power, engine type, etc, and the target variable indicating whether the policyholder files a claim in the next 6 months or not.

Attribute information

Understanding the fields mean and how they might affect our target variables in theory. Based on our logical understanding of insurance, this is just wild speculation at this point.

Field Name	Description
1. ID:	Identification
2. KIDSDRIV:	Number of driving children
3. BIRTH:	Date of driver's birth
4. AGE:	Age of driver
5. HOMEKIDS:	Number of children at home
6. YOJ:	Years on Job
7. INCOME:	Income
8. PARENT1:	Single parent
9. HOME_VAL:	Home value
10. MSTATUS:	Marital status
11. GENDER:	Gender
12. EDUCATION:	Highest level of education
13. OCCUPATION:	Seniority at work
14. TRAVTIME:	Distance to work
15. CAR_USE:	Purpose of vehicle
16. BLUEBOOK:	Value of vehicle
17. TIF:	Time In-force
18. CAR_TYPE:	Type of vehicle
19. RED_CAR:	Whether or not vehicle is red in colour
20. OLDCLAIM:	Total claims in the past 5 years
21. CLM_FREQ:	Claim frequency in past 5 years
22. REVOKED:	License revoked in the past 7 years
23. MVR_PTS:	Motor Vehicle record points
24. CLM_AMT:	Claim Amount
25. CAR_AGE:	Age of car
26. CLAIM_FLAG:	Claim or no claim
27. URBANICITY:	Urban or rural

Explain your key target variable, what type of variable it is, and what are the independent variables, as well as what type of variables they are.

The dataset has 26 features (Please see table below for the type of variables), and the target variable is 'CLAIM_FLAG' a categorical binary.

Dataset Attribute Information			
#	Column	Description	Variable Type
0	ID	Identification	Nominal
1	KIDSDRIV	Number of driving children	Numerical
2	BIRTH	Date of driver's birth	Continuous
3	AGE	Age of driver	Numerical
4	HOMEKIDS	Number of children at home	Numerical
5	YOJ	Years on job	Numerical
6	INCOME	Income	Numerical
7	PARENT1	Single parent	Categorical (Binary)
8	HOME_VAL	Home value	Numerical
9	MSTATUS	Marital status	Categorical (Binary)
10	GENDER	Gender	Categorical (Binary)
11	EDUCATION	Highest level of education	Categorical (Ordinal)
12	OCCUPATION	Seniority at work	Categorical (Nominal)
13	TRAVTIME	Distance to work	Numerical
14	CAR_USE	Purpose of vehicle	Categorical (Binary)
15	BLUEBOOK	Value of vehicle	Numerical
16	TIF	Time In-force	Numerical
17	CAR_TYPE	Type of vehicle	Categorical (Nominal)
18	RED_CAR	Whether or not vehicle is red in colour	Categorical (Binary)
19	OLDCLAIM	Total claims in the past 5 years	Numerical
20	CLM_FREQ	Claim frequency in past 5 years	Numerical
21	REVOKED	License revoked in the past 7 years	Categorical (Binary)
22	MVR_PTS	Motor Vehicle record points	Numerical
23	CLM_AMT	Claim Amount	Numerical
24	CAR_AGE	Age of car	Numerical
25	CLAIM_FLAG	Claim or no claim	Categorical (Binary) -> Target variable
26	URBANICITY	Urban or rural	Categorical (Binary)

1. Data Preprocessing: Data cleaning

1.1 Understanding the data and its dimension

Data Loading

loading the data and working with the data description.

In [2]:
```
data = 'car_insurance_claim1.csv'
df = pd.read_csv(data)
pd.set_option('display.max_columns', None)
df.head()
```

Out[2]:

	ID	KIDSDRIV	BIRTH	AGE	HOMEKIDS	YOJ	INCOME	PARENT1	HOME_VAL	MSTATUS	GENDER	EDUCATION	OCCUPATION	TRAVTIME	CA
0	63581743	0	16MAR39	60.0	0	11.0	$67,349	No	$0	z_No	M	PhD	Professional	14	
1	132761049	0	21JAN56	43.0	0	11.0	$91,449	No	$257,252	z_No	M	z_High School	z_Blue Collar	22	Com
2	921317019	0	18NOV51	48.0	0	11.0	$52,881	No	$0	z_No	M	Bachelors	Manager	26	
3	727598473	0	05MAR64	35.0	1	10.0	$16,039	No	$124,191	Yes	z_F	z_High School	Clerical	5	
4	450221861	0	05JUN48	51.0	0	14.0	NaN	No	$306,251	Yes	M	<High School	z_Blue Collar	32	

In [3]:
```
df.shape
```

Out[3]: (10302, 27)

Note: Checking the data shape, data has 10302 rows and 27 columns.

In [4]:
```
df.columns
```

Out[4]:
```
Index(['ID', 'KIDSDRIV', 'BIRTH', 'AGE', 'HOMEKIDS', 'YOJ', 'INCOME',
       'PARENT1', 'HOME_VAL', 'MSTATUS', 'GENDER', 'EDUCATION', 'OCCUPATION',
       'TRAVTIME', 'CAR_USE', 'BLUEBOOK', 'TIF', 'CAR_TYPE', 'RED_CAR',
       'OLDCLAIM', 'CLM_FREQ', 'REVOKED', 'MVR_PTS', 'CLM_AMT', 'CAR_AGE',
       'CLAIM_FLAG', 'URBANICITY'],
      dtype='object')
```

Note: Showing the 27 column names

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10302 entries, 0 to 10301
Data columns (total 27 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          10302 non-null  int64
 1   KIDSDRIV    10302 non-null  int64
 2   BIRTH       10302 non-null  object
 3   AGE         10295 non-null  float64
 4   HOMEKIDS    10302 non-null  int64
 5   YOJ         9754 non-null   float64
 6   INCOME      9732 non-null   object
 7   PARENT1     10302 non-null  object
 8   HOME_VAL    9727 non-null   object
 9   MSTATUS     10302 non-null  object
 10  GENDER      10302 non-null  object
 11  EDUCATION   10302 non-null  object
 12  OCCUPATION  9637 non-null   object
 13  TRAVTIME    10302 non-null  int64
 14  CAR_USE     10302 non-null  object
 15  BLUEBOOK    10302 non-null  object
 16  TIF         10302 non-null  int64
 17  CAR_TYPE    10302 non-null  object
 18  RED_CAR     10302 non-null  object
 19  OLDCLAIM    10302 non-null  object
 20  CLM_FREQ    10302 non-null  int64
 21  REVOKED     10302 non-null  object
 22  MVR_PTS     10302 non-null  int64
 23  CLM_AMT     10302 non-null  object
 24  CAR_AGE     9663 non-null   float64
 25  CLAIM_FLAG  10302 non-null  int64
 26  URBANICITY  10302 non-null  object
dtypes: float64(3), int64(8), object(16)
memory usage: 2.1+ MB
```

In [6]:
```
print(f"Dataset has {df.shape[0]} rows and {df.shape[1]} columns")
print(f"Duplicates: {df.duplicated().sum()}")
print(f"Total Missing Values: {df.isna().sum().sum()}")
print(f"Number of rows with missing values: {df.isna().any(axis=1).sum()}")
```

```
Dataset has 10302 rows and 27 columns
Duplicates: 1
Total Missing Values: 3004
Number of rows with missing values: 2645
```

Note: As observe in the data info that there are null records in the dataset. Next step, is to explore the missing values

1.2 Performing Clean Up of Data

1.2 Data Preprocessing

Performing Clean Up of Data

data processing of cleaning up of data to remove duplicates and remove columns that don't add any value.

In [16]:
```
# Removing the duplicate in the dataframe
df.drop_duplicates(inplace=True)
```

In [17]:
```
# Convert currecy into floats
from re import sub
from decimal import Decimal

#Function to convert the data type to numerical type
def convert_currency(df, columns: list):
    for col in columns:
        df[col] = np.where(pd.isnull(df[col]), df[col], df[col]
                        .astype('str')
                        .map(lambda x: x.replace(',','').replace('$',''))).astype('float')

currency_cols = ['INCOME','HOME_VAL','BLUEBOOK','OLDCLAIM','CLM_AMT']
convert_currency(df, currency_cols)
df
```

Out[17]:

	ID	KIDSDRIV	BIRTH	AGE	HOMEKIDS	YOJ	INCOME	PARENT1	HOME_VAL	MSTATUS	GENDER	EDUCATION	OCCUPATION	TRAVTIME
0	63581743	0	16MAR39	60.0	0	11.0	67349.0	No	0.0	z_No	M	PhD	Professional	14
1	132761049	0	21JAN56	43.0	0	11.0	91449.0	No	257252.0	z_No	M	z_High School	z_Blue Collar	22
2	921317019	0	18NOV51	48.0	0	11.0	52881.0	No	0.0	z_No	M	Bachelors	Manager	26
3	727598473	0	05MAR64	35.0	1	10.0	16039.0	No	124191.0	Yes	z_F	z_High School	Clerical	5
4	450221861	0	05JUN48	51.0	0	14.0	NaN	No	306251.0	Yes	M	<High School	z_Blue Collar	32
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10297	67790126	1	13AUG54	45.0	2	9.0	164669.0	No	386273.0	Yes	M	PhD	Manager	21
10298	61970712	0	17JUN53	46.0	0	9.0	107204.0	No	332591.0	Yes	M	Masters	NaN	36
10299	849208064	0	18JUN51	48.0	0	15.0	39837.0	No	170611.0	Yes	z_F	<High School	z_Blue Collar	12

Below screenshot is all about removing values that have a prefix 'z_' that does not mean anything ad dropping a column

```
In [18]: # There are some values that have a prefix 'z_' that does not mean anything.
def remove_z(df, columns: list):
    for col in columns:
        df[col] = np.where(pd.isnull(df[col]), df[col], df[col].astype('str').map(lambda x: x.replace('z_','')))


z_cols = ['MSTATUS', 'GENDER', 'EDUCATION', 'OCCUPATION', 'CAR_TYPE','URBANICITY']
remove_z(df, z_cols)
```

```
In [19]: # Renaming  target columns with a prefix 'TGT_' to have better visualization on the target variable
df.rename({'CLM_AMT': 'TGT_CLAIM_AMT', 'CLAIM_FLAG': 'TGT_CLAIM_FLAG'}, axis=1, inplace=True);
```

```
In [20]: # Remove columns don't add any value. 'BIRTH' is redundant with the 'AGE' column present
df.drop(['BIRTH','ID'], axis=1, inplace=True);
df.head()
```

Out[20]:

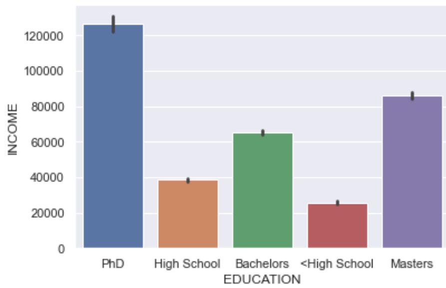
	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	PARENT1	HOME_VAL	MSTATUS	GENDER	EDUCATION	OCCUPATION	TRAVTIME	CAR_USE	BLUEBOOK	T
0	0	60.0	0	11.0	67349.0	No	0.0	No	M	PhD	Professional	14	Private	14230.0	
1	0	43.0	0	11.0	91449.0	No	257252.0	No	M	High School	Blue Collar	22	Commercial	14940.0	
2	0	48.0	0	11.0	52881.0	No	0.0	No	M	Bachelors	Manager	26	Private	21970.0	
3	0	35.0	1	10.0	16039.0	No	124191.0	Yes	F	High School	Clerical	5	Private	4010.0	
4	0	51.0	0	14.0	NaN	No	306251.0	Yes	M	<High School	Blue Collar	32	Private	15440.0	

1.3 Data Visualization

1.3 Data Visualization

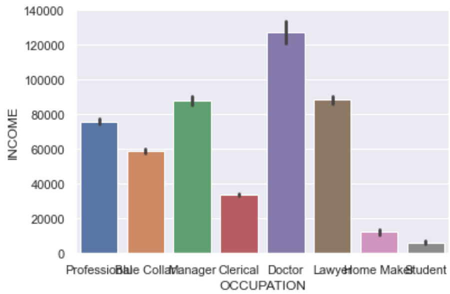
In [25]:
```
sns.set_theme()
sns.barplot(data = df,x = 'EDUCATION', y = 'INCOME', label = 'CLAIM_FLAG')
```

Out[25]:
```
<AxesSubplot:xlabel='EDUCATION', ylabel='INCOME'>
```



In [26]:
```
sns.set_theme()
sns.barplot(data = df,x = 'OCCUPATION', y = 'INCOME', label = 'CLAIM_FLAG')
```

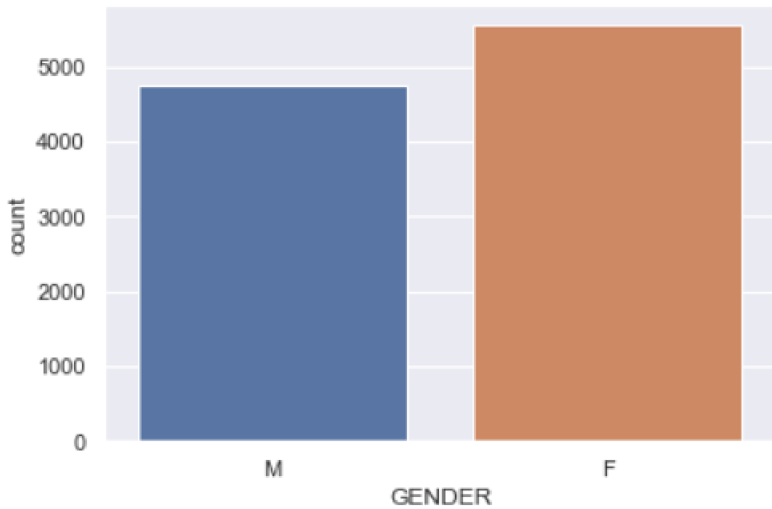
Out[26]:
```
<AxesSubplot:xlabel='OCCUPATION', ylabel='INCOME'>
```



In [27]:
```
sns.set_theme()
sns.countplot(data = df, x = 'GENDER')
```

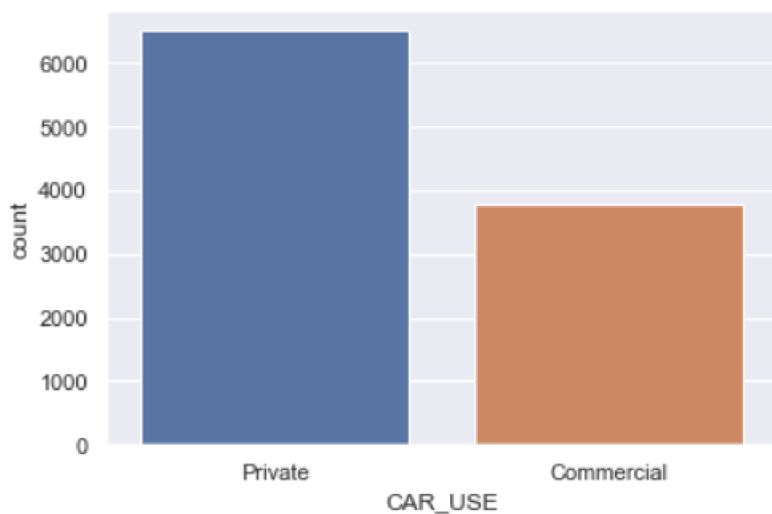
```
']: sns.set_theme()
    sns.countplot(data = df, x = 'GENDER')
```

']: <AxesSubplot:xlabel='GENDER', ylabel='count'>



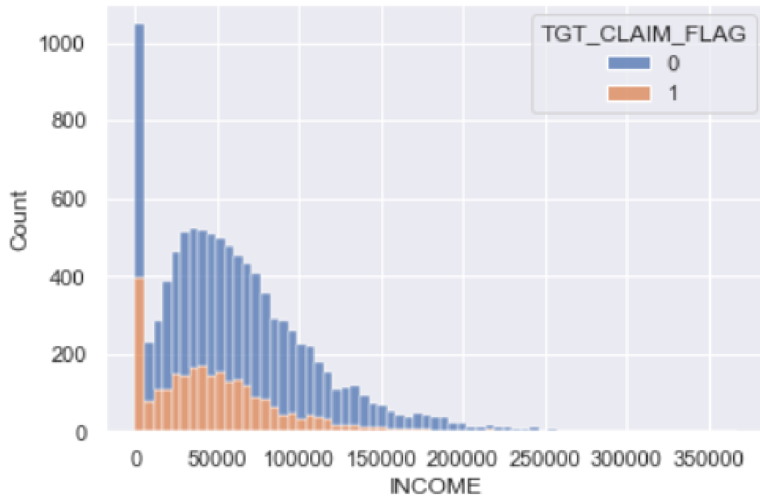

```
In [28]: sns.set_theme()
         sns.countplot(data = df, x = 'CAR_USE')
```

Out[28]: <AxesSubplot:xlabel='CAR_USE', ylabel='count'>



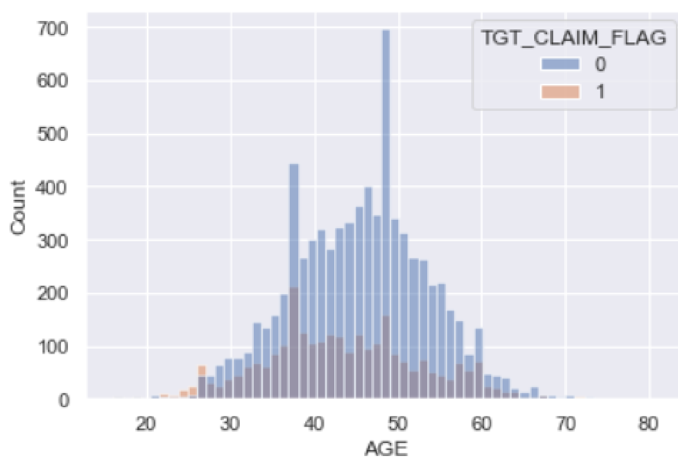
In [29]:
```
sns.set_theme()
sns.histplot(df, x = 'INCOME', hue = 'TGT_CLAIM_FLAG', multiple = 'stac
```

Out[29]: <AxesSubplot:xlabel='INCOME', ylabel='Count'>



In [30]:
```
sns.set_theme()
sns.histplot(data = df, x = 'AGE', hue = 'TGT_CLAIM_FLAG')
```

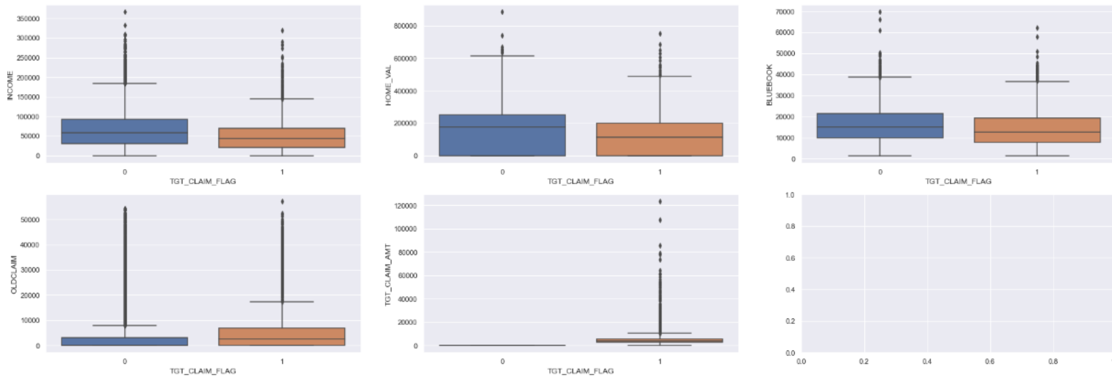
Out[30]: <AxesSubplot:xlabel='AGE', ylabel='Count'>



Checking for outlier using boxplot

In [31]:
```python
sns.set_theme()
figure, axis = plt.subplots(2,3, figsize = (30,10))
sns.boxplot(y = 'INCOME', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[0,0])
sns.boxplot(y = 'HOME_VAL', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[0,1])
sns.boxplot(y = 'BLUEBOOK', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[0,2])
sns.boxplot(y = 'OLDCLAIM', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[1,0])
sns.boxplot(y = 'TGT_CLAIM_AMT', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[1,1])
```

Out[31]: <AxesSubplot:xlabel='TGT_CLAIM_FLAG', ylabel='TGT_CLAIM_AMT'>



Note: As observed in the boxplot figure, its shows the outliers. Next step, is I gonna remove the outlier.

1.4 Removing Outlier

1.4 Removing Outlier

In [32]:
```python
def IQR(data):
  Q1 = data.quantile(0.25)
  Q3 = data.quantile(0.75)

  IQR = Q3 - Q1
  LR = Q1 - (IQR * 1.5)
  UR = Q3 + (IQR * 1.5)

  return LR, UR
```

In [33]:
```python
class_one = df[df['TGT_CLAIM_FLAG'] == 1]
class_zero = df[df['TGT_CLAIM_FLAG'] == 0]
```

In [34]:
```python
LR1, UR1 = IQR(class_one.INCOME)
LR0, UR0 = IQR(class_zero.INCOME)
class_one = class_one[(class_one['INCOME'] > LR1) & (class_one['INCOME'] < UR1)]
class_zero = class_zero[(class_zero['INCOME'] > LR0) & (class_zero['INCOME'] < UR0)]
```

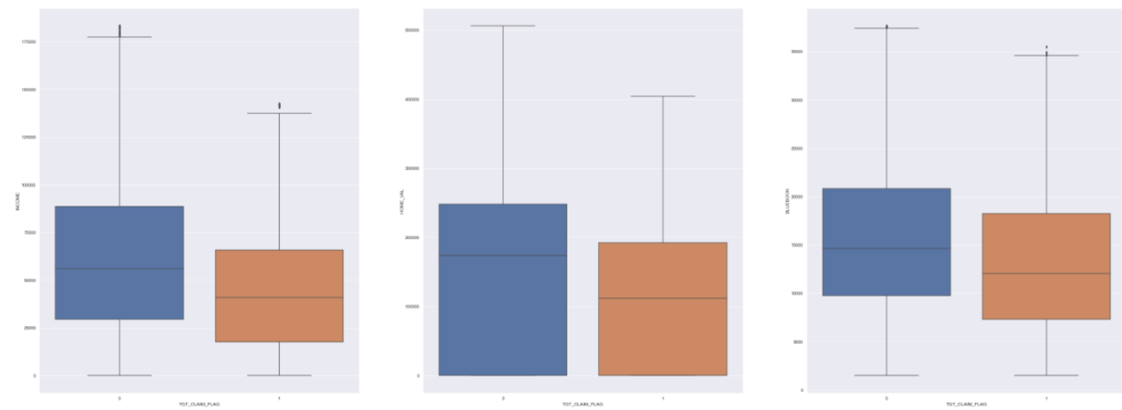
In [35]:
```python
LR1, UR1 = IQR(class_one.HOME_VAL)
LR0, UR0 = IQR(class_zero.HOME_VAL)
class_one = class_one[(class_one['HOME_VAL'] > LR1) & (class_one['HOME_VAL'] < UR1)]
class_zero = class_zero[(class_zero['HOME_VAL'] > LR0) & (class_zero['HOME_VAL'] < UR0)]
```

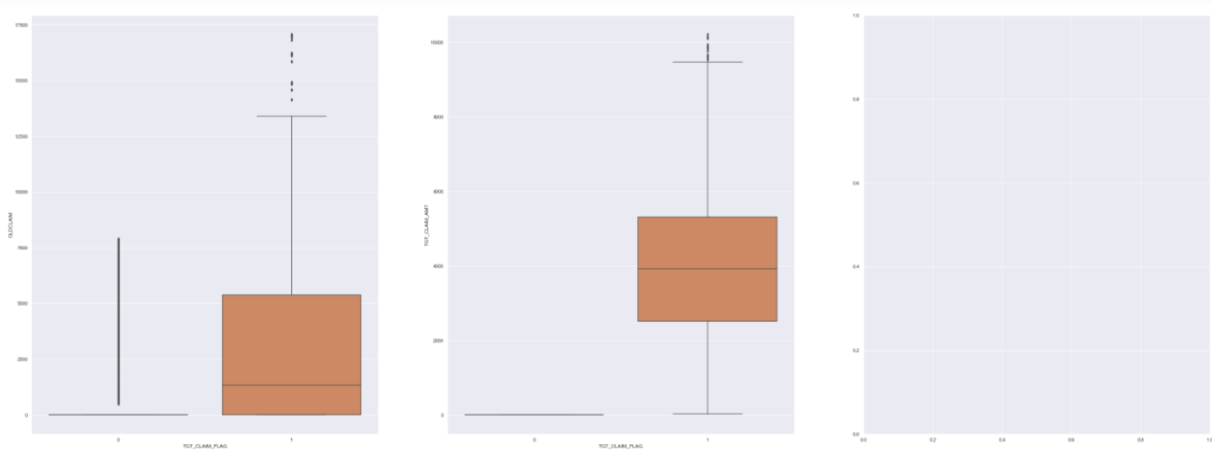
In [36]:
```python
LR1, UR1 = IQR(class_one.BLUEBOOK)
LR0, UR0 = IQR(class_zero.BLUEBOOK)
class_one = class_one[(class_one['BLUEBOOK'] > LR1) & (class_one['BLUEBOOK'] < UR1)]
class_zero = class_zero[(class_zero['BLUEBOOK'] > LR0) & (class_zero['BLUEBOOK'] < UR0)]
```

In [37]:
```python
LR1, UR1 = IQR(class_one.OLDCLAIM)
LR0, UR0 = IQR(class_zero.OLDCLAIM)
class_one = class_one[(class_one['OLDCLAIM'] > LR1) & (class_one['OLDCLAIM'] < UR1)]
class_zero = class_zero[(class_zero['OLDCLAIM'] > LR0) & (class_zero['OLDCLAIM'] < UR0)]
```

```
sns.boxplot(y = 'HOME_VAL', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[0,1])
sns.boxplot(y = 'BLUEBOOK', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[0,2])
sns.boxplot(y = 'OLDCLAIM', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[1,0])
sns.boxplot(y = 'TGT_CLAIM_AMT', x = 'TGT_CLAIM_FLAG', data = df, ax = axis[1,1])
```

Out[40]: <AxesSubplot:xlabel='TGT_CLAIM_FLAG', ylabel='TGT_CLAIM_AMT'>





1.5 Create Hold-out test K-Folds

In [41]:
```python
from sklearn.model_selection import StratifiedKFold

def make_stratified_k_folds(df,tgt_col:str,n_splits):
    # Randomise and reset index for splitting
    df = df.sample(frac=1,random_state=0).reset_index(drop=True)
    n_rows = df.shape[0]

    # Calculate k in Sturges Formula
    n_bins = int(np.floor(np.log2(n_rows) + 1))

    # Create bins
    df.loc[:,'bins'] = pd.cut(
        df[tgt_col], bins=n_bins, labels=False
    )

    skf = StratifiedKFold(n_splits=n_splits)
    for f, (t_, v_) in enumerate(skf.split(X=df, y=df['bins'].values)):
        df.loc[v_, 'kfold'] = f

    df = df.drop('bins',axis=1)
    return df
```

In [42]:
```python
# Create hold-out test fold
pre_df = make_stratified_k_folds(df,'TGT_CLAIM_AMT',8)
train_df = pre_df.copy().loc[pre_df['kfold'] != 0].drop('kfold', axis=1)
test_df = pre_df.copy().loc[pre_df['kfold'] == 0].drop('kfold', axis=1)
```

2. Exploratory Data Analysis (EDA)

2.1 Basic Exploration

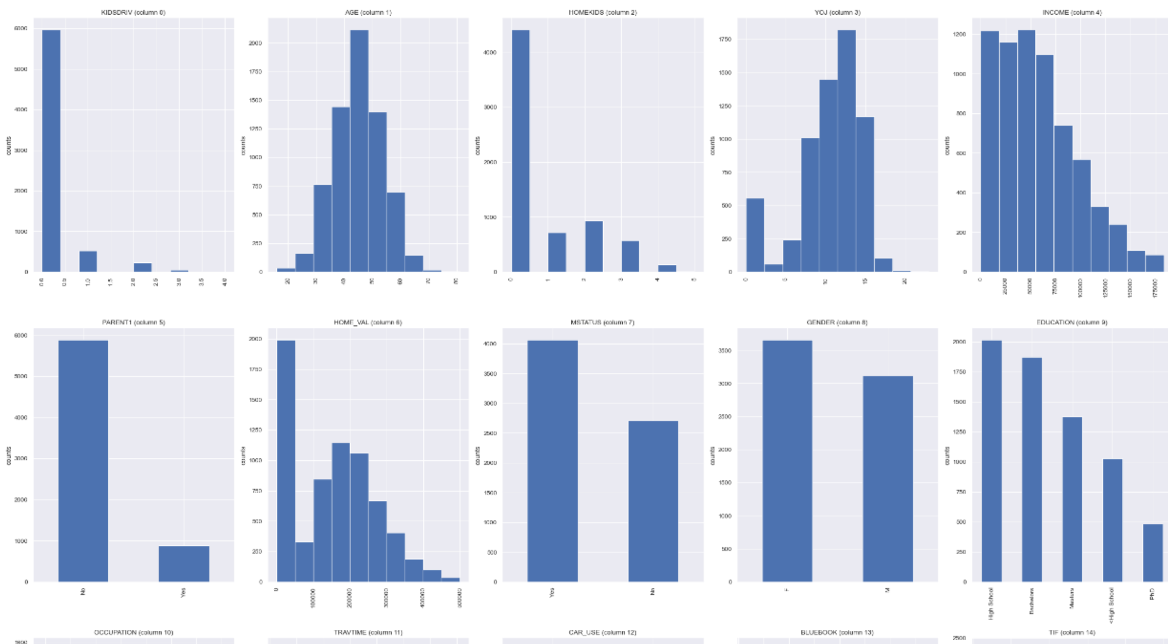
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6775 entries, 604 to 7742
Data columns (total 25 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   KIDSDRIV        6775 non-null   int64  
 1   AGE             6771 non-null   float64
 2   HOMEKIDS        6775 non-null   int64  
 3   YOJ             6416 non-null   float64
 4   INCOME          6775 non-null   float64
 5   PARENT1         6775 non-null   object 
 6   HOME_VAL        6775 non-null   float64
 7   MSTATUS         6775 non-null   object 
 8   GENDER          6775 non-null   object 
 9   EDUCATION       6775 non-null   object 
 10  OCCUPATION      6423 non-null   object 
 11  TRAVTIME        6775 non-null   int64  
 12  CAR_USE         6775 non-null   object 
 13  BLUEBOOK        6775 non-null   float64
 14  TIF             6775 non-null   int64  
 15  CAR_TYPE        6775 non-null   object 
 16  RED_CAR         6775 non-null   object 
 17  OLDCLAIM        6775 non-null   float64
 18  CLM_FREQ        6775 non-null   int64  
 19  REVOKED         6775 non-null   object 
 20  MVR_PTS         6775 non-null   int64  
 21  TGT_CLAIM_AMT   6775 non-null   float64
 22  CAR_AGE         6348 non-null   float64
 23  TGT_CLAIM_FLAG  6775 non-null   int64  
 24  URBANICITY      6775 non-null   object 
dtypes: float64(8), int64(7), object(10)
memory usage: 1.3+ MB
```

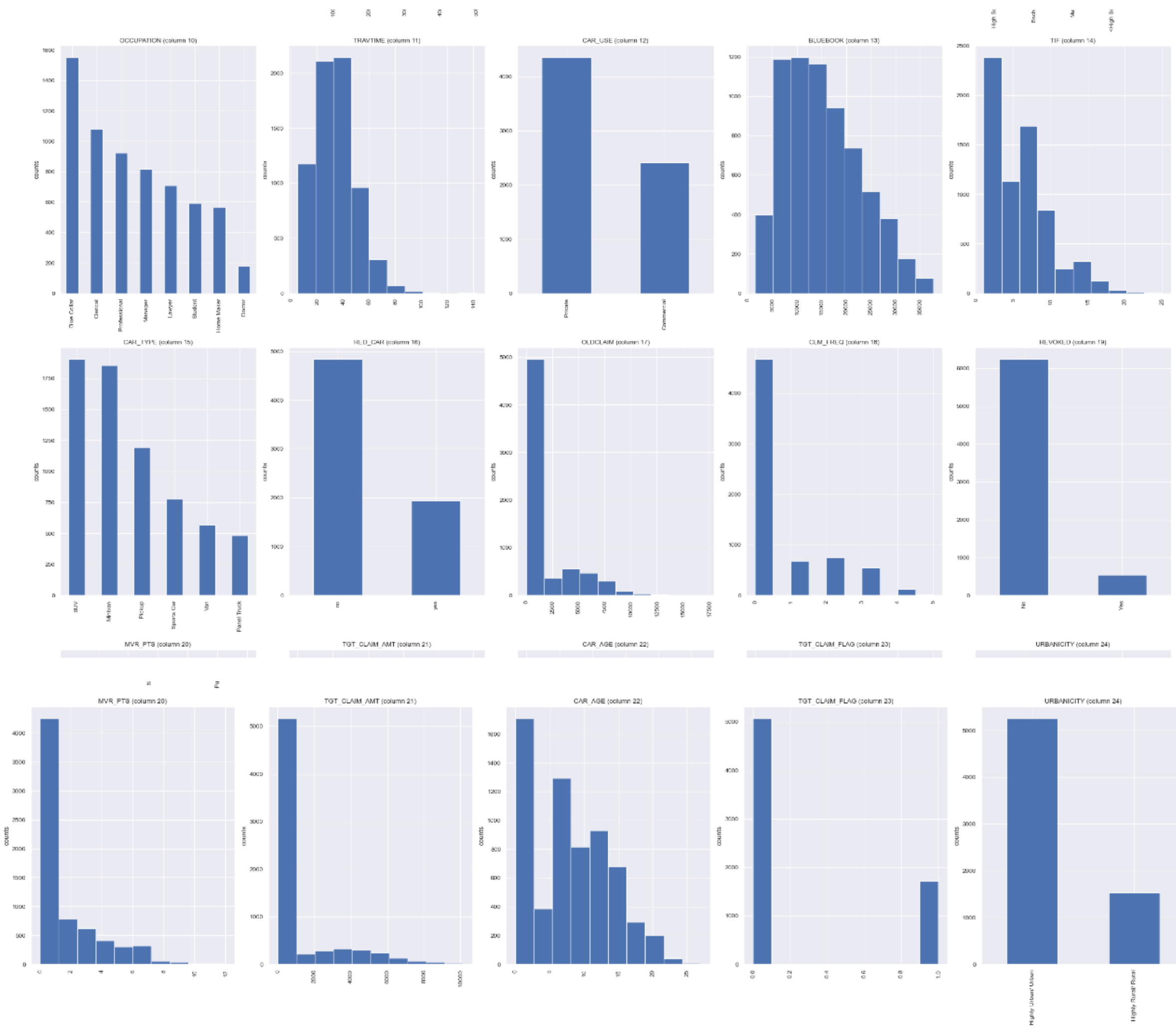
In [45]:
```
print(f"Dataset has {train_df.shape[0]} rows and {train_df.shape[1]} columns")
print(f"Duplicates: {train_df.duplicated().sum()}")
print(f"Total Missing Values: {train_df.isna().sum().sum()}")
print(f"Number of rows with missing values: {train_df.isna().any(axis=1).sum()}")
```

```
Dataset has 6775 rows and 25 columns
Duplicates: 0
Total Missing Values: 1142
Number of rows with missing values: 1076
```

2.2 Exploring Feature Distribution

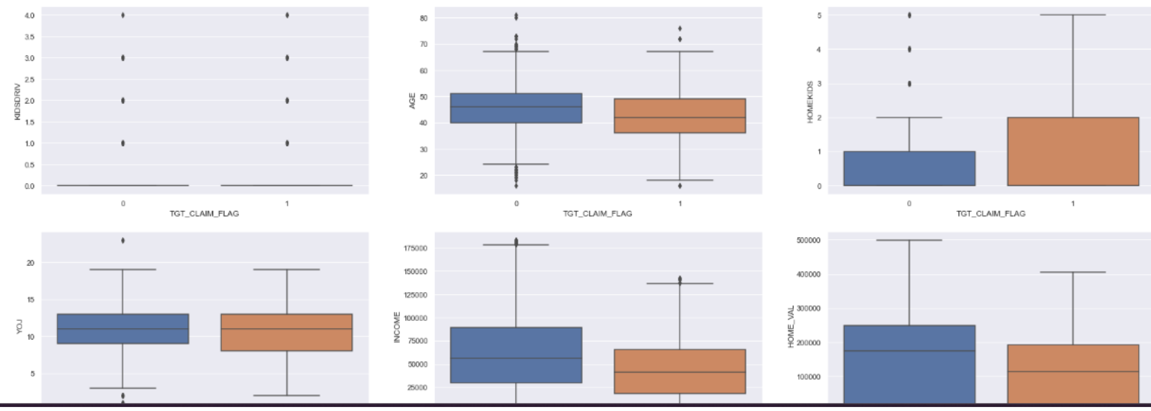
```python
def plot_col_distribution(df, n_graph_per_row):
    n_col = df.shape[1]
    column_names = list(df)
    n_graph_row = (n_col + n_graph_per_row - 1) // n_graph_per_row
    plt.figure(num = None, figsize = (6 * n_graph_per_row, 8 * n_graph_row), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(n_col):
        plt.subplot(n_graph_row, n_graph_per_row, i + 1)
        column_df = df.iloc[:, i]
        if (not np.issubdtype(type(column_df.iloc[0]), np.number)):
            column_df.value_counts().plot.bar()
        else:
            column_df.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{column_names[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()

plot_col_distribution(train_df,5)
```

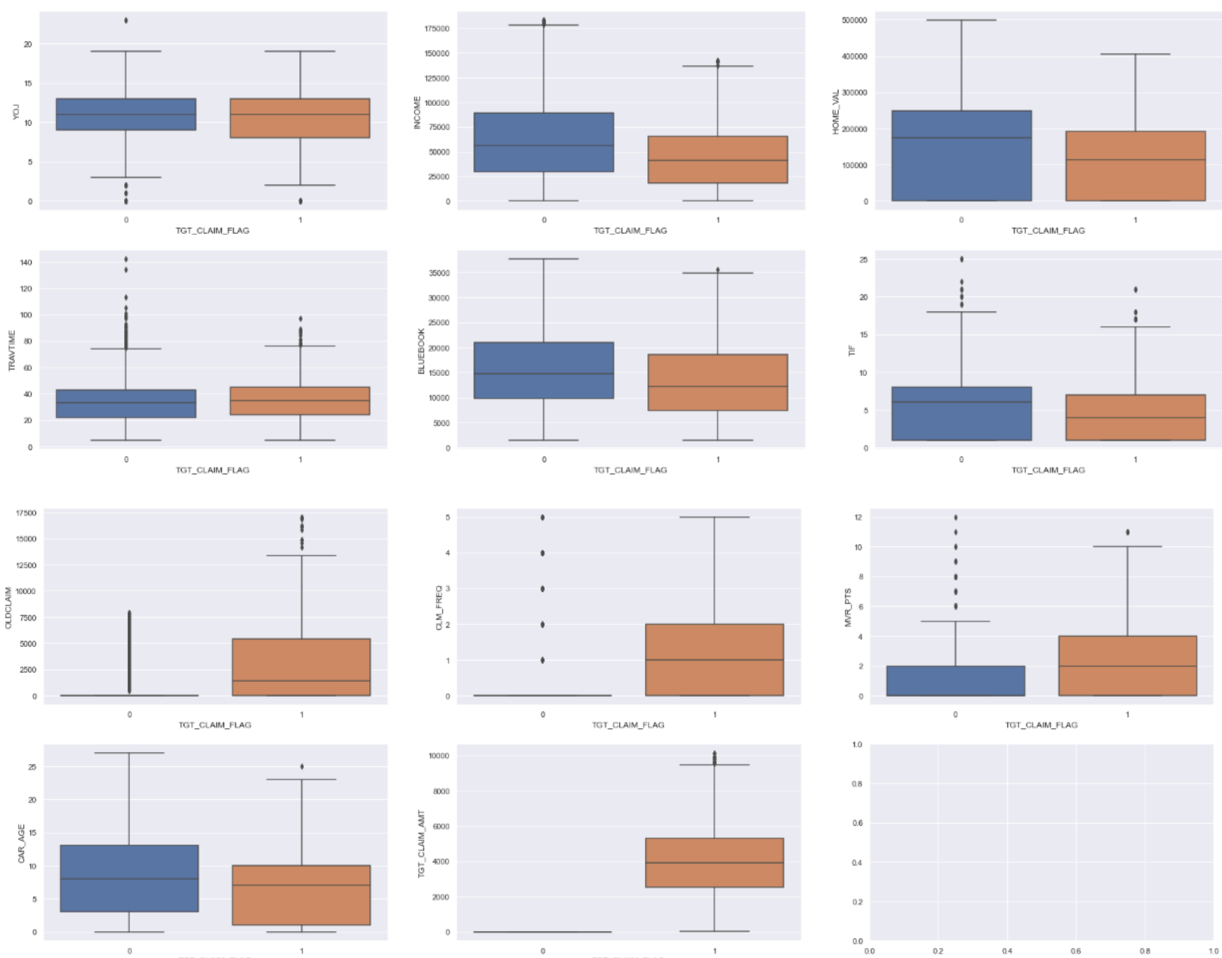



OCCUPATION (column 10)
TRAVTIME (column 11)
CAR_USE (column 12)
BLUEBOOK (column 13)
TIF (column 14)
CAR_TYPE (column 15)
RED_CAR (column 16)
OLDCLAIM (column 17)
CLM_FREQ (column 18)
REVOKED (column 19)
MVR_PTS (column 20)
TGT_CLAIM_AMT (column 21)
CAR_AGE (column 22)
TGT_CLAIM_FLAG (column 23)
URBANICITY (column 24)

n [47]:

```
def plot_boxplot_by_claim_flag(df,cols,n_graph_per_row):
    sns.set_theme()
    n_rows, n_cols = 5,3
    figure, axis = plt.subplots(n_rows, n_cols, figsize = (30,30))
    i = 0
    for col in cols:
        j = i//n_cols
        k = i%n_cols
        sns.boxplot(y=col, x='TGT_CLAIM_FLAG', data=df, ax=axis[j,k])
        i +=1

all_num_cols = numerical_cols + ['TGT_CLAIM_AMT']
plot_boxplot_by_claim_flag(train_df,all_num_cols,5)
```





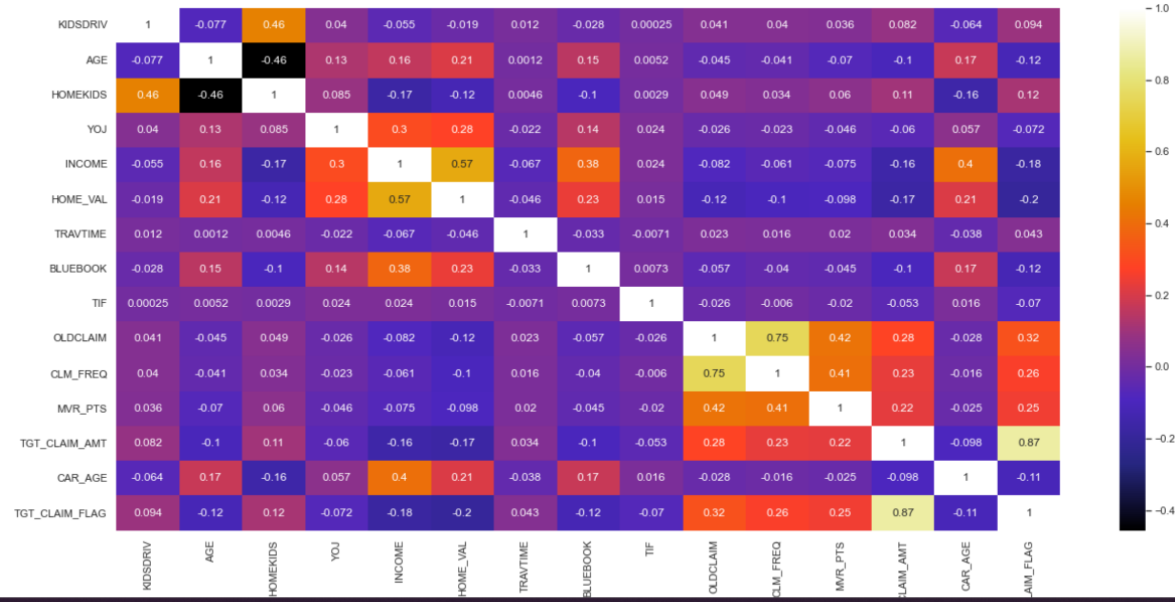
```
from scipy.stats import skew
for col in numerical_cols:
    print(f"{col} : {skew(train_df[col])}")
print(f"TGT_CLAIM_AMT : {skew(train_df['TGT_CLAIM_AMT'])}")
```

```
KIDSDRIV : 3.3604614013726164
AGE : nan
HOMEKIDS : 1.3339166469416346
YOJ : nan
INCOME : 0.6493657126398241
HOME_VAL : 0.20243804258617146
TRAVTIME : 0.46753288820208544
BLUEBOOK : 0.487205993113608
TIF : 0.8746313760275789
OLDCLAIM : 1.796190784141474
CLM_FREQ : 1.5379519284080385
MVR_PTS : 1.4933221288004255
CAR_AGE : nan
TGT_CLAIM_AMT : 1.9543473441657233
```

Note: Several of the numerical variables appear to be quite skewed. But despite the outliers, I'll make every effort to maintain the structure in order to make the most of it. The regression target's notable positive skew should also be noted.

2.3 Checking Correlation between Variables

[49]:
```python
def plot_corr(df):
    corr = df.corr()
    sns.set_theme()
    plt.figure(figsize = (22,10))
    sns.heatmap(corr, cbar=True, annot=True, cmap='CMRmap')
plot_corr(train_df)
```



2.4 Checking for Number of Categories in Each Categorical Feature

In [51]:
```python
for col in categorical_cols:
    print(col, 'has', df[col].nunique(),'unique variables')
```

```
PARENT1 has 2 unique variables
MSTATUS has 2 unique variables
GENDER has 2 unique variables
EDUCATION has 5 unique variables
OCCUPATION has 8 unique variables
CAR_USE has 2 unique variables
CAR_TYPE has 6 unique variables
RED_CAR has 2 unique variables
REVOKED has 2 unique variables
URBANICITY has 2 unique variables
```

Note: Fortunately, all categorical features have a reasonable number of categories.

Some can be categorised as binary, having only two states, since they have two distinct variables.

2.5 Check for Missing Values

2.5 Check for Missing Values

In [52]:
```
print(f"Missing Values in DF: {train_df.isna().sum().sum()}")
print("\n")

def find_missing(df, cols:list):
    for i in cols:
        print(" - ", i, f", Missing: {df[i].isna().sum()}")
    print("\n")

print("Categorical features are:")
find_missing(train_df, categorical_cols)
print("Numerical features are:")
find_missing(train_df, numerical_cols)

print("Targets")
find_missing(train_df, ['TGT_CLAIM_FLAG','TGT_CLAIM_AMT'])
```

```
Missing Values in DF: 1142


Categorical features are:
 -  PARENT1 , Missing: 0
 -  MSTATUS , Missing: 0
 -  GENDER , Missing: 0
 -  EDUCATION , Missing: 0
 -  OCCUPATION , Missing: 352
 -  CAR_USE , Missing: 0
 -  CAR_TYPE , Missing: 0
 -  RED_CAR , Missing: 0
 -  REVOKED , Missing: 0
 -  URBANICITY , Missing: 0
```

```
Missing Values in DF: 1142


Categorical features are:
 -  PARENT1 , Missing: 0
 -  MSTATUS , Missing: 0
 -  GENDER , Missing: 0
 -  EDUCATION , Missing: 0
 -  OCCUPATION , Missing: 352
 -  CAR_USE , Missing: 0
 -  CAR_TYPE , Missing: 0
 -  RED_CAR , Missing: 0
 -  REVOKED , Missing: 0
 -  URBANICITY , Missing: 0


Numerical features are:
 -  KIDSDRIV , Missing: 0
 -  AGE , Missing: 4
 -  HOMEKIDS , Missing: 0
 -  YOJ , Missing: 359
 -  INCOME , Missing: 0
 -  HOME_VAL , Missing: 0
 -  TRAVTIME , Missing: 0
 -  BLUEBOOK , Missing: 0
 -  TIF , Missing: 0
 -  OLDCLAIM , Missing: 0
 -  CLM_FREQ , Missing: 0
 -  MVR_PTS , Missing: 0
 -  CAR_AGE , Missing: 427


Targets
 -  TGT_CLAIM_FLAG , Missing: 0
```

2.6 Selecting Model and Metrics

For the TGT CLAIM FLAG classification issue:

This is a problem of binary classification.

The F1-score, which is calculated as a "harmonic mean" of Precision and Recall, is a fantastic metric to use in such circumstances. Since the target has a high degree of skewness.

3. Cross Validation

3. Cross Validation

I'll start by dividing the data here in order to train the models while using cross-validation. once more employ a stratified K-fold s
fold will be recorded in a new column called kfold. Instead of stratifying on 'TGT CLAIM FLAG'

In [53]:
```
n_folds = 5
train_cv_df = make_stratified_k_folds(train_df,'TGT_CLAIM_AMT',n_folds)
```

In [54]:
```
train_cv_df.shape
```

Out[54]:
```
(6775, 26)
```

In [55]:
```
train_cv_df.columns
```

Out[55]:
```
Index(['KIDSDRIV', 'AGE', 'HOMEKIDS', 'YOJ', 'INCOME', 'PARENT1', 'HOME_VAL',
       'MSTATUS', 'GENDER', 'EDUCATION', 'OCCUPATION', 'TRAVTIME', 'CAR_USE',
       'BLUEBOOK', 'TIF', 'CAR_TYPE', 'RED_CAR', 'OLDCLAIM', 'CLM_FREQ',
       'REVOKED', 'MVR_PTS', 'TGT_CLAIM_AMT', 'CAR_AGE', 'TGT_CLAIM_FLAG',
       'URBANICITY', 'kfold'],
      dtype='object')
```

4. Feature Engineering

4.1 Imputing Missing Values in Numerical Variables

When possible, I'll use a KNN imputation for features with missing values since it "interpolates" data based on the information that is currently available.

Since categorical values cannot be imputed using KNN, I will only use KNN on numerical features.

n [56]:
```
from sklearn.impute import KNNImputer, SimpleImputer

# Define imputers we'll be using
knn_imputer = KNNImputer(n_neighbors=2)

# Function to show a sample of the missing values that will be imputed
def missing_head(df,cols,n=5):
    df = df[cols]
    missing = df[df.isna().any(axis=1)].head(n)
    print('These are some samples with missing values')
    display(missing)
    print('\n')
    return missing.index.to_list()
```

n [57]:
```
# Some boiler plate code to impute and then sense check imputed values
def imputer_test(input_df, cols, imputer, test_samples):

    df = input_df[cols]
    df_imputed = pd.DataFrame(imputer.fit_transform(df))
    df_imputed.columns = df.columns
    print('The same samples with missing values imputed')
    display(df_imputed.loc[test_samples])

    imputed_missing_df = df_imputed[df_imputed.isna().any(axis=1)]
    print('Sense check for any empty values remaining')
    display(imputed_missing_df)
    print('The imputed df is empty:',imputed_missing_df.empty)
    print('\n')
    return df_imputed
```

58]:
```python
# Find missing values and impute numerical columns
print('NUMERICAL FEATURES')
missing_num_samples = missing_head(train_cv_df, numerical_cols)
train_cv_df_num_imp = imputer_test(train_cv_df, numerical_cols, knn_imputer, missing_num_samples)
```

```
NUMERICAL FEATURES
These are some samples with missing values
```

	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	HOME_VAL	TRAVTIME	BLUEBOOK	TIF	OLDCLAIM	CLM_FREQ	MVR_PTS	CAR_AGE
0	0	38.0	3	16.0	16596.0	86339.0	47	7120.0	13	0.0	0	0	NaN
31	0	51.0	0	12.0	51628.0	206070.0	34	6940.0	4	0.0	0	0	NaN
32	0	39.0	0	12.0	40337.0	200448.0	14	6130.0	4	0.0	0	2	NaN
42	0	56.0	2	NaN	60286.0	213596.0	43	23480.0	10	0.0	0	0	11.0
47	0	53.0	0	NaN	147579.0	443598.0	50	16260.0	1	0.0	0	1	NaN

```
The same samples with missing values imputed
```

	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	HOME_VAL	TRAVTIME	BLUEBOOK	TIF	OLDCLAIM	CLM_FREQ	MVR_PTS	CAR_AGE
0	0.0	38.0	3.0	16.0	16596.0	86339.0	47.0	7120.0	13.0	0.0	0.0	0.0	9.5
31	0.0	51.0	0.0	12.0	51628.0	206070.0	34.0	6940.0	4.0	0.0	0.0	0.0	3.5
32	0.0	39.0	0.0	12.0	40337.0	200448.0	14.0	6130.0	4.0	0.0	0.0	2.0	6.5
42	0.0	56.0	2.0	12.5	60286.0	213596.0	43.0	23480.0	10.0	0.0	0.0	0.0	11.0
47	0.0	53.0	0.0	14.0	147579.0	443598.0	50.0	16260.0	1.0	0.0	0.0	1.0	16.5

4.2 Imputing Missing Values in Categorical Variables

4.2 Imputing Missing Values in Categorical Variables

Reusing the previous code, I'll temporarily store missing categorical values in a new category called "MISSING."

In [59]:
```python
# Define imputers we'll be using
categorical_imputer = SimpleImputer(strategy='constant', fill_value='MISSING')

# Repeat for categorical columns, with the Simple
print('CATEGORICAL FEATURES')
missing_categorical_samples = missing_head(train_cv_df, categorical_cols)
train_cv_df_categorical_imp = imputer_test(train_cv_df, categorical_cols, categorical_imputer, missing_categorical_
```

```
CATEGORICAL FEATURES
These are some samples with missing values
```

	PARENT1	MSTATUS	GENDER	EDUCATION	OCCUPATION	CAR_USE	CAR_TYPE	RED_CAR	REVOKED	URBANICITY
4	No	No	M	Masters	NaN	Commercial	Pickup	yes	No	Highly Urban/ Urban
15	No	No	M	Masters	NaN	Private	Minivan	no	No	Highly Urban/ Urban
29	Yes	No	M	Masters	NaN	Commercial	Panel Truck	yes	No	Highly Urban/ Urban
38	No	No	M	Masters	NaN	Commercial	Van	yes	No	Highly Urban/ Urban
57	No	Yes	M	Masters	NaN	Commercial	Panel Truck	yes	No	Highly Urban/ Urban

```
The same samples with missing values imputed
```

	PARENT1	MSTATUS	GENDER	EDUCATION	OCCUPATION	CAR_USE	CAR_TYPE	RED_CAR	REVOKED	URBANICITY
4	No	No	M	Masters	MISSING	Commercial	Pickup	yes	No	Highly Urban/ Urban
15	No	No	M	Masters	MISSING	Private	Minivan	no	No	Highly Urban/ Urban
29	Yes	No	M	Masters	MISSING	Commercial	Panel Truck	yes	No	Highly Urban/ Urban

In [60]:
```
# # Merge the two dataframes together
train_cv_df_imputed = pd.concat([train_cv_df_num_imp, train_cv_df_categorical_imp, train_cv_df[['TGT_CLAIM_FLAG','TGT_CLAIM_AMT',
print(train_cv_df_imputed.shape)
train_cv_df_imputed.head()
```

(6775, 26)

Out[60]:

	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	HOME_VAL	TRAVTIME	BLUEBOOK	TIF	OLDCLAIM	CLM_FREQ	MVR_PTS	CAR_AGE	PARENT1	MSTATUS
0	0.0	38.0	3.0	16.0	16596.0	86339.0	47.0	7120.0	13.0	0.0	0.0	0.0	9.5	No	Yes
1	0.0	33.0	1.0	11.0	14277.0	109348.0	34.0	6230.0	6.0	1225.0	3.0	3.0	5.0	No	Yes
2	0.0	38.0	0.0	10.0	34734.0	138910.0	38.0	8770.0	7.0	0.0	0.0	4.0	14.0	No	Yes
3	0.0	44.0	1.0	12.0	51120.0	0.0	36.0	26840.0	1.0	0.0	0.0	2.0	8.0	Yes	No
4	0.0	37.0	0.0	13.0	82444.0	226818.0	5.0	9740.0	1.0	0.0	0.0	1.0	15.0	No	No

Note: The missing numerical and categorical values are imputed, next is to encoding the categorical values.

4.3 Encoding Categorical Variables

4.3 Encoding Categorical Variables

Looking at the nunique values from the Exploratory Data Analysis (EDA), and some analysis of each categorical value, there are 3 typ
perform here.

- Ordinal category: Only Education appears to have some linkage between the classes
- Binary categories: These categories have only two classes
- Nominal categories: There are multiple classes and they do not have any relationship with one another

In [61]:
```
# Define what to do with each categorical column
categorical_cols_ordinal = ['EDUCATION']
categorical_cols_binary = ['PARENT1','MSTATUS','GENDER','CAR_USE','RED_CAR','REVOKED','URBANICITY']
categorical_cols_nominal = ['CAR_TYPE', 'OCCUPATION']
```

In [62]:
```
# Set the order of 'EDUCATION' for the ordinal encoder
EDUCATION_ordinal = [['<High School','High School','Bachelors','Masters','PhD']]
EDUCATION_ordinal
```

Out[62]: [['<High School', 'High School', 'Bachelors', 'Masters', 'PhD']]

In [63]:
```
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder, OneHotEncoder, StandardScaler
# Define encoder
ordinal_encoder_EDUCATION = OrdinalEncoder(categories=EDUCATION_ordinal)
binary_encoder = OrdinalEncoder()
oh_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
```

In [64]:
```
# Take only the categorical columns
train_cv_df_imputed_cat = train_cv_df_imputed[categorical_cols]
train_cv_df_imputed_cat
```

In [64]:

```python
# Take only the categorical columns
train_cv_df_imputed_cat = train_cv_df_imputed[categorical_cols]
train_cv_df_imputed_cat
```

Out[64]:

	PARENT1	MSTATUS	GENDER	EDUCATION	OCCUPATION	CAR_USE	CAR_TYPE	RED_CAR	REVOKED	URBANICITY
0	No	Yes	F	<High School	Clerical	Private	Sports Car	no	No	Highly Rural/ Rural
1	No	Yes	F	<High School	Professional	Private	SUV	no	No	Highly Urban/ Urban
2	No	Yes	M	Bachelors	Clerical	Commercial	Pickup	yes	No	Highly Rural/ Rural
3	Yes	No	M	High School	Professional	Commercial	Panel Truck	yes	Yes	Highly Urban/ Urban
4	No	No	M	Masters	MISSING	Commercial	Pickup	yes	No	Highly Urban/ Urban
...	...	...	...	...	...	...	...	...	...	...
6770	Yes	No	F	High School	Clerical	Private	Sports Car	no	No	Highly Rural/ Rural
6771	No	No	F	PhD	Home Maker	Private	Sports Car	no	No	Highly Urban/ Urban
6772	Yes	No	F	High School	Blue Collar	Commercial	Minivan	no	No	Highly Urban/ Urban
6773	Yes	No	F	<High School	Student	Private	SUV	no	No	Highly Urban/ Urban
6774	No	Yes	F	Masters	Lawyer	Private	Minivan	no	No	Highly Urban/ Urban

6775 rows × 10 columns

In [65]:

```python
# Encode each categorical type
## Ordinal categories
train_cv_df_categorical_enc_cat_ordinal = pd.DataFrame(ordinal_encoder_EDUCATION.fit_transform(train_cv_df_imputed_cat[categorica
train_cv_df_categorical_enc_cat_ordinal.columns = train_cv_df_imputed_cat[categorical_cols_ordinal].columns
train_cv_df_categorical_enc_cat_ordinal
```

Out[65]:

	EDUCATION
0	0.0
1	0.0
2	2.0
3	1.0
4	3.0
...	...
6770	1.0
6771	4.0
6772	1.0
6773	0.0
6774	3.0

6775 rows × 1 columns

In [66]:

```python
## Binary categories
train_cv_df_imp_enc_categorical_binary = pd.DataFrame(binary_encoder.fit_transform(train_cv_df_imputed_cat[categorical_cols_binar
train_cv_df_imp_enc_categorical_binary.columns = train_cv_df_imputed_cat[categorical_cols_binary].columns
train_cv_df_imp_enc_categorical_binary
```

Out[66]:

	PARENT1	MSTATUS	GENDER	CAR_USE	RED_CAR	REVOKED	URBANICITY
0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	1.0	0.0	0.0	1.0
2	0.0	1.0	1.0	0.0	1.0	0.0	0.0
3	1.0	0.0	1.0	0.0	1.0	1.0	1.0

```
In [67]: ## Nominal categories
train_cv_df_imp_enc_cat_nominal = pd.DataFrame(oh_encoder.fit_transform(train_cv_df_imputed_cat[categorical_cols_nominal]))
train_cv_df_imp_enc_cat_nominal.columns = oh_encoder.get_feature_names_out()
train_cv_df_imp_enc_cat_nominal
```

Out[67]:

	CAR_TYPE_Minivan	CAR_TYPE_Panel Truck	CAR_TYPE_Pickup	CAR_TYPE_SUV	CAR_TYPE_Sports Car	CAR_TYPE_Van	OCCUPATION_Blue Collar	OCCUPATION_Clerica
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.(
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.(
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.(
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.(
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.(
...	...	...	...	...	...	...	...	..
6770	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.(
6771	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.(
6772	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.(
6773	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.(
6774	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(

6775 rows × 15 columns

In [68]: # merge the three sub-df's

```
In [68]: # merge the three sub-df's
train_cv_df_imp_enc_cat = pd.concat([train_cv_df_categorical_enc_cat_ordinal, train_cv_df_imp_enc_categorical_binary, train_cv_df
train_cv_df_imp_enc_cat.head()
train_cv_df_imp_enc_cat
```

Out[68]:

	EDUCATION	PARENT1	MSTATUS	GENDER	CAR_USE	RED_CAR	REVOKED	URBANICITY	CAR_TYPE_Minivan	CAR_TYPE_Panel Truck	CAR_TYPE_Pickup
0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
2	2.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0
4	3.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
6770	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
6771	4.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
6772	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
6773	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
6774	3.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0

6775 rows × 23 columns

In [69]:
```
# Merge the targets and kfold indexes from the Cros Validation df
train_cv_df_imp_enc = pd.concat([train_cv_df_imputed[numerical_cols], train_cv_df_imp_enc_cat, train_cv_df[['TGT_CLAIM_FLAG','TGT
train_cv_df_imp_enc.head()
train_cv_df_imp_enc
```

Out[69]:

	KIDSDRIV	AGE	HOMEKIDS	YOJ	INCOME	HOME_VAL	TRAVTIME	BLUEBOOK	TIF	OLDCLAIM	CLM_FREQ	MVR_PTS	CAR_AGE	EDUCATION	PAR
0	0.0	38.0	3.0	16.0	16596.0	86339.0	47.0	7120.0	13.0	0.0	0.0	0.0	9.5	0.0	
1	0.0	33.0	1.0	11.0	14277.0	109348.0	34.0	6230.0	6.0	1225.0	3.0	3.0	5.0	0.0	
2	0.0	38.0	0.0	10.0	34734.0	138910.0	38.0	8770.0	7.0	0.0	0.0	4.0	14.0	2.0	
3	0.0	44.0	1.0	12.0	51120.0	0.0	36.0	26840.0	1.0	0.0	0.0	2.0	8.0	1.0	
4	0.0	37.0	0.0	13.0	82444.0	226818.0	5.0	9740.0	1.0	0.0	0.0	1.0	15.0	3.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6770	0.0	29.0	3.0	10.0	53643.0	0.0	54.0	10080.0	7.0	5910.0	2.0	7.0	11.0	1.0	
6771	0.0	36.0	0.0	0.0	0.0	68436.0	62.0	2680.0	6.0	0.0	0.0	0.0	21.0	4.0	
6772	0.0	29.0	2.0	11.0	47621.0	196036.0	26.0	17130.0	9.0	5584.0	3.0	6.0	1.0	1.0	
6773	0.0	37.0	1.0	12.0	5678.0	0.0	42.0	2070.0	1.0	6535.0	2.0	1.0	1.0	0.0	
6774	0.0	53.0	0.0	16.0	94786.0	331854.0	55.0	12060.0	1.0	4441.0	1.0	3.0	11.0	3.0	

6775 rows × 39 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6775 entries, 0 to 6774
Data columns (total 39 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   KIDSDRIV                 6775 non-null   float64
 1   AGE                      6775 non-null   float64
 2   HOMEKIDS                 6775 non-null   float64
 3   YOJ                      6775 non-null   float64
 4   INCOME                   6775 non-null   float64
 5   HOME_VAL                 6775 non-null   float64
 6   TRAVTIME                 6775 non-null   float64
 7   BLUEBOOK                 6775 non-null   float64
 8   TIF                      6775 non-null   float64
 9   OLDCLAIM                 6775 non-null   float64
 10  CLM_FREQ                 6775 non-null   float64
 11  MVR_PTS                  6775 non-null   float64
 12  CAR_AGE                  6775 non-null   float64
 13  EDUCATION                6775 non-null   float64
 14  PARENT1                  6775 non-null   float64
 15  MSTATUS                  6775 non-null   float64
 16  GENDER                   6775 non-null   float64
 17  CAR_USE                  6775 non-null   float64
 18  RED_CAR                  6775 non-null   float64
 19  REVOKED                  6775 non-null   float64
 20  URBANICITY               6775 non-null   float64
 21  CAR_TYPE_Minivan         6775 non-null   float64
 22  CAR_TYPE_Panel Truck     6775 non-null   float64
 23  CAR_TYPE_Pickup          6775 non-null   float64
 24  CAR_TYPE_SUV             6775 non-null   float64
 25  CAR_TYPE_Sports Car      6775 non-null   float64
 26  CAR_TYPE_Van             6775 non-null   float64
 27  OCCUPATION_Blue Collar   6775 non-null   float64
 28  OCCUPATION_Clerical      6775 non-null   float64
 29  OCCUPATION_Doctor        6775 non-null   float64
 30  OCCUPATION_Home Maker    6775 non-null   float64
```

Note: Now, there are no missing values and data type has ben converted to numerical. The data is now clean and ready for model selection. But before that, I need to evaluate the one-hot encoded variables for Multicollinearity issue.

4.4 Checking for multicollinearity

```
In [71]: def calc_vif(df):
             df_cols = df.columns
             vif_values = [
                 variance_inflation_factor(df.values, i) for i in ran
                 ]
             return pd.DataFrame(zip(df_cols, vif_values),columns=['V

         calc_vif(train_cv_df_imp_enc.drop(['TGT_CLAIM_FLAG','TGT_CLA
```

```
C:\Users\Vengie Dinampo\AppData\Roaming\Python\Python38\site
ng: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

Out[71]:

	Variable	VIF
0	KIDSDRIV	1.311130
1	AGE	1.480411
2	HOMEKIDS	2.102144
3	YOJ	1.467125
4	INCOME	3.059462
5	HOME_VAL	2.416044
6	TRAVTIME	1.040660
7	BLUEBOOK	1.910197
8	TIF	1.005576
9	OLDCLAIM	2.383528
10	CLM_FREQ	2.385528
11	MVR_PTS	1.264249
12	CAR_AGE	1.907642
13	EDUCATION	3.681239

14	PARENT1	1.851680
15	MSTATUS	2.088593
16	GENDER	3.289534
17	CAR_USE	2.301792
18	RED_CAR	1.785558
19	REVOKED	1.035908
20	URBANICITY	1.270913
21	CAR_TYPE_Minivan	inf
22	CAR_TYPE_Panel Truck	inf
23	CAR_TYPE_Pickup	inf
24	CAR_TYPE_SUV	inf
25	CAR_TYPE_Sports Car	inf
26	CAR_TYPE_Van	inf
27	OCCUPATION_Blue Collar	inf
28	OCCUPATION_Clerical	inf
29	OCCUPATION_Doctor	inf
30	OCCUPATION_Home Maker	inf
31	OCCUPATION_Lawyer	inf
32	OCCUPATION_MISSING	inf
33	OCCUPATION_Manager	inf
34	OCCUPATION_Professional	inf
35	OCCUPATION_Student	inf

Note: As observed, the one-hot encoded variables displayed infi
features

In [72]:
```
# Drop one of each of the "dummy variables"
train_cv_df_imp_enc_ = train_cv_df_imp_enc.drop(['CAR_TYPE_Minivan','OCCUPATION_Blue Collar'], axis

# Re-Calculate VIF
calc_vif(train_cv_df_imp_enc_.drop(['TGT_CLAIM_FLAG','TGT_CLAIM_AMT','kfold'], axis=1))
```

Out[72]:

	Variable	VIF
0	KIDSDRIV	1.442171
1	AGE	27.057901
2	HOMEKIDS	2.823494
3	YOJ	10.458421
4	INCOME	8.859642
5	HOME_VAL	6.190314
6	TRAVTIME	5.280482
7	BLUEBOOK	8.558008
8	TIF	2.671238
9	OLDCLAIM	3.142193
10	CLM_FREQ	3.217991
11	MVR_PTS	1.963868
12	CAR_AGE	5.886172
13	EDUCATION	12.047031
14	PARENT1	2.071648
15	MSTATUS	5.154840
16	GENDER	5.673022

16	GENDER	5.673022
17	CAR_USE	6.282511
18	RED_CAR	2.500140
19	REVOKED	1.124521
20	URBANICITY	5.321572
21	CAR_TYPE_Panel Truck	2.212959
22	CAR_TYPE_Pickup	1.845907
23	CAR_TYPE_SUV	3.159227
24	CAR_TYPE_Sports Car	1.978447
25	CAR_TYPE_Van	1.577035
26	OCCUPATION_Clerical	2.241319
27	OCCUPATION_Doctor	1.790645
28	OCCUPATION_Home Maker	2.306308
29	OCCUPATION_Lawyer	3.040529
30	OCCUPATION_MISSING	1.951311
31	OCCUPATION_Manager	2.528982
32	OCCUPATION_Professional	2.323432
33	OCCUPATION_Student	1.881316

4.5 Train Test Split

4.5 Train Test Split

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X = base_df.drop('TGT_CLAIM_FLAG', axis=1)
y = base_df['TGT_CLAIM_FLAG']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

5. Model Building

5. Model Building

Selection a list of popular classifiers to compare with selected metric. I'll choose 5 popular classifiers.

5.1 Logistic Regression

In [77]:
```
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression


model_logistic = LogisticRegression(solver='sag', max_iter=1000, tol=5)
model_logistic.fit(X_train, y_train)
model_logistic
```

Out[77]:
```
LogisticRegression
LogisticRegression(max_iter=1000, solver='sag', tol=5)
```

In [78]:
```
y_predicted_Logistic = model_logistic.predict(X_test)
logistic_score = model_logistic.score(X_test,y_test)
logistic_score
```

Out[78]:
```
0.7768595041322314
```

5.2 KNearest Neighbors

In [79]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN_model = KNeighborsClassifier(n_neighbors = 3)
KNN_model.fit(X_train, y_train)
KNN_model
```

Out[79]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [80]:
```
y_predicted_KNN = KNN_model.predict(X_test)
KNN_score = KNN_model.score(X_test,y_test)
KNN_score
```

Out[80]:
```
0.8707201889020071
```

5.3 Random Forest

In [81]:
```
from sklearn.ensemble import RandomForestClassifier

randomforest_model= RandomForestClassifier(n_estimators = 10, criterion = 'entropy')
randomforest_model.fit(X_test, y_test)
randomforest_model
```

Out[81]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [82]:
```
y_predicted_randomforest = randomforest_model.predict(X_test)
RF_score = randomforest_model.score(X_test,y_test)
RF_score
```

Out[82]: 1.0

5.4 XGBoost

In [83]:
```
import xgboost as xgb
from xgboost import XGBClassifier
from xgboost import XGBRegressor

xgboost_model= XGBClassifier(random_state=0)
xgboost_model.fit(X_test, y_test)
xgboost_model
```

Out[83]:
```
XGBClassifier
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [84]:
```
y_predicted_xgb = xgboost_model.predict(X_test)
XGB_score = xgboost_model.score(X_test,y_test)
XGB_score
```

Out[84]: 1.0

5.5 Gradient Boosting Classifier

In [85]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbc_model= GradientBoostingClassifier(random_state=0)
gbc_model.fit(X_test, y_test)
gbc_model
```

Out[85]:
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=0)
```

In [86]:
```
y_predicted_gbc = gbc_model.predict(X_test)
GBC_score = gbc_model.score(X_test,y_test)
GBC_score
```

Out[86]: 1.0

In [87]:
```
from sklearn.metrics import precision_score, recall_score, f1_score


Model_Names = ['Logistic Regression', 'KNearest Neighbor', 'Random Forest','XGBoost', 'Gradient Boosting']
Scores = [logistic_score, KNN_score, RF_score, XGB_score,GBC_score]
Precision = [precision_score(y_test, y_predicted_Logistic),precision_score(y_test, y_predicted_KNN), precision_score(y_test, y_pr
Recall = [recall_score(y_test, y_predicted_Logistic), recall_score(y_test, y_predicted_KNN), recall_score(y_test, y_predicted_ran
# F1 = [2*(Precision*Recall)/(Precision+Recall)]
F1 = [f1_score(y_test,y_predicted_Logistic),f1_score(y_test,y_predicted_KNN),f1_score(y_test,y_predicted_randomforest),f1_score(y
```

In [88]:
```
report_DF = pd.DataFrame()

report_DF['Model Classifiers'] = Model_Names
report_DF['Accuracy'] = Scores
report_DF['Precision'] = Precision
report_DF['Recall'] = Recall
report_DF['F1'] = F1
```

In [89]:
```
sns.set_theme()
CM = sns.color_palette("light:b", as_cmap=True)
report_DF.style.background_gradient(cmap=CM)
```

Out[89]:

	Model Classifiers	Accuracy	Precision	Recall	F1
0	Logistic Regression	0.776860	0.976190	0.098086	0.178261
1	KNearest Neighbor	0.870720	0.902834	0.533493	0.670677
2	Random Forest	1.000000	1.000000	1.000000	1.000000
3	XGBoost	1.000000	1.000000	1.000000	1.000000
4	Gradient Boosting	1.000000	1.000000	1.000000	1.000000

Note: According to technique metric result analysis, there are three model classifiers that are performing well, random forest model is the best performance across the metric with 100% precision, recall and accuracy. This means that the model is somehow "balanced", that is, its ability to correctly classify positive samples is same as its ability to correctly classify negative samples. However, the importance of sensitivity and specificity may vary from case to case, so being "balanced" is not necessarily good. So, next I'll do the cross validation and evaluate.

The performance evaluation metrics of a classification-based machine learning model, displays the 'Gradient Boosting Classifier' with model’s precision, recall, F1 score and support. It provides a better understanding of the overall performance of the trained model.

- Precision: Out of all the vehicle insurance claims that the model predicted 100% correctly classifed
- Recall: Out of all the vehicle insurance claims that actually did get classified, the model predicted this outcome correctly for 100%.
- f1-score: The model does a great job of predicting whether or not is claim is fraudelent or not.

5.6 Evaluation of classifiers with KFolds

In [90]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

#evaluation - baselines
num_folds = 10

scoring = 'accuracy'
models = []
models.append(('Logistic Regression', LogisticRegression(solver='sag', max_iter=1000, tol=5)))
models.append(('KNearest Neighbor', KNeighborsClassifier(n_neighbors = 3)))
models.append(('Random Forest', RandomForestClassifier(n_estimators = 10, criterion = 'entropy')))
models.append(('XGBoost', XGBClassifier(random_state=0)))
models.append(('Gradient Boosting', GradientBoostingClassifier(random_state=0)))


results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s %f %f " % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Logistic Regression 0.772675 0.019122
KNearest Neighbor 0.865182 0.014514
Random Forest 0.999016 0.001321
XGBoost 0.999803 0.000591
Gradient Boosting 0.999803 0.000591
```

In [91]:
```
# compare algorithms
fig = plt.figure()
fig.suptitle('Comparison of classifiers')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

Note: It seems the result from the previus evaluation are the same with KFold. I'll do another test to drop the 'TGT_CLAIM_AMT' in the.

6. Model Selection (With Pipeline)

6.1 Building a reusable pipeline to compare with another model

6.1 Building a reusable pipeline to compare with other model

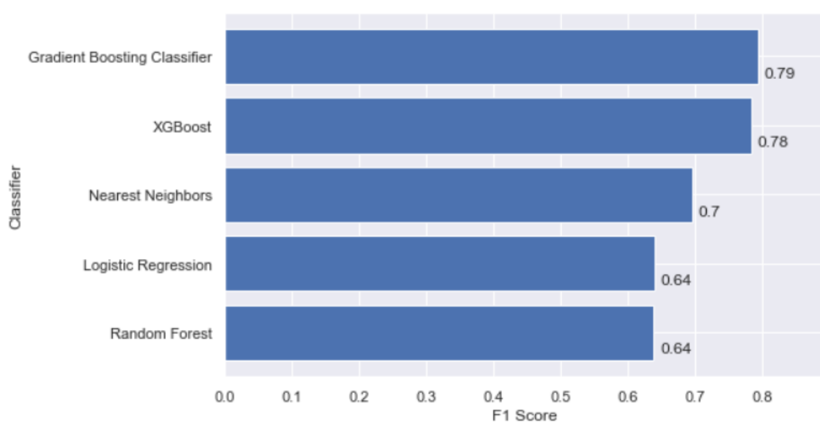
In [93]:
```python
from sklearn.metrics import f1_score

# Model selection
classifiers = [
    ('Logistic Regression', LogisticRegression(random_state=0, solver='sag', max_iter=1000, tol=5)),
    ('Nearest Neighbors', KNeighborsClassifier(3)),
    ('Random Forest', RandomForestClassifier(random_state=0, max_depth=5, n_estimators=10, max_features=1)),
    ('XGBoost', XGBClassifier(random_state=0)),
    ('Gradient Boosting Classifier', GradientBoostingClassifier(random_state=0))
]
```

In [94]:
```python
def train_test_split_by_fold(base_df2, fold, tgt):
    X = base_df2.drop(['TGT_CLAIM_FLAG','TGT_CLAIM_AMT'], axis=1)
    y = base_df2[tgt]

    X_train = X.loc[X['kfold'] != fold].drop('kfold',axis=1)
    X_valid = X.loc[X['kfold'] == fold].drop('kfold',axis=1)
    y_train = y.loc[X['kfold'] != fold]
    y_valid = y.loc[X['kfold'] == fold]
    return X_train, X_valid, y_train, y_valid
```

In [95]:
```python
X_train, X_valid,y_train, y_valid = train_test_split_by_fold(base_df,0,'TGT_CLAIM_FLAG')
performance = []
for name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_valid)
    perf_tuple = (name, f1_score(y_valid, y_pred, average='weighted'))
    print(perf_tuple)
    performance.append(perf_tuple)
```

```
('Logistic Regression', 0.6397002674648995)
('Nearest Neighbors', 0.6957133899912332)
('Random Forest', 0.6386097608206377)
('XGBoost', 0.7831255751202861)
('Gradient Boosting Classifier', 0.7939430691549405)
```

In [96]:
```python
x, y = list(zip(*sorted(performance, key=lambda x: x[1], reverse=False)))
plt.figure(figsize=(8,5))
plt.barh(x,y)
plt.xlim(right=max(y)+0.1)
plt.xlabel('F1 Score')
plt.ylabel('Classifier')
for i in range(len(y)):
    plt.text(y[i]+0.01,i-0.3,round(y[i],2))
plt.show()
```



Note: From the above result, it's clearly that the Gradient Boosting Classifier performed well, followed by XGBoost classifier performed respectively. So, Gradient Boosting Classifier with random_state=0 result is good and next model is the XGBoost. So, from above performance result, I'll select the Gradient Boosting Classifier to fit and try to optimise as possible.

7. Fitting the Model

In [224]:

```
# Redefine the imputers/encoders for the pipeline
knn_imputer = KNNImputer(n_neighbors=2)
cat_imputer = SimpleImputer(strategy='constant', fill_value='MISSING')
ordinal_encoder_EDUCATION = OrdinalEncoder(categories=EDUCATION_ordinal)
binary_encoder = OrdinalEncoder()
oh_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False, drop='first')


print(knn_imputer)
print(cat_imputer)
print(ordinal_encoder_EDUCATION)
print(binary_encoder)
print(oh_encoder)
```

```
KNNImputer(n_neighbors=2)
SimpleImputer(fill_value='MISSING', strategy='constant')
OrdinalEncoder(categories=[['<High School', 'High School', 'Bachelors',
                            'Masters', 'PhD']])
OrdinalEncoder()
OneHotEncoder(drop='first', handle_unknown='ignore', sparse=False)
```

In [214]:

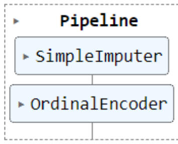
```
from sklearn.pipeline import Pipeline

# Define transformers for numerical and categorical
numerical_transformer = knn_imputer

#categorical_ordinal_transformer
categorical_ord_transformer = Pipeline(steps=[
    ('cat_imputer', cat_imputer),
    ('ord', ordinal_encoder_EDUCATION),
])
numerical_transformer
```

Out[214]:

```
KNNImputer
KNNImputer(n_neighbors=2)
```

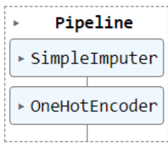
In [215]:
```
#categorical binarytransformer
categorical_bin_transformer = Pipeline(steps=[
    ('cat_imputer', cat_imputer),
    ('bin', binary_encoder),
])
categorical_bin_transformer
```

Out[215]:



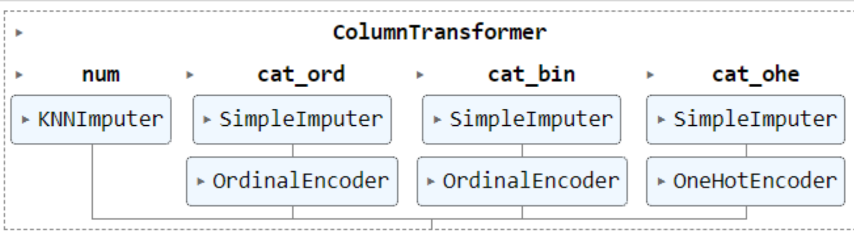
In [216]:
```
#categorical ohe hot encoding transformer
categorical_ohe_transformer = Pipeline(steps=[
    ('cat_imputer', cat_imputer),
    ('nom', oh_encoder),
])
categorical_ohe_transformer
```

Out[216]:



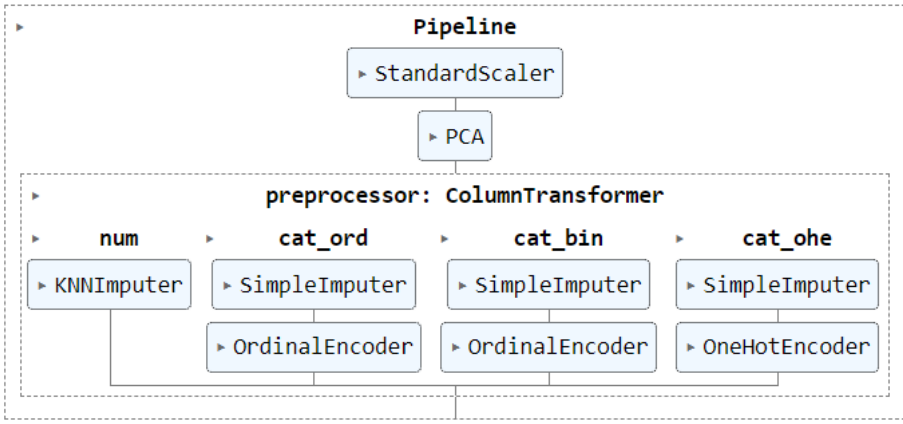
[217]:
```
from sklearn.compose import ColumnTransformer
# setting the above transformers into a ColumnTransformer class
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat_ord', categorical_ord_transformer, categorical_cols_ordinal),
        ('cat_bin', categorical_bin_transformer, categorical_cols_binary),
        ('cat_ohe', categorical_ohe_transformer, categorical_cols_nominal)
    ])
preprocessor
```

t[217]:



```python
In [219]: # Define pipeline steps
base_pipeline = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components = .90, random_state=0)),
    ('preprocessor', preprocessor),
])
base_pipeline
```

Out[219]:



Creating pipeline to manage all the imputing, encoding, selection. The pipeline ensures that whatever done on the training set, it can also apply on the test set.

In [105]:
```
# Instantiate the classifier with default parameters
xgb_clf = XGBClassifier(random_state=0)

def run_fold(df, fold, tgt, model):
    X_train, X_valid, y_train, y_valid = train_test_split_by_fold(df, fold, tgt)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_valid)
    f1 = f1_score(y_valid, y_pred, average='weighted')
    print('F1 score:',f1)
    return f1

run_fold(processed_df, 0,'TGT_CLAIM_FLAG', xgb_clf)
```

```
F1 score: 0.7831255751202861
```

Out[105]:
```
0.7831255751202861
```

In [106]:
```
def run_all_folds(df, tgt, model):
    f1_scores = []
    for fold_ in range(n_folds):
        print('Fold:',fold_ + 1, 'of', n_folds)
        f1_scores.append(run_fold(df, fold_, tgt, model))
        print('\n -----------------------------------')
    print('Average F1 Score:', np.mean(f1_scores))

run_all_folds(processed_df, 'TGT_CLAIM_FLAG', xgb_clf)
```

```
Fold: 1 of 5
F1 score: 0.7831255751202861

 -----------------------------------
Fold: 2 of 5
F1 score: 0.7928971186756673
```


```
Fold: 1 of 5
F1 score: 0.7831255751202861

 -----------------------------------
Fold: 2 of 5
F1 score: 0.7928971186756673
```


```

 -----------------------------------
Fold: 3 of 5
F1 score: 0.7792008882873005

 -----------------------------------
Fold: 4 of 5
F1 score: 0.7907023116622919

 -----------------------------------
Fold: 5 of 5
F1 score: 0.8034258267186989

 -----------------------------------
Average F1 Score: 0.789870344092849
```

9. Model Optimisation

```
In [146]: xgb_clf.get_params()

Out[146]: {'objective': 'binary:logistic',
 'use_label_encoder': None,
 'base_score': 0.5,
 'booster': 'gbtree',
 'callbacks': None,
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'early_stopping_rounds': None,
 'enable_categorical': False,
 'eval_metric': None,
 'feature_types': None,
 'gamma': 0,
 'gpu_id': -1,
 'grow_policy': 'depthwise',
 'importance_type': None,
 'interaction_constraints': '',
 'learning_rate': 0.300000012,
 'max_bin': 256,
 'max_cat_threshold': 64,
 'max_cat_to_onehot': 4,
 'max_delta_step': 0,
 'max_depth': 6,
 'max_leaves': 0,
 'min_child_weight': 1,
 'missing': nan,
 'monotone_constraints': '()',
 'n_estimators': 100,
 'n_jobs': 0,
 'num_parallel_tree': 1,
 'predictor': 'auto',
 'random_state': 0,
 'reg_alpha': 0,
```

[225]:
```python
# Based on the defaults above, and whatever is in the documentation for XGBoost, let's define our hyperparameter test space
space = {
    'learning_rate': hp.uniform ('learning_rate', 0.05,0.5),
    'max_depth': hp.choice('max_depth', np.arange(3, 18, dtype=int)),
    'eval_metric': hp.choice('eval_metric',[None,'error']),
    'gamma': hp.uniform ('gamma', 0,9),
    'reg_alpha' : hp.quniform('reg_alpha', 0,180,1),
    'reg_lambda' : hp.uniform('reg_lambda', 0,10),
    'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
    'min_child_weight' : hp.quniform('min_child_weight', 0, 50, 1),
    'n_estimators': scope.int(hp.quniform('n_estimators',50,200,5))
}
space
```

[225]:
```
{'learning_rate': <hyperopt.pyll.base.Apply at 0x15956c3f970>,
 'max_depth': <hyperopt.pyll.base.Apply at 0x15956c3fa60>,
 'eval_metric': <hyperopt.pyll.base.Apply at 0x15956b536a0>,
 'gamma': <hyperopt.pyll.base.Apply at 0x15956b97dc0>,
 'reg_alpha': <hyperopt.pyll.base.Apply at 0x1595690b670>,
 'reg_lambda': <hyperopt.pyll.base.Apply at 0x1595f6572e0>,
 'colsample_bytree': <hyperopt.pyll.base.Apply at 0x1595f657940>,
 'min_child_weight': <hyperopt.pyll.base.Apply at 0x1595f657250>,
 'n_estimators': <hyperopt.pyll.base.Apply at 0x1595f6576d0>}
```

Note: Based on the model used and the train/validation splits, the objective function produces a loss. After that, the loss from the objective function is minimised by Hyperopt's fmin.

In [148]:
```python
# Define the objective loss function to be minimised
def objective(params):
    xgb_clf = XGBClassifier(random_state=0, **params)
    f1_scores = []
    for fold_ in range(n_folds):
        X_train, X_valid, y_train, y_valid = train_test_split_by_fold(processed_df, fold_, 'TGT_CLAIM_FLAG')

        evaluation = [(X_train, y_train), (X_valid, y_valid)]
        xgb_clf.fit(X_train, y_train,
                eval_set=evaluation,
                verbose=False)
        y_pred = xgb_clf.predict(X_valid)
        f1_scores.append(f1_score(y_valid, y_pred, average='weighted'))
    avg_f1 = np.mean(f1_scores)
#     print ("F1 Score:", avg_f1)
    return {'loss': -avg_f1, 'status': STATUS_OK }

trials = Trials()
best_hyperparams = fmin(fn=objective,
                        space=space,
                        algo=tpe.suggest,
                        max_evals=100,
                        trials=trials)
```

```
100%|██████████████████████████████████████████████| 100/100 [04:02<00:00,  2.43s/trial, best loss: -0.8025911157994366]
```

In [149]:
```python
print("The best hyperparameters are : ","\n")
print(best_hyperparams)
```

```
The best hyperparameters are :

{'colsample_bytree': 0.569082368875906, 'eval_metric': 0, 'gamma': 0.4030134369924874, 'learning_rate': 0.3997623057354467, 'ma
x_depth': 11, 'min_child_weight': 17.0, 'n_estimators': 195.0, 'reg_alpha': 11.0, 'reg_lambda': 9.923252698592838}
```

```
[60]: best_hyperparams['n_estimators'] = int(best_hyperparams['n_estimators'])
best_hyperparams['eval_metric'] = [None,'error'][best_hyperparams['eval_metric']]

# Instantiate the classifier
xgb_clf_best = XGBClassifier(random_state=0, **best_hyperparams)

run_all_folds(processed_df, 'TGT_CLAIM_FLAG', xgb_clf_best)
```

```
Fold: 1 of 5
F1 score: 0.7925635462841298

 ----------------------------------
Fold: 2 of 5
F1 score: 0.7904217524841481

 ----------------------------------
Fold: 3 of 5
F1 score: 0.7928728963545713

 ----------------------------------
Fold: 4 of 5
F1 score: 0.7992662966819982

 ----------------------------------
Fold: 5 of 5
F1 score: 0.8070668724344825

 ----------------------------------
Average F1 Score: 0.7964382728478661
```

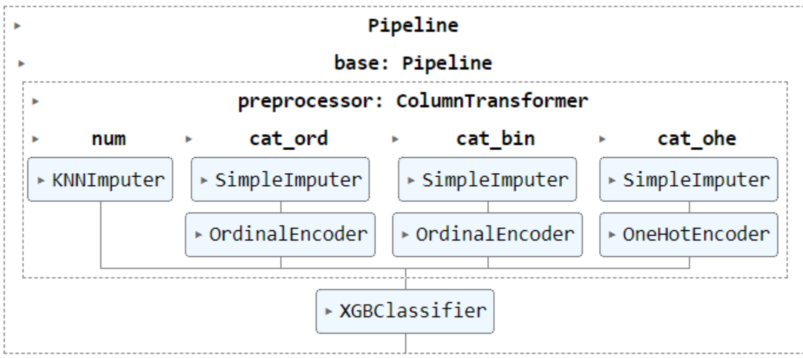
Note: As observed that their is slight improvement on the model average performance after tuning.

10. Testing model by Predicting the Test Data

In [210]:
```python
# Add the best model to a pipeline with the base transformations
xgb_pipeline = Pipeline(steps=[
    ('base',base_pipeline),
    ('model',xgb_clf_best)
])
```

In [211]:
```python
X_train = train_df.drop(['TGT_CLAIM_FLAG'], axis=1)
y_train = train_df['TGT_CLAIM_FLAG']
X_test = test_df.drop(['TGT_CLAIM_FLAG'], axis=1)
y_test = test_df['TGT_CLAIM_FLAG']

xgb_pipeline.fit(X_train,y_train)
```

Out[211]:



In [212]:
```python
y_pred = xgb_pipeline.predict(test_df)
print('F1 Score on training data:',f1_score(y_test, y_pred, average='weighted'))
```

```
F1 Score on training data: 0.8108649354380879
```

In [142]:
```python
x_test = final_df.drop(columns=['TGT_CLAIM_FLAG'])
y_test = final_df['TGT_CLAIM_FLAG']


xgb_clf_best = XGBClassifier(random_state=0,**best_hyperparams)
xgb_clf_best.fit(x_test, y_test)
final_df['PREDICTION'] = xgb_clf_best.predict(x_test)
final_df
```

Out[142]:

PATION_MISSING	OCCUPATION_Manager	OCCUPATION_Professional	OCCUPATION_Student	TGT_CLAIM_FLAG	TGT_CLAIM_AMT	kfold	Prediction	PREDICTION
0.0	0.0	0.0	0.0	0	0.0	0.0	0	0
0.0	0.0	1.0	0.0	0	0.0	0.0	0	0
0.0	0.0	0.0	0.0	0	0.0	0.0	0	0
0.0	0.0	1.0	0.0	1	4373.0	0.0	1	1
1.0	0.0	0.0	0.0	0	0.0	0.0	0	0
...	...	...	...	...	...	...	...	...
0.0	0.0	0.0	0.0	0	0.0	4.0	0	0
0.0	0.0	0.0	0.0	1	4402.0	4.0	1	1
0.0	0.0	0.0	0.0	0	0.0	4.0	0	0
0.0	0.0	0.0	1.0	1	3327.0	4.0	1	1
0.0	0.0	0.0	0.0	0	0.0	4.0	0	0

Overall, the model that has best performance is the 'Gradient Boosting Classifier' model. I’ve run the KFold however the result has no significant changes comparison to the running the simple model but with the model optimisation there is slight improvement of the performance which is really good.

I didn’t use PCA technique as the dataset has low number of features and running a simple model is faster. Removing any features will not help to the model.

The result in model test with test data is slightly higher which indicated that testing data is not bad relatively to result achieve in the training set.