BZAN 6357 - Final Project - Team 2

For this projects purpose we have used Python Version 3.11.2

Introduction

In the insurance industry, being able to predict if a customer has a potiential to buy a policy can be very cost effective to the company and can increase the profits as it less communication and time are required to acquire a deal.

objective

Our objective in this project is to train ML models on historical data about the customers and try to predict if customers will buy another kind of policy, the thing that will help in generating leads to contact for sale.

Process Summary

1. Get a feel of the data provided.
2. Check the data for missing values.
3. Do EDA on the dataset to know more about the data provided and the relations between the variables.
4. Chose what to do with each input varible.
5. Train a Neural network model on the prepared data.
6. Tweak the model parameters to improve performance.
7. Do the prediction using the chosen model.

Code

```
#Importing Libraries and Setting a random state:
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from statsmodels.stats.proportion import proportions_ztest
from sklearn.preprocessing import MinMaxScaler,
OneHotEncoder,FunctionTransformer
from sklearn.compose import make_column_transformer
import numpy.random as random
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from time import time

import tensorflow as tf
from tensorflow import keras

%matplotlib inline
sns.set_style('ticks')

np.random.seed(10)
random_state = 10

tf.random.set_seed(random_state)
tf.keras.utils.set_random_seed(random_state)
```

1. Data Managment

```
# Reading the training dataset:

Training_Data = pd.read_csv("bzan6357_insurance_3_TRAINING.csv")

Training_Data.head(5)
```

```
      id_new  buy  age  gender  tenure  region  dl  has_v_insurance
v_age  \
0  a00000000    0   34  female      31      19   1                0
1-2 year
1  a00000001    0   50    male     211      34   1                0
1-2 year
2  a00000002    0   42  female     122      29   1                0
1-2 year
3  a00000003    0   28    male      75       3   1                0
1-2 year
4  a00000004    0   75  female      19      28   1                0
1-2 year

  v_accident  v_prem_quote  cs_rep
0        yes       27715.0     154
1        yes       33945.0     154
2        yes       37577.0     163
3        yes        2630.0     154
4        yes       47511.0     122
```

```
# Checking the shape of the dataset:
Training_Data.shape
```

```
(20460, 12)
```

```
# Checking variables data types
Training_Data.dtypes
```

```
id_new                object
buy                    int64
age                    int64
gender                object
tenure                 int64
region                 int64
dl                     int64
has_v_insurance        int64
v_age                 object
v_accident            object
v_prem_quote         float64
cs_rep                 int64
dtype: object
```

```
# Checking for missing values in the training data:
Training_Data.isnull().sum().sum()
```

```
0
```

From the data dictionary we found many numeric data types that have discrete or binary values, which needs to be converted

We found the region variable to have 53 different regions:

```
len(Training_Data['region'].value_counts())
```

```
53
```

We found the vehicle age variable to have 3 different categories:

```
len(Training_Data['v_age'].value_counts())
```

```
3
```

We found the customer representative variable to have 4 different representatives:

```
len(Training_Data['cs_rep'].value_counts())
```

```
4
```

We found the age variable to have 65 different ages:

```
len(Training_Data['age'].value_counts())
```

```
65
```

Checking the binary variables for errors : we found all the binary variables to have 2 different categories

```
len(Training_Data['gender'].value_counts())
```

```
2
```

```
len(Training_Data['dl'].value_counts())
```

```
2
```

```
len(Training_Data['has_v_insurance'].value_counts())
```

```
2
```

```
len(Training_Data['v_accident'].value_counts())
```

```
2
```

2. EDA

```
# Check the structure and size of the dataset
print("Dataset shape:", Training_Data.shape)
print("Dataset information:")
print(Training_Data.info())
```

```
Dataset shape: (20460, 12)
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20460 entries, 0 to 20459
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id_new           20460 non-null  object 
 1   buy              20460 non-null  int64  
 2   age              20460 non-null  int64  
 3   gender           20460 non-null  object 
 4   tenure           20460 non-null  int64  
 5   region           20460 non-null  int64  
 6   dl               20460 non-null  int64  
 7   has_v_insurance  20460 non-null  int64  
 8   v_age            20460 non-null  object 
 9   v_accident       20460 non-null  object 
 10  v_prem_quote     20460 non-null  float64
 11  cs_rep           20460 non-null  int64  
dtypes: float64(1), int64(7), object(4)
memory usage: 1.9+ MB
None
```

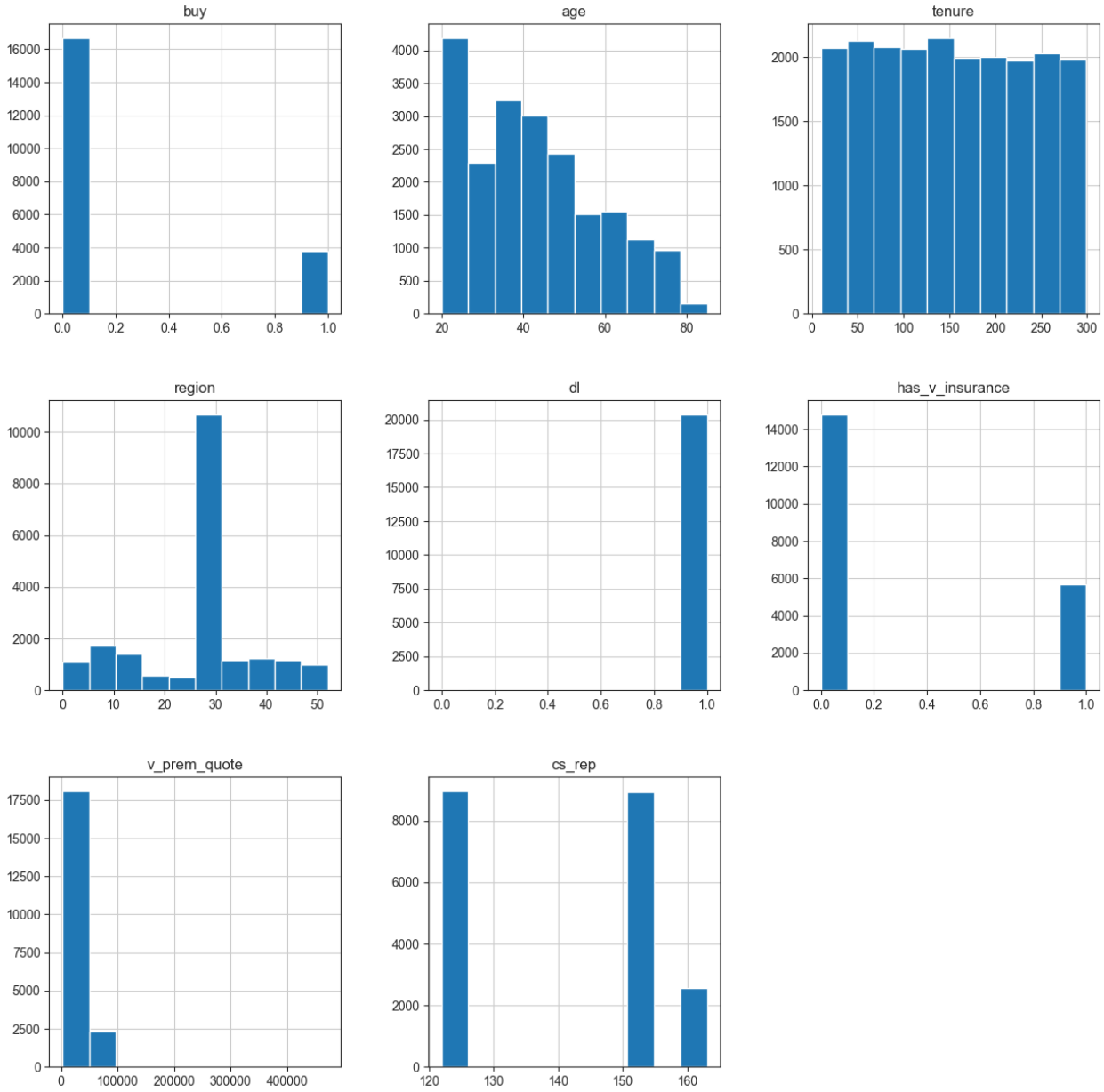
```
# Check for missing values
print("Number of missing values in each column:")
print(Training_Data.isna().sum())
```

```
Number of missing values in each column:
id_new             0
buy                0
age                0
gender             0
tenure             0
region             0
dl                 0
has_v_insurance    0
v_age              0
v_accident         0
v_prem_quote       0
cs_rep             0
dtype: int64
```

```
# Check the distribution and summary statistics of the variables
print("Summary statistics of numerical variables:")
print(Training_Data.describe())
Training_Data.hist(figsize=(15,15))
```

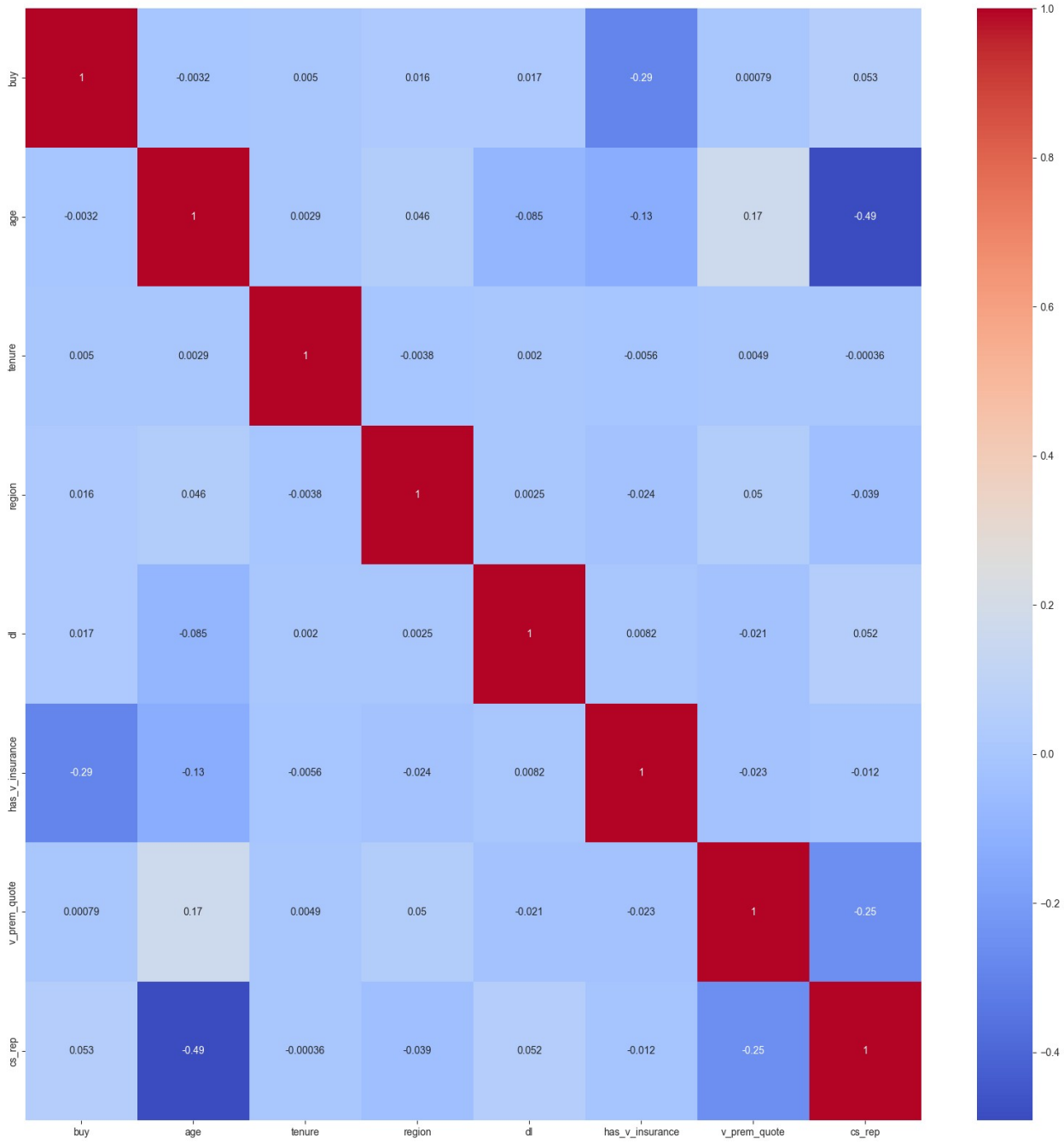
```
Summary statistics of numerical variables:
                buy           age        tenure         region
dl  \
count  20460.000000  20460.000000  20460.000000  20460.000000
20460.000000
mean       0.183529     42.512317    152.908016     26.624878
0.996970
std        0.387109     15.615623     83.483286     11.568606
0.054966
min        0.000000     20.000000     10.000000      0.000000
0.000000
25%        0.000000     29.000000     81.000000     24.000000
1.000000
50%        0.000000     40.000000    151.000000     28.000000
1.000000
75%        0.000000     53.000000    225.250000     29.000000
1.000000
max        1.000000     85.000000    299.000000     52.000000
1.000000

       has_v_insurance   v_prem_quote        cs_rep
count     20460.000000   20460.000000  20460.000000
mean          0.276540   31689.197361    140.588759
std           0.447298   18757.868023     16.749921
min           0.000000    2630.000000    122.000000
25%           0.000000   24731.750000    122.000000
50%           0.000000   32533.500000    151.000000
75%           1.000000   41294.250000    154.000000
max           1.000000  472042.000000    163.000000
```

```
array([[<Axes: title={'center': 'buy'}>, <Axes: title={'center':
'age'}>,
        <Axes: title={'center': 'tenure'}>],
       [<Axes: title={'center': 'region'}>,
        <Axes: title={'center': 'dl'}>,
        <Axes: title={'center': 'has_v_insurance'}>],
       [<Axes: title={'center': 'v_prem_quote'}>,
        <Axes: title={'center': 'cs_rep'}>, <Axes: >]], dtype=object)
```




```
# Create a heatmap to visualize correlations
plt.figure(figsize=(20, 20))
sns.heatmap(Training_Data.corr(), annot=True, cmap='coolwarm')
```

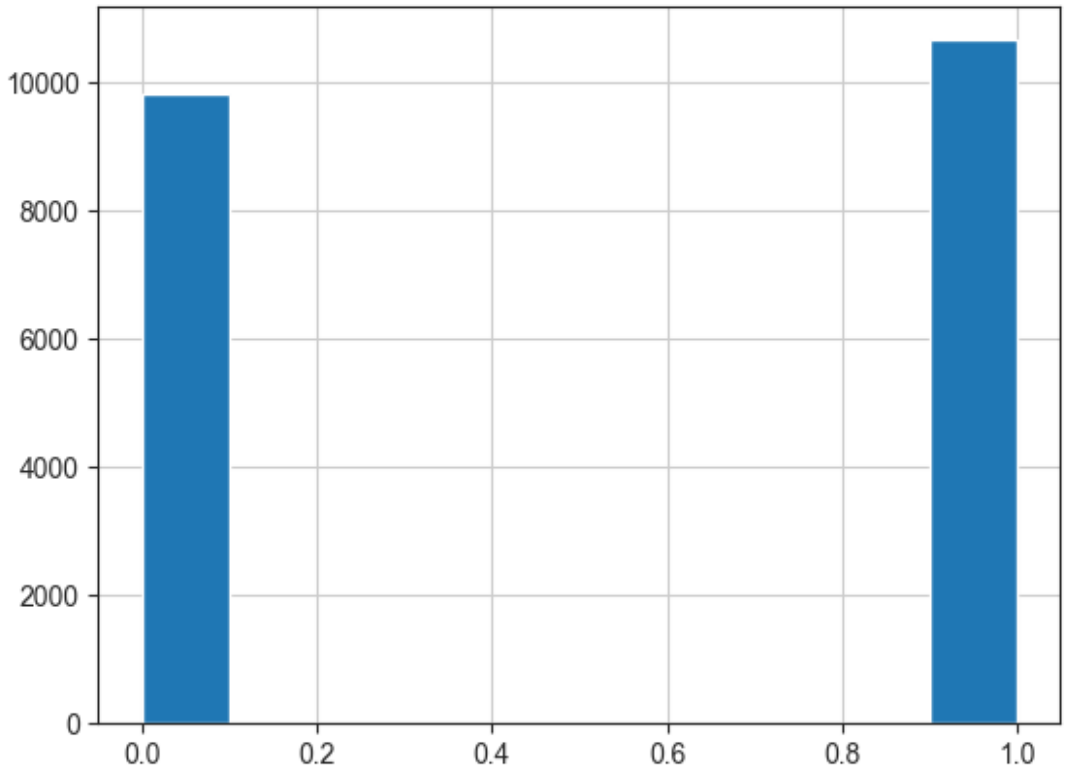
```
C:\Users\hania\AppData\Local\Temp\ipykernel_12972\2270168611.py:3:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this
warning.
  sns.heatmap(Training_Data.corr(), annot=True, cmap='coolwarm')

<Axes: >
```



From the heatmap above we don't find any linearly correlated variables, no variable should be excluded from the model, but for thr region variable we need to partion it into majority class (regions between 25 and 30), and rest of the regions into minority class as it has many discrete unrelated numeric values.

```
#For the Region variable we need to turn it into majority and minority
classes using a binary codeing:
Region_binary = []
for i in Training_Data['region']:
    if i >= 25 and i <=30:
        Region_binary.append(1);
    else:
        Region_binary.append(0) ;

# Droping the original region variable:
Training_Data = Training_Data.drop("region", axis =1)

#Adding the new transformed region binary variable:
Training_Data['Region_binary'] = Region_binary
Training_Data['Region_binary'].hist()
```

```
<Axes: >
```



Performing train-test split

```
X = Training_Data.drop(columns=['id_new','buy'])
y = Training_Data['buy']


y.value_counts()
```

```
0    16705
1     3755
Name: buy, dtype: int64
```

We can see a big imbalance between classes in the target variable, this imbalance will be accounted for in the model by providing different weights to classes in the model.

```
# Proportions Probabilities of y:
y_1 = y.value_counts()[1]/(y.value_counts()[0]+y.value_counts()[1])
y_0 = 1- y_1
print(y_0,y_1)
```

```
0.8164711632453567 0.1835288367546432
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=random_state, shuffle=True)
```

Evaluate validity of split using hypothesis-testing

Question: Are the values of the train and test sets of the buy variable of equal proportions.

H0: Mean_train = Mean_test , Alpha = 0.05

```
count_default_train = y_train.value_counts()[1]
count_default_test = y_test.value_counts()[1]
z_stat, p = proportions_ztest(count=[count_default_train,
count_default_test], nobs=[y_train.shape[0], y_test.shape[0]])
print('z-stat = {:0.3f}, p = {:0.3f}'.format(z_stat, p))
```

```
z-stat = 0.774, p = 0.439
```

```
print('''Since the p-value of the z-test is equal to ''' +
str(p.round(4)) + ''' which is larger than 0.05 significance level
then when we can
accept the null hypothisis that the Proportion_train = Proportion_test
which means that there is no significant difference
between the proportions of the training and test samples of the buy
variable''')
```

```
Since the p-value of the z-test is equal to 0.4388 which is larger
than 0.05 significance level then when we can
accept the null hypothisis that the Proportion_train = Proportion_test
which means that there is no significant difference
```

```
between the proportions of the training and test samples of the buy variable
```

Transforming the variables

For the y set no preprocessing is needed.

For the X set here is What we chose to do with each variable:

1. buy: Keep as it is.
2. age: Keep it numeric as it is since it has many different values and perform log tansformation then Standarization.
3. gender: Convert the values to zeros and ones using OneHotEncoder.
4. tenure: Perform Standarization.
5. region: Partion it into majority class (regions between 25 and 30), and rest of the regions into minority class.
6. dl: Keep as it is.
7. has_v_insurance: Keep as it is.
8. v_age: Convert to flag variables.
9. v_accident: Convert to zeros and ones using OneHotEncoder.
10. v_prem_quote: Perform log tansformation then Standarization.
11. cs_rep: Convert to flag variables.

```
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

#OneHotEncoder
onehot_encoder = OneHotEncoder()

# prepare list of column labels for one-hot-encoder
cols_onehot = ['gender', 'v_age', 'v_accident','cs_rep']

# prepare list of column labels for Standarization:
cols_mmn_scale = ['tenure']

# prepare list of column labels for log transformation:
log_cols = ['age','v_prem_quote']

# define identity function:
Identity = FunctionTransformer(feature_names_out= "one-to-one")

# prepare list of column labels for keeping the variables as they are:
As_is_cols = ['dl','has_v_insurance','Region_binary']


X_train
```

```
       age  gender  tenure  dl  has_v_insurance      v_age v_accident
\
15747   40    male     296   1                0   1-2 year        yes

6735    50  female      64   1                0   1-2 year        yes

15307   64    male     211   1                0  > 2 years        yes

10898   33  female      24   1                1   1-2 year         no

245     78    male     227   1                0  > 2 years        yes

...    ...     ...     ...  ..              ...        ...        ...

9372    23    male     197   1                1   < 1 year         no

7291    34    male     161   1                0   1-2 year        yes

17728   34    male     102   1                1   1-2 year         no

7293    49    male     149   1                0   1-2 year        yes

17673   23    male     282   1                1   < 1 year         no


       v_prem_quote  cs_rep  Region_binary
15747       34277.0     154              1
6735        59184.0     154              0
15307       37356.0     122              0
10898       25416.0     122              1
245         38041.0     122              1
...             ...     ...            ...
9372        23666.0     151              0
7291        45021.0     154              0
17728       29472.0     151              0
7293        42694.0     163              1
17673       31218.0     151              0

[12276 rows x 10 columns]
```

```
# Defining a pipeline 1 for Imputation and Standarization:
Pipe1_transformer = make_pipeline(SimpleImputer(strategy =
'most_frequent'), preprocessing.StandardScaler())

# Defining a pipeline 2 for Imputation, Log transformation and
Standarization:
Pipe2_transformer = make_pipeline(SimpleImputer(strategy = 'median'),

preprocessing.FunctionTransformer(np.log1p, feature_names_out= "one-
to-one"),
                                  preprocessing.StandardScaler())
```

```
# Defineing a column tranformer:
transformer = make_column_transformer((onehot_encoder, cols_onehot),
(Pipe2_transformer,log_cols),
                                      (Pipe1_transformer,
cols_mmn_scale),(Identity,As_is_cols))


# use fit_transform() to fit train set X vars
X_train_processed =
pd.DataFrame(transformer.fit_transform(X_train),columns =
transformer.get_feature_names_out())


# only use transform() on the test set X vars
X_test_processed = pd.DataFrame(transformer.transform(X_test),columns
= transformer.get_feature_names_out(),
                                index = X_test.index)

display(X_train_processed,X_test_processed)
```

	onehotencoder__gender_female	onehotencoder__gender_male	\
0	0.0	1.0	
1	1.0	0.0	
2	0.0	1.0	
3	1.0	0.0	
4	0.0	1.0	
...	...	...	
12271	0.0	1.0	
12272	0.0	1.0	
12273	0.0	1.0	
12274	0.0	1.0	
12275	0.0	1.0	

	onehotencoder__v_age_1-2 year	onehotencoder__v_age_< 1 year	\
0	1.0	0.0	
1	1.0	0.0	
2	0.0	0.0	
3	1.0	0.0	
4	0.0	0.0	
...	...	...	
12271	0.0	1.0	
12272	1.0	0.0	
12273	1.0	0.0	
12274	1.0	0.0	
12275	0.0	1.0	

	onehotencoder__v_age_> 2 years	onehotencoder__v_accident_no	\
0	0.0	0.0	
1	0.0	0.0	

```
2                                   1.0                                   0.0
3                                   0.0                                   1.0
4                                   1.0                                   0.0
...                                 ...                                   ...
12271                               0.0                                   1.0
12272                               0.0                                   0.0
12273                               0.0                                   1.0
12274                               0.0                                   0.0
12275                               0.0                                   1.0

       onehotencoder__v_accident_yes  onehotencoder__cs_rep_122  \
0                                1.0                        0.0
1                                1.0                        0.0
2                                1.0                        1.0
3                                0.0                        1.0
4                                1.0                        1.0
...                              ...                        ...
12271                            0.0                        0.0
12272                            1.0                        0.0
12273                            0.0                        0.0
12274                            1.0                        0.0
12275                            0.0                        0.0

       onehotencoder__cs_rep_151  onehotencoder__cs_rep_154  \
0                            0.0                        1.0
1                            0.0                        1.0
2                            0.0                        0.0
3                            0.0                        0.0
4                            0.0                        0.0
...                          ...                        ...
12271                        1.0                        0.0
12272                        0.0                        1.0
12273                        1.0                        0.0
12274                        0.0                        0.0
12275                        1.0                        0.0

       onehotencoder__cs_rep_163  pipeline-1__age  pipeline-
1__v_prem_quote  \
0                            0.0         0.012771
0.391410
1                            0.0         0.616787
0.920892
2                            0.0         1.288076
0.474799
3                            0.0        -0.505337
0.101460
4                            0.0         1.827906
0.492415
...                          ...              ...
...
```

```
12271                           0.0        -1.469275
0.032303
12272                           0.0        -0.425114
0.655730
12273                           0.0        -0.425114
0.244994
12274                           1.0         0.561983
0.604281
12275                           0.0        -1.469275
0.300789

       pipeline-2__tenure  functiontransformer__dl  \
0                1.728158                      1.0
1               -1.060089                      1.0
2                0.706602                      1.0
3               -1.540821                      1.0
4                0.898895                      1.0
...                   ...                      ...
12271            0.538346                      1.0
12272            0.105687                      1.0
12273           -0.603393                      1.0
12274           -0.038533                      1.0
12275            1.559902                      1.0

       functiontransformer__has_v_insurance  \
0                                       0.0
1                                       0.0
2                                       0.0
3                                       1.0
4                                       0.0
...                                     ...
12271                                   1.0
12272                                   0.0
12273                                   1.0
12274                                   0.0
12275                                   1.0

       functiontransformer__Region_binary
0                                     1.0
1                                     0.0
2                                     0.0
3                                     1.0
4                                     1.0
...                                   ...
12271                                 0.0
12272                                 0.0
12273                                 0.0
12274                                 1.0
12275                                 0.0
```

[12276 rows x 17 columns]

	onehotencoder__gender_female	onehotencoder__gender_male	\
11366	1.0	0.0	
20403	0.0	1.0	
13929	1.0	0.0	
10712	0.0	1.0	
3772	1.0	0.0	
...	...	...	
5471	0.0	1.0	
7821	1.0	0.0	
20406	1.0	0.0	
20360	1.0	0.0	
2967	1.0	0.0	

	onehotencoder__v_age_1-2 year	onehotencoder__v_age_< 1 year	\
11366	0.0	0.0	
20403	1.0	0.0	
13929	1.0	0.0	
10712	1.0	0.0	
3772	0.0	1.0	
...	...	...	
5471	1.0	0.0	
7821	0.0	1.0	
20406	1.0	0.0	
20360	0.0	1.0	
2967	0.0	1.0	

	onehotencoder__v_age_> 2 years	onehotencoder__v_accident_no	\
11366	1.0	0.0	
20403	0.0	0.0	
13929	0.0	0.0	
10712	0.0	0.0	
3772	0.0	1.0	
...	...	...	
5471	0.0	0.0	
7821	0.0	1.0	
20406	0.0	0.0	
20360	0.0	1.0	
2967	0.0	0.0	

	onehotencoder__v_accident_yes	onehotencoder__cs_rep_122	\
11366	1.0	0.0	
20403	1.0	0.0	
13929	1.0	1.0	
10712	1.0	1.0	
3772	0.0	0.0	
...	...	...	
5471	1.0	1.0	

```
7821                                0.0                           0.0
20406                               1.0                           0.0
20360                               0.0                           0.0
2967                                1.0                           0.0

       onehotencoder__cs_rep_151  onehotencoder__cs_rep_154  \
11366                        0.0                        1.0
20403                        0.0                        0.0
13929                        0.0                        0.0
10712                        0.0                        0.0
3772                         1.0                        0.0
...                          ...                        ...
5471                         0.0                        0.0
7821                         1.0                        0.0
20406                        0.0                        0.0
20360                        1.0                        0.0
2967                         1.0                        0.0

       onehotencoder__cs_rep_163  pipeline-1__age  pipeline-
1__v_prem_quote  \
11366                        0.0         1.493170
0.618437
20403                        1.0         0.670527                   -
2.097317
13929                        0.0        -0.271324
0.350601
10712                        0.0         0.825754                   -
2.097317
3772                         0.0        -1.469275                   -
2.097317
...                          ...              ...
...
5471                         0.0         0.616787
0.818195
7821                         0.0        -1.356300
0.606459
20406                        1.0        -0.055566                   -
2.097317
20360                        0.0        -0.945548
0.408733
2967                         0.0        -1.356300                   -
2.097317

       pipeline-2__tenure  functiontransformer__dl  \
11366           -0.242844                      1.0
20403           -1.204308                      1.0
13929            0.057614                      1.0
10712            0.946968                      1.0
3772             0.093669                      1.0
...                   ...                      ...
```

```
5471            -1.372565                       1.0
7821            -0.447155                       1.0
20406           -0.675503                       1.0
20360            0.742657                       1.0
2967             0.490273                       1.0

       functiontransformer__has_v_insurance  \
11366                                   0.0
20403                                   0.0
13929                                   0.0
10712                                   0.0
3772                                    1.0
...                                     ...
5471                                    0.0
7821                                    1.0
20406                                   0.0
20360                                   1.0
2967                                    0.0

       functiontransformer__Region_binary
11366                                 1.0
20403                                 0.0
13929                                 0.0
10712                                 0.0
3772                                  0.0
...                                   ...
5471                                  1.0
7821                                  0.0
20406                                 0.0
20360                                 0.0
2967                                  0.0

[8184 rows x 17 columns]
```

```
X_train_processed.describe()
```

```
       onehotencoder__gender_female  onehotencoder__gender_male  \
count                   12276.00000                 12276.00000
mean                        0.44811                     0.55189
std                         0.49732                     0.49732
min                         0.00000                     0.00000
25%                         0.00000                     0.00000
50%                         0.00000                     1.00000
75%                         1.00000                     1.00000
max                         1.00000                     1.00000

       onehotencoder__v_age_1-2 year  onehotencoder__v_age_< 1 year  \
count                   12276.000000                   12276.000000
mean                        0.680026                       0.250733
std                         0.466485                       0.433453
```

```
min                            0.000000                          0.000000
25%                            0.000000                          0.000000
50%                            1.000000                          0.000000
75%                            1.000000                          1.000000
max                            1.000000                          1.000000

       onehotencoder__v_age_> 2 years  onehotencoder__v_accident_no  \
count                     12276.000000                  12276.000000
mean                          0.069241                      0.312154
std                           0.253874                      0.463391
min                           0.000000                      0.000000
25%                           0.000000                      0.000000
50%                           0.000000                      0.000000
75%                           0.000000                      1.000000
max                           1.000000                      1.000000

       onehotencoder__v_accident_yes  onehotencoder__cs_rep_122  \
count                   12276.000000               12276.000000
mean                        0.687846                   0.443304
std                         0.463391                   0.496795
min                         0.000000                   0.000000
25%                         0.000000                   0.000000
50%                         1.000000                   0.000000
75%                         1.000000                   1.000000
max                         1.000000                   1.000000

       onehotencoder__cs_rep_151  onehotencoder__cs_rep_154  \
count               12276.000000               12276.000000
mean                    0.170414                   0.263359
std                     0.376011                   0.440474
min                     0.000000                   0.000000
25%                     0.000000                   0.000000
50%                     0.000000                   0.000000
75%                     0.000000                   1.000000
max                     1.000000                   1.000000

       onehotencoder__cs_rep_163  pipeline-1__age  pipeline-
1__v_prem_quote  \
count               12276.000000     1.227600e+04
1.227600e+04
mean                    0.122923    -1.651913e-15
1.939001e-16
std                     0.328362     1.000041e+00
1.000041e+00
min                     0.000000    -1.838822e+00             -
2.097317e+00
25%                     0.000000    -8.517253e-01
7.557198e-02
50%                     0.000000     1.277090e-02
3.445048e-01
```

```
75%                       0.000000     7.749729e-01
5.769060e-01
max                       1.000000     2.062865e+00
2.933871e+00

       pipeline-2__tenure  functiontransformer__dl  \
count        1.227600e+04              12276.000000
mean        -2.257345e-17                  0.996986
std          1.000041e+00                  0.054819
min         -1.709077e+00                  0.000000
25%         -8.677958e-01                  1.000000
50%         -2.651448e-02                  1.000000
75%          8.628400e-01                  1.000000
max          1.764213e+00                  1.000000

       functiontransformer__has_v_insurance  \
count                          12276.000000
mean                               0.273379
std                                0.445712
min                                0.000000
25%                                0.000000
50%                                0.000000
75%                                1.000000
max                                1.000000

       functiontransformer__Region_binary
count                        12276.000000
mean                             0.520283
std                              0.499609
min                              0.000000
25%                              0.000000
50%                              1.000000
75%                              1.000000
max                              1.000000
```

We can see that the feauters were transformed successfully.

3. Model

```
# Neiral Network Model using Keras
from sklearn.utils import class_weight
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import Metric

# define the early stopping callback:
earlystop_callback = EarlyStopping(monitor='accuracy', patience=10)
```

```
# Calculate class weights based on class frequencies in the training
data
class_weights = class_weight.compute_class_weight(class_weight =
'balanced', classes = np.unique(y_train), y = y_train)

# Convert class weights to a dictionary
class_weights_dict = dict(enumerate(class_weights))

# Defining and Fitting a NNM:
model = Sequential()
model.add(Dense(50, activation='relu',name = "Layer_One"))
model.add(Dense(17, activation='relu',name = "Layer_Two"))
model.add(Dense(7, activation='relu',name = "Layer_Three"))
model.add(Dense(1, activation='sigmoid', name = "Output"))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy','AUC'])

NNM = model.fit(X_train_processed, y_train, epochs=150,
batch_size=100, validation_data=(X_test_processed, y_test),

callbacks=[earlystop_callback],class_weight=class_weights_dict)
```

```
Epoch 1/150
123/123 [==============================] - 1s 3ms/step - loss: 0.6235
- accuracy: 0.5316 - auc: 0.7110 - val_loss: 0.5265 - val_accuracy:
0.6647 - val_auc: 0.7526
Epoch 2/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5449
- accuracy: 0.6084 - auc: 0.7553 - val_loss: 0.5902 - val_accuracy:
0.5910 - val_auc: 0.7698
Epoch 3/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5360
- accuracy: 0.6022 - auc: 0.7641 - val_loss: 0.5429 - val_accuracy:
0.6019 - val_auc: 0.7746
Epoch 4/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5327
- accuracy: 0.6029 - auc: 0.7681 - val_loss: 0.5378 - val_accuracy:
0.6172 - val_auc: 0.7749
Epoch 5/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5305
- accuracy: 0.6021 - auc: 0.7697 - val_loss: 0.5704 - val_accuracy:
0.6062 - val_auc: 0.7756
Epoch 6/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5284
- accuracy: 0.6095 - auc: 0.7727 - val_loss: 0.5556 - val_accuracy:
0.6050 - val_auc: 0.7772
Epoch 7/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5285
```

```
- accuracy: 0.6078 - auc: 0.7727 - val_loss: 0.5081 - val_accuracy:
0.6493 - val_auc: 0.7789
Epoch 8/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5269
- accuracy: 0.6079 - auc: 0.7745 - val_loss: 0.5508 - val_accuracy:
0.6101 - val_auc: 0.7780
Epoch 9/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5263
- accuracy: 0.6153 - auc: 0.7753 - val_loss: 0.5947 - val_accuracy:
0.5760 - val_auc: 0.7763
Epoch 10/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5253
- accuracy: 0.6030 - auc: 0.7768 - val_loss: 0.5520 - val_accuracy:
0.6160 - val_auc: 0.7786
Epoch 11/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5246
- accuracy: 0.6153 - auc: 0.7773 - val_loss: 0.5433 - val_accuracy:
0.6263 - val_auc: 0.7765
Epoch 12/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5241
- accuracy: 0.6155 - auc: 0.7781 - val_loss: 0.5352 - val_accuracy:
0.6382 - val_auc: 0.7772
Epoch 13/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5242
- accuracy: 0.6187 - auc: 0.7773 - val_loss: 0.5630 - val_accuracy:
0.6008 - val_auc: 0.7745
Epoch 14/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5243
- accuracy: 0.6205 - auc: 0.7758 - val_loss: 0.5725 - val_accuracy:
0.6056 - val_auc: 0.7781
Epoch 15/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5219
- accuracy: 0.6166 - auc: 0.7799 - val_loss: 0.5190 - val_accuracy:
0.6518 - val_auc: 0.7794
Epoch 16/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5208
- accuracy: 0.6250 - auc: 0.7807 - val_loss: 0.6070 - val_accuracy:
0.5820 - val_auc: 0.7757
Epoch 17/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5208
- accuracy: 0.6206 - auc: 0.7809 - val_loss: 0.5400 - val_accuracy:
0.6365 - val_auc: 0.7784
Epoch 18/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5198
- accuracy: 0.6244 - auc: 0.7827 - val_loss: 0.5144 - val_accuracy:
0.6548 - val_auc: 0.7789
Epoch 19/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5189
- accuracy: 0.6256 - auc: 0.7823 - val_loss: 0.5014 - val_accuracy:
```

```
0.6695 - val_auc: 0.7795
Epoch 20/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5189
- accuracy: 0.6250 - auc: 0.7822 - val_loss: 0.5466 - val_accuracy:
0.6267 - val_auc: 0.7785
Epoch 21/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5176
- accuracy: 0.6281 - auc: 0.7835 - val_loss: 0.5608 - val_accuracy:
0.6202 - val_auc: 0.7773
Epoch 22/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5176
- accuracy: 0.6223 - auc: 0.7833 - val_loss: 0.5408 - val_accuracy:
0.6298 - val_auc: 0.7783
Epoch 23/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5188
- accuracy: 0.6285 - auc: 0.7827 - val_loss: 0.5501 - val_accuracy:
0.6257 - val_auc: 0.7769
Epoch 24/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5162
- accuracy: 0.6301 - auc: 0.7850 - val_loss: 0.5625 - val_accuracy:
0.6141 - val_auc: 0.7781
Epoch 25/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5157
- accuracy: 0.6247 - auc: 0.7854 - val_loss: 0.5446 - val_accuracy:
0.6295 - val_auc: 0.7774
Epoch 26/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5155
- accuracy: 0.6274 - auc: 0.7849 - val_loss: 0.5449 - val_accuracy:
0.6345 - val_auc: 0.7759
Epoch 27/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5149
- accuracy: 0.6300 - auc: 0.7859 - val_loss: 0.5492 - val_accuracy:
0.6248 - val_auc: 0.7781
Epoch 28/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5141
- accuracy: 0.6291 - auc: 0.7880 - val_loss: 0.5515 - val_accuracy:
0.6322 - val_auc: 0.7784
Epoch 29/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5146
- accuracy: 0.6268 - auc: 0.7861 - val_loss: 0.5428 - val_accuracy:
0.6310 - val_auc: 0.7785
Epoch 30/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5140
- accuracy: 0.6306 - auc: 0.7870 - val_loss: 0.5668 - val_accuracy:
0.6117 - val_auc: 0.7772
Epoch 31/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5134
- accuracy: 0.6273 - auc: 0.7869 - val_loss: 0.5524 - val_accuracy:
0.6355 - val_auc: 0.7781
```

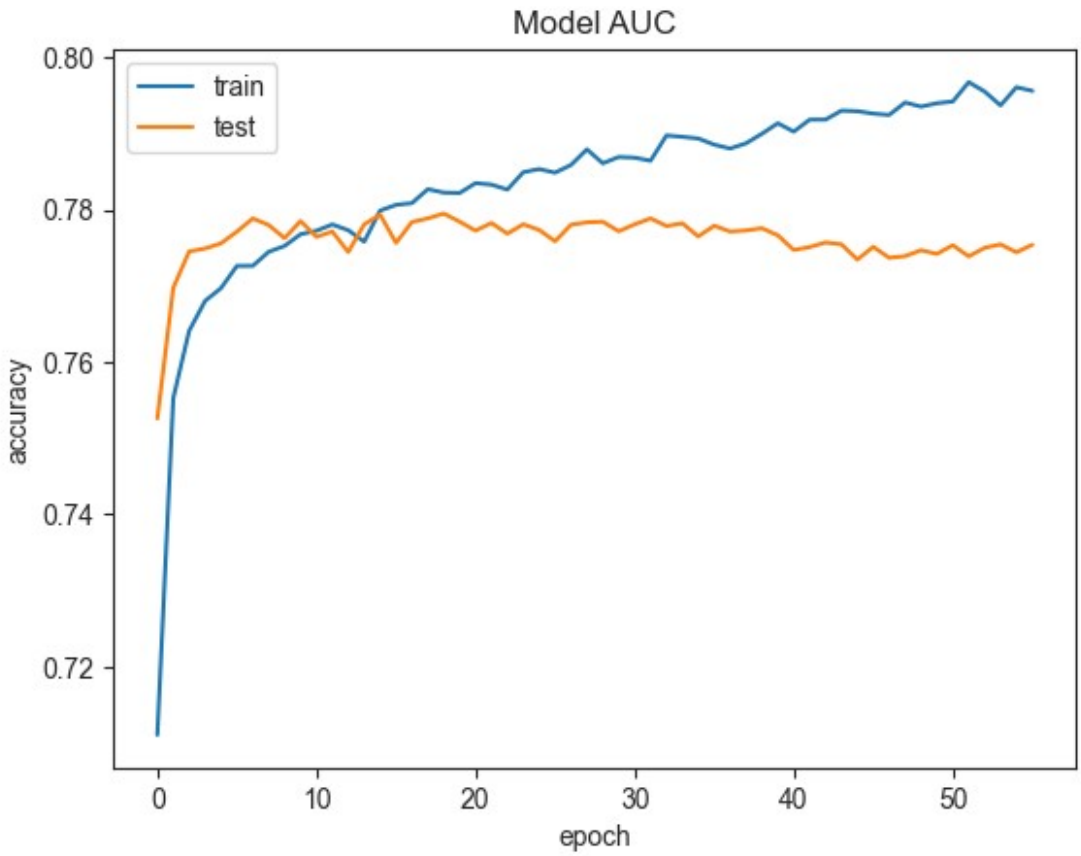
```
Epoch 32/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5139
- accuracy: 0.6301 - auc: 0.7865 - val_loss: 0.5584 - val_accuracy:
0.6238 - val_auc: 0.7789
Epoch 33/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5123
- accuracy: 0.6250 - auc: 0.7898 - val_loss: 0.5475 - val_accuracy:
0.6263 - val_auc: 0.7779
Epoch 34/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5115
- accuracy: 0.6313 - auc: 0.7896 - val_loss: 0.5463 - val_accuracy:
0.6356 - val_auc: 0.7783
Epoch 35/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5122
- accuracy: 0.6326 - auc: 0.7894 - val_loss: 0.5417 - val_accuracy:
0.6296 - val_auc: 0.7765
Epoch 36/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5112
- accuracy: 0.6303 - auc: 0.7886 - val_loss: 0.5326 - val_accuracy:
0.6432 - val_auc: 0.7779
Epoch 37/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5120
- accuracy: 0.6294 - auc: 0.7881 - val_loss: 0.5444 - val_accuracy:
0.6353 - val_auc: 0.7772
Epoch 38/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5111
- accuracy: 0.6281 - auc: 0.7888 - val_loss: 0.5722 - val_accuracy:
0.6191 - val_auc: 0.7773
Epoch 39/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5100
- accuracy: 0.6264 - auc: 0.7900 - val_loss: 0.5524 - val_accuracy:
0.6299 - val_auc: 0.7776
Epoch 40/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5090
- accuracy: 0.6339 - auc: 0.7914 - val_loss: 0.5329 - val_accuracy:
0.6398 - val_auc: 0.7767
Epoch 41/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5096
- accuracy: 0.6337 - auc: 0.7903 - val_loss: 0.5443 - val_accuracy:
0.6256 - val_auc: 0.7748
Epoch 42/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5087
- accuracy: 0.6353 - auc: 0.7919 - val_loss: 0.5460 - val_accuracy:
0.6337 - val_auc: 0.7751
Epoch 43/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5084
- accuracy: 0.6320 - auc: 0.7919 - val_loss: 0.5467 - val_accuracy:
0.6337 - val_auc: 0.7757
Epoch 44/150
```

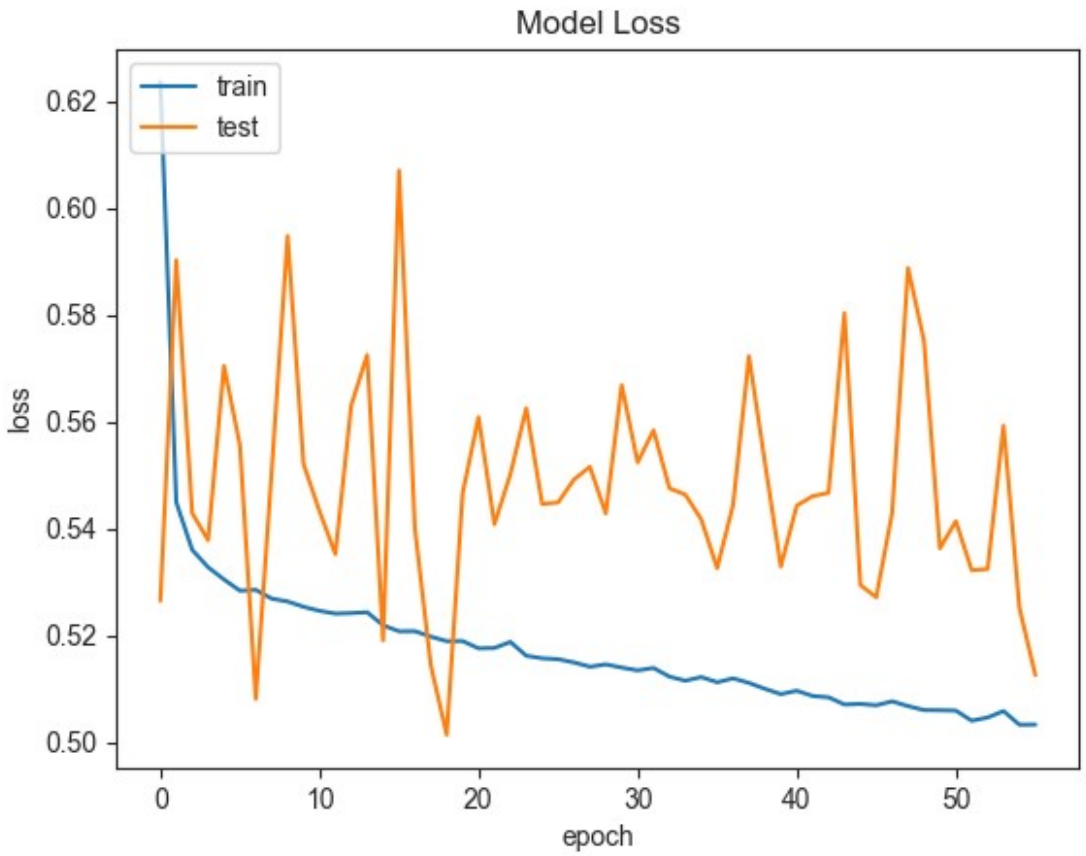
```
123/123 [==============================] - 0s 2ms/step - loss: 0.5071
- accuracy: 0.6356 - auc: 0.7931 - val_loss: 0.5803 - val_accuracy:
0.6119 - val_auc: 0.7755
Epoch 45/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5072
- accuracy: 0.6336 - auc: 0.7930 - val_loss: 0.5294 - val_accuracy:
0.6386 - val_auc: 0.7735
Epoch 46/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5069
- accuracy: 0.6394 - auc: 0.7927 - val_loss: 0.5271 - val_accuracy:
0.6459 - val_auc: 0.7752
Epoch 47/150

123/123 [==============================] - 0s 2ms/step - loss: 0.5077
- accuracy: 0.6367 - auc: 0.7925 - val_loss: 0.5430 - val_accuracy:
0.6273 - val_auc: 0.7737
Epoch 48/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5068
- accuracy: 0.6328 - auc: 0.7941 - val_loss: 0.5887 - val_accuracy:
0.5968 - val_auc: 0.7739
Epoch 49/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5060
- accuracy: 0.6332 - auc: 0.7936 - val_loss: 0.5751 - val_accuracy:
0.6124 - val_auc: 0.7747
Epoch 50/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5060
- accuracy: 0.6342 - auc: 0.7940 - val_loss: 0.5363 - val_accuracy:
0.6395 - val_auc: 0.7742
Epoch 51/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5059
- accuracy: 0.6382 - auc: 0.7943 - val_loss: 0.5414 - val_accuracy:
0.6384 - val_auc: 0.7754
Epoch 52/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5040
- accuracy: 0.6375 - auc: 0.7968 - val_loss: 0.5322 - val_accuracy:
0.6361 - val_auc: 0.7739
Epoch 53/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5046
- accuracy: 0.6338 - auc: 0.7956 - val_loss: 0.5324 - val_accuracy:
0.6349 - val_auc: 0.7750
Epoch 54/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5058
- accuracy: 0.6364 - auc: 0.7937 - val_loss: 0.5592 - val_accuracy:
0.6233 - val_auc: 0.7755
Epoch 55/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5033
- accuracy: 0.6391 - auc: 0.7961 - val_loss: 0.5251 - val_accuracy:
0.6449 - val_auc: 0.7745
Epoch 56/150
123/123 [==============================] - 0s 2ms/step - loss: 0.5033
```

```
- accuracy: 0.6391 - auc: 0.7957 - val_loss: 0.5125 - val_accuracy:
0.6612 - val_auc: 0.7754
```

```
# list all data in history
print(NNM.history.keys())
# summarize history for AUC
plt.plot(NNM.history['auc'])
plt.plot(NNM.history['val_auc'])
plt.title('Model AUC')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(NNM.history['loss'])
plt.plot(NNM.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'auc', 'val_loss', 'val_accuracy',
'val_auc'])
```


Model AUC
train
test
accuracy
epoch
0.80
0.78
0.76
0.74
0.72
0
10
20
30
40
50




```
print("Maximum AUC of the model and Max Validation AUC respectivly, "+
str(max(NNM.history['auc']))+" " +str(max(NNM.history['val_auc'])))
```

```
Maximum AUC of the model and Max Validation AUC respectivly,
0.796802818775177 0.7795366048812866
```

```
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(X_test_processed, y_test)
results
```

```
Evaluate on test data
256/256 [==============================] - 0s 793us/step - loss:
0.5125 - accuracy: 0.6612 - auc: 0.7754

[0.512546181678772, 0.6611681580543518, 0.7754406929016113]
```

```
print("Our model has AUC of " + str(round(results[2],4))  + """
An AUC Values betwenn 0.7 to 0.8 is considered acceptable, and it
means that the model makes
around 75.5% correct predictions""")
```

```
Our model has AUC of 0.7754
An AUC Values betwenn 0.7 to 0.8 is considered acceptable, and it
```

```
means that the model makes
around 75.5% correct predictions
```

Comparing F1 scores for different thresholds

```
from sklearn.metrics import f1_score

# Doing predictions using the test data:
y_pred_proba_test = model.predict(X_test_processed)

# Create an array of threshold values to try
thresholds = np.arange(0.1, 1, 0.1)

# Initialize variables to keep track of best F1 score and
corresponding threshold
best_f1_score = 0
best_threshold = 0
for threshold in thresholds:
    y_pred = (y_pred_proba_test > threshold).astype(int)
    f1 = f1_score(y_test, y_pred)
    print(f"Threshold: {threshold:.1f}, F1 score: {f1:.3f}")
```

```
256/256 [==============================] - 0s 680us/step
Threshold: 0.1, F1 score: 0.419
Threshold: 0.2, F1 score: 0.432
Threshold: 0.3, F1 score: 0.448
Threshold: 0.4, F1 score: 0.456
Threshold: 0.5, F1 score: 0.465
Threshold: 0.6, F1 score: 0.452
Threshold: 0.7, F1 score: 0.336
Threshold: 0.8, F1 score: 0.039
Threshold: 0.9, F1 score: 0.000
```

The highest F1 score is at 0.5 threshold, which is 0.465 and it measures the accuracy of the model but does not conclude that the model is useful or not as all models are wrong, but some are useful, and since we have an accceptable AUC we can conclude that our model is useful in prediction the outcome

Score Data

```
#Importing the Score data:
Score_Data = pd.read_csv("bzan6357_insurance_3_SCORE.csv")

#Checking for missing values:
Score_Data.isnull().sum()
```

```
id_new              0
age                 0
gender              0
tenure              0
```

```
region              0
dl                  0
has_v_insurance     0
v_age               0
v_accident          0
v_prem_quote        0
cs_rep              0
dtype: int64
```

```
# Transforming the region variable in the score dataset:
Region_binary_Score = []
for i in Score_Data['region']:
    if i >= 25 and i <=30:
        Region_binary_Score.append(1);
    else:
        Region_binary_Score.append(0) ;
```

```
# Droping the original region variable:
Score_Data = Score_Data.drop("region", axis =1)
```

```
#Adding the new transformed region binary variable:
Score_Data['Region_binary'] = Region_binary_Score
Score_Data
```

```
          id_new  age  gender  tenure  dl  has_v_insurance
v_age  \
0     a00020460   46    male     124   1                0   1-2 year

1     a00020461   25  female     285   1                0   < 1 year

2     a00020462   42    male      95   1                1   1-2 year

3     a00020463   39  female     229   1                0   1-2 year

4     a00020464   42    male     114   1                0   1-2 year

...         ...  ...     ...     ...  ..              ...        ...

1995  a00022455   25    male      46   1                0   < 1 year

1996  a00022456   45  female     248   1                1   1-2 year

1997  a00022457   65    male     237   1                1   1-2 year

1998  a00022458   76    male     162   1                0   1-2 year

1999  a00022459   77    male     257   1                0  > 2 years


     v_accident  v_prem_quote  cs_rep  Region_binary
0           yes       26226.0     163              0
```

```
1           yes       44564.0     154               0
2            no       30796.0     122               1
3           yes       35669.0     163               1
4           yes       32039.0     122               1
...         ...           ...     ...             ...
1995        yes       66605.0     154               0
1996         no       37984.0     122               1
1997         no       68068.0     122               1
1998        yes       36590.0     122               1
1999        yes       39476.0     122               1

[2000 rows x 11 columns]
```

```
# Preprocessing the Score data:
Processed_Score_Data =
pd.DataFrame(transformer.transform(Score_Data),columns =
transformer.get_feature_names_out(),
                                index = Score_Data.index)

Processed_Score_Data.describe()
```

```
       onehotencoder__gender_female  onehotencoder__gender_male  \
count                   2000.000000                 2000.000000
mean                       0.457500                    0.542500
std                        0.498315                    0.498315
min                        0.000000                    0.000000
25%                        0.000000                    0.000000
50%                        0.000000                    1.000000
75%                        1.000000                    1.000000
max                        1.000000                    1.000000

       onehotencoder__v_age_1-2 year  onehotencoder__v_age_< 1 year  \
count                    2000.000000                    2000.000000
mean                        0.674500                       0.253500
std                         0.468679                       0.435123
min                         0.000000                       0.000000
25%                         0.000000                       0.000000
50%                         1.000000                       0.000000
75%                         1.000000                       1.000000
max                         1.000000                       1.000000

       onehotencoder__v_age_> 2 years  onehotencoder__v_accident_no  \
count                     2000.000000                   2000.000000
mean                         0.072000                      0.318500
std                          0.258553                      0.466011
min                          0.000000                      0.000000
25%                          0.000000                      0.000000
50%                          0.000000                      0.000000
75%                          0.000000                      1.000000
max                          1.000000                      1.000000
```

```
       onehotencoder__v_accident_yes  onehotencoder__cs_rep_122  \
count                    2000.000000                2000.000000
mean                        0.681500                   0.438000
std                         0.466011                   0.496265
min                         0.000000                   0.000000
25%                         0.000000                   0.000000
50%                         1.000000                   0.000000
75%                         1.000000                   1.000000
max                         1.000000                   1.000000

       onehotencoder__cs_rep_151  onehotencoder__cs_rep_154  \
count                2000.000000                2000.000000
mean                    0.173000                   0.263500
std                     0.378342                   0.440641
min                     0.000000                   0.000000
25%                     0.000000                   0.000000
50%                     0.000000                   0.000000
75%                     0.000000                   1.000000
max                     1.000000                   1.000000

       onehotencoder__cs_rep_163  pipeline-1__age  pipeline-
1__v_prem_quote  \
count                2000.000000      2000.000000
2000.000000
mean                    0.125500        -0.011069
0.026101
std                     0.331368         0.998700
0.984974
min                     0.000000        -1.838822                   -
2.097317
25%                     0.000000        -0.851725
0.095933
50%                     0.000000         0.079461
0.350556
75%                     0.000000         0.774973
0.582199
max                     1.000000         1.997744
2.780240

       pipeline-2__tenure  functiontransformer__dl  \
count         2000.000000              2000.000000
mean             0.021937                 0.996500
std              1.000570                 0.059072
min             -1.709077                 0.000000
25%             -0.855777                 1.000000
50%              0.045595                 1.000000
75%              0.874858                 1.000000
max              1.764213                 1.000000
```

```
       functiontransformer__has_v_insurance  \
count                           2000.000000
mean                               0.290500
std                                0.454107
min                                0.000000
25%                                0.000000
50%                                0.000000
75%                                1.000000
max                                1.000000

       functiontransformer__Region_binary
count                         2000.000000
mean                             0.523500
std                              0.499572
min                              0.000000
25%                              0.000000
50%                              1.000000
75%                              1.000000
max                              1.000000
```

```
# Predicting the probabilties:
y_pred_proba = model.predict(Processed_Score_Data)

# Coverting probabilties into classes:
y_pred =[]
for i in y_pred_proba:
    if i >= 0.5:
        y_pred.append(1);
    else:
        y_pred.append(0)
```

```
63/63 [==============================] - 0s 677us/step
```

```
# Creating the final dataset
Final_predictions =
pd.DataFrame([Score_Data["id_new"],pd.Series(y_pred_proba[:,0]),pd.Ser
ies(y_pred)])
Final_predictions=Final_predictions.T
Final_predictions.columns = ['id_new', 'probability','classification']

Final_predictions
```

```
         id_new probability classification
0     a00020460    0.631623              1
1     a00020461    0.527706              1
2     a00020462    0.000706              0
3     a00020463    0.760506              1
4     a00020464    0.652414              1
...         ...         ...            ...
1995  a00022455    0.383449              0
```

```
1996  a00022456    0.000809              0
1997  a00022457    0.000015              0
1998  a00022458    0.458563              0
1999  a00022459    0.097942              0

[2000 rows x 3 columns]
```

```
Final_predictions['classification'].value_counts()
```

```
0    1076
1     924
Name: classification, dtype: int64
```

```
Final_predictions.to_csv("Final Predictions (Buy).csv", index=False)
```

Interpretation

This project has implemented a sequential Keras neural network for modeling. The network has been designed to be flexible, modular, and scalable. Furthermore, the hyperparameters of the model can be adjusted and various regularization techniques can be utilized to reduce overfitting. Additionally, data augmentation techniques can be used to increase the size of the training dataset. As a result, this project has the potential to achieve a high level of accuracy and performance.

The model consists of 4 layers:

1. Layer_One consists of 50 neurons and relu activation function.
2. Layer_Two consists of 17 neurons and relu activation function.
3. Layer_Three consists of 7 neurons and relu activation function.
4. Output layer of one neuron and sigmoid activation function.

The model was compiled with adam optimizer and binary_crossentropy for the loss.

The model was fitted to the training data and the test data was used to validate the model performance.

Recommendations

- Tune hyperparameters: Adjust the hyperparameters of the model such as learning rate, number of layers, number of neurons, etc. to improve the accuracy of the model.
- Use other regularization techniques: Implement regularization techniques such as weight decay, etc. to reduce overfitting.
- Utilize data augmentation: Utilize data augmentation techniques such as flipping, scaling, rotating, etc. to increase the size of the training dataset.

- Explore different architectures: Explore different architectures such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), etc. to improve the accuracy of the model.
- Monitor training and validation losses: Monitor the training and validation losses to identify any signs of overfitting or underfitting.
- Use learning rate scheduling: Utilize learning rate scheduling to adjust the learning rate during the training process.

Conclusion

We started by taking a look at the provided datasets and knowing more about problems with them, EDA was done on the training dataset to know about the realations between the variables, we then chose what to do with each input variable and after that data was prepared to go into the model, A NNM was defined and fitted to the prepared data, AUC and F1 scores were compared for different model parameter and using trail and error we found what we think would be a good model for predicting the outcome, After that, the score dataset was prepared and prediction were made using this dataset.